

# A Knowledge Acquisition System for Constraint-based Intelligent Tutoring Systems

Pramuditha Suraweera, Antonija Mitrovic and Brent Martin  
*Intelligent Computer Tutoring Group*  
*Department of Computer Science, University of Canterbury*  
*Private Bag 4800, Christchurch, New Zealand*  
[psu16,tanja,brent}@cosc.canterbury.ac.nz](mailto:{psu16,tanja,brent}@cosc.canterbury.ac.nz)

**Abstract.** Building a domain model consumes a major portion of the time and effort required for building an Intelligent Tutoring System. Past attempts at reducing the knowledge acquisition bottleneck by automating the knowledge acquisition process have focused on procedural tasks. We present CAS (Constraint Acquisition System), an authoring system for automatically acquiring the domain model for non-procedural as well as procedural constraint-based tutoring systems. CAS follows a four-phase approach: building a domain ontology, acquiring syntax constraint directly from it, generating semantic constraints by learning from examples and validating the generated constraints. This paper describes the knowledge acquisition process and reports on results of a preliminary evaluation. The results have been encouraging and further evaluations are planned.

## 1 Introduction

Numerous empirical studies have shown that Intelligent Tutoring Systems (ITS) are effective tools for education. However, developing an ITS is a labour intensive and time consuming process. A major portion of the development effort is spent on acquiring the domain knowledge that accounts for the intelligence of the system. Our goal is to significantly reduce the time and effort required for building a knowledge base by automating the process.

This paper details the Constraint Acquisition System (CAS), which automatically acquires the required knowledge for ITSs by learning from examples. The knowledge acquisition process consists of four phases, initiated by an expert of the domain describing the domain in terms of an ontology. Secondly, syntax constraints are automatically generated by analysing the ontology. Semantic constraints are generated in the third phase from problems and solutions provided by the author. Finally, the generated constraints are validated with the assistance of the author.

The remainder of the paper is initiated by a brief introduction to Constraint-based modelling, the student modelling technique focused in this research, and a brief overview of related research. We then present a detailed description of CAS, including its architecture and a description of the knowledge acquisition process. Finally, conclusions and future work is outlined.

## 2 Related work

Constraint based modelling (CBM) [6] is a student modelling approach that somewhat eases the knowledge acquisition bottleneck by using a more abstract representation of the domain compared to other commonly used approaches [5]. However, building constraint sets still remains a major challenge. Our goal is to significantly reduce the time and effort required for acquiring the domain knowledge for CBM tutors by automating the knowledge acquisition process. Unlike other automated knowledge acquisition systems, we aim to produce a system that has the ability to acquire knowledge for non-procedural, as well as procedural, domains.

Existing systems for automated knowledge acquisition have focused on acquiring procedural knowledge in simulated environments or highly restrictive environments. KnoMic [10] is a learning-by-observation system for acquiring procedural knowledge in a simulated environment. It generates the domain model by generalising recorded domain experts' traces. Koedinger et al have constructed a set of authoring tools that enable non AI experts to develop cognitive tutors. They allow domain experts to create "Pseudo tutors" which contain a hard coded domain model specific to the problems demonstrated by the expert [3]. Research has also been conducted to generalise the domain model of "Pseudo tutors" by using machine learning techniques [2].

Most existing systems focus on acquiring procedural knowledge by recording the domain expert's actions and generalising recorded traces using machine learning algorithms. Although these systems appear well suited to tasks where goals are achieved by performing a set of steps in a specific order, they fail to acquire knowledge for non-procedural domains, i.e. where problem-solving requires complex, non-deterministic actions in no particular order. Our goal is to develop an authoring system that can acquire procedural as well as declarative knowledge.

The domain model for CBM tutors [7] consists of a set of constraints, which are used to identify errors in student solutions. In CBM knowledge is modelled by a set of constraints that identify the set of correct solutions from the set of all possible student inputs. CBM represents knowledge as a set of ordered pairs of relevance and satisfaction conditions. The relevance condition identifies the states in which the represented concept is relevant, while the satisfaction condition identifies the subset of the relevant states in which the concept has been successfully applied.

### **3 Constraint Authoring System**

The proposed system is an extension of WETAS [4], a web-based tutoring shell that facilitates building constraint-based tutors. WETAS provides all the domain-independent components for a text-based ITS, including the user interface, pedagogical module and student modeller. The pedagogical module makes decisions based on the student model regarding problem/feedback generation, and the student modeller evaluates student solutions by comparing them to the domain model and updates the student model. The main limitation of WETAS is its lack of support for authoring the domain model.

As WETAS does not provide any assistance for developing the knowledge base, typically a knowledge base is composed using a text editor. Although the flexibility of a text editor may be adequate for knowledge engineers, novices tend to be overwhelmed by the task. The goal of CAS (Constraint Authoring System) is to reduce the complexity of the task by automating the constraint acquisition process. As a consequence the time and effort required for building constraint bases should reduce dramatically.

CAS consists of an ontology workspace, ontology checker, problem/solution manager, syntax and semantic constraint generators, and constraint validation as depicted in Figure 1. During the initial phase, the domain expert develops an ontology of the domain in the ontology workspace. This is then evaluated by the ontology checker, and the result is stored in the ontology repository.

The syntax constraints generator analyses the completed ontology and generates syntax constraints directly from it. These constraints are generated from the restrictions on attributes and relationships specified in the ontology. The resulting constraints are stored in the syntax constraints repository.

CAS induces semantic constraints during the third phase by learning from sample problems and their solutions. Prior to entering problems and sample solutions, the domain ex-

pert specifies the representation for solutions. This is a decomposition of the solution into components consisting of a list of instances of concepts. For example, an algebraic equation consists of a list of terms in the left hand and a list of terms in the right hand side.

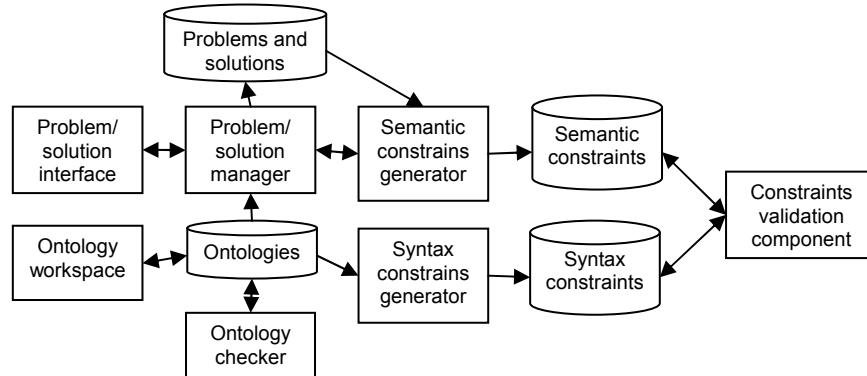


Figure 1: Architecture of the constraint-acquisition system

The final phase involves ensuring the validity of the generated constraints. During this phase the system generates examples to be validated by the author. In situations where the author’s validation conflicts with the system’s evaluation according to the domain model, the author is requested to provide further examples to illustrate the rationale behind the conflict. The new examples are then used to resolve the conflicts, and may also lead to the generation of new constraints.

### 3.1 Modelling the domain’s ontology

Domain ontologies play a central role in the knowledge acquisition process of the constraint authoring system [9]. 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 the constraints [8]. The study showed that ontologies help organise constraints into meaningful categories. This enables the author to visualise the constraint set and to reflect on the domain, assisting them to create more complete constraint bases.

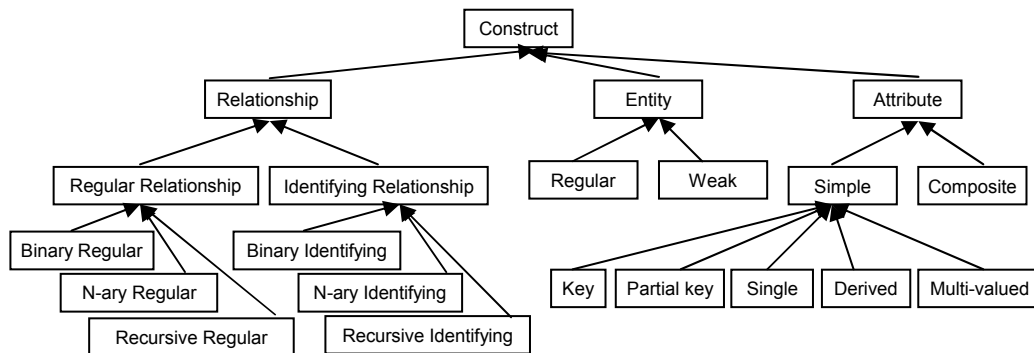


Figure 2: Ontology for ER modelling domain

An ontology describes the domain by identifying important concepts and relationships between them. It outlines the hierarchical structure of the domain in terms of sub- and super-concepts. CAS contains an ontology workspace for modelling an ontology of the domain. An example ontology for Entity Relationship Modelling is depicted in Figure 2. The root node, *Construct*, is the most general concept, of which *Relationship*, *Entity* and *Attribute* are sub-concepts. *Relationship* is further specialised into *Regular* and *Identifying*, which are the two possible types of relationships, and so on.

As syntax constraints are generated directly from the ontology, it is imperative that all relationships are correct. The ontology checker verifies that the relationships between con-

cepts are correct by engaging the user in a dialog. The author is presented with lists of specialisations of concepts involved in a relationship and is asked to label the specialisations that are incorrect. For example, consider a relationship between *Binary identifying relationship* and *Attribute*. CAS asks whether all of the specialisations of *attribute* (*key*, *partial key*, *single-valued* etc) can participate in this relationship. The user indicates that *key* and *partial key* attributes cannot be used in this relationship. CAS therefore replaces the original relationship with specialised relationships between *Binary identifying relationship* and the nodes *single-valued*, *multi-valued* and *derived*.

Ontologies are internally represented in XML. We have defined set of XML tags specifically for this project, which can be easily be transformed to a standard ontology representation form such as DAML [1]. The XML representation also includes positional and dimensional details of each concept for regenerating the layout of concepts in the ontology.

### 3.2 Syntax Constraint Generation

An ontology contains much of information about the syntax of the domain: information about domain concepts; the domains (i.e. possible values) of their properties; restrictions on how concepts participate in relationships. Restrictions on a property can be specified in terms of whether its value has to be unique or whether it has to contain a certain value. Similarly, restrictions on the participation in relationships can also be specified in terms of minimum and maximum cardinality.

The syntax constraints generator analyses the ontology and generates constraints from all the restrictions specified on properties and relationships. For example, consider the *owner* relationship between *Binary identifying relationship* and *Regular entity* from the ontology in Figure 2, which has a minimum cardinality of 1. This restriction specifies that each *Binary identifying relationship* has to have at least one *Regular entity* participating as the *owner*, and can be translated to a constraint that asserts that each *Identifying relationship* found in a solution has to have at least one *Regular entity* as its *owner*.

To evaluate the syntax constraints generator, we ran it over the ER ontology in Figure 2. It produced a total of 49 syntax constraints, covering all the syntax constraints that were manually developed for KERMIT [7], an existing constraint-based tutor for ER modelling. The generated constraint set was more specific than the constraints found in KERMIT, i.e. in some cases several constraints generated by CAS would be required to identify the problem states identified by a single constraint in KERMIT. This may mean that the set of generated constraints would be more effective for an ITS, since they would provide feedback that is more specific to a single problem state. However, it is also possible that they would be overly specific.

We also experimented with basic algebraic equations, a domain significantly different to ER modelling. The ontology for algebraic equations included only four basic operations: addition, subtraction, multiplication and division. The syntax constraints generator produced three constraints from an ontology composed for this domain, including constraints that ensure whenever an opening parenthesis is used there should be a corresponding closing parenthesis, a constant should contain a plus or minus symbol as its sign, and a constant's value should be greater than or equal to 0. Because basic algebraic expressions have very little syntax restrictions, three constraints are sufficient to impose the basic syntax rules.

### 3.3 Semantic Constraint Generation

Semantic constraints are generated by a machine learning algorithm that learns from examples. The author is required to provide several problems, with a set of correct solutions for

each depicting different ways of solving it. A solution is composed by populating each of its components by adding instances of concepts, which ensures that a solution strictly adheres to the domain ontology. Alternate solutions, which depict alternate ways of solving the problem, are composed by modifying the first solution. The author can transform the first solution into the desired alternative by adding, editing or dropping elements. This reduces the amount of effort required for composing alternate solutions, as most alternatives are similar. It also enables the system to correctly identify matching elements in two alternate solutions.

The algorithm generates semantic constraints by analysing pairs of solutions to identify similarities and differences between them. The constraints generated from a pair of solutions contribute towards either generalising or specialising constraints in the main constraint base. The detailed algorithm is given in Figure 3.

- |  |
|--|
| <ul style="list-style-type: none"> <li>a. For each problem <math>P_i</math></li> <li>b. For each pair of solutions <math>S_i</math> &amp; <math>S_j</math> <ul style="list-style-type: none"> <li>a. Generate a set of new constraints <math>N</math></li> <li>b. Evaluate each constraint <math>CB_i</math> in main constraint base, <math>CB</math>, against <math>S_i</math> &amp; <math>S_j</math><br/>If <math>CB_i</math> is violated, generalise or specialise <math>CB_i</math> to satisfy <math>S_i</math> &amp; <math>S_j</math></li> <li>c. Evaluate each constraint <math>N_i</math> in set <math>N</math> against each previously analysed pair of solutions <math>S_x</math> &amp; <math>S_y</math> for each previously analysed problem <math>P_z</math>,<br/>If <math>N_i</math> is violated, generalise or specialise <math>CB_i</math> to satisfy <math>S_x</math> &amp; <math>S_y</math></li> <li>d. Add constraints in <math>N</math> that were not involved in generalisation or specialisation to <math>CB</math></li> </ul> </li> </ul> |
|--|

Figure 3: Semantic constraint generation algorithm

The constraint learning algorithm focuses on a single problem at a time. Constraints are generated by comparing one solution to another of the same problem, where all permutations of solution pairs, including solutions compared to themselves, are analysed. Each solution pair is evaluated against all constraints in the main constraint base. Any that are violated are either specialised to be irrelevant for the particular pair of solutions, or generalised to satisfy that pair of solutions. Once no constraint in the main constraint base is violated by the solution pair, the newly generated set of constraints is evaluated against all previously analysed pairs of solutions. The violated constraints from this new set are also either specialised or generalised in order to be satisfied. Finally, constraints in the new set that are not found in the main constraint base are added to the constraint base.

- |  |
|--|
| <ul style="list-style-type: none"> <li>1. Treat <math>S_i</math> as the ideal solution (IS) and <math>S_j</math> as the student solution (SS)</li> <li>2. For each element <math>A</math> in the IS <ul style="list-style-type: none"> <li>a. Generate a constraint that asserts that if IS contains the element <math>A</math>, SS should contain a matching element</li> <li>b. For each relationship that element is involved with,<br/>Generate constraints that ensures that the relationship holds between the corresponding elements of the SS</li> </ul> </li> <li>3. Generalise the properties of similar constraints by introducing variables or wild cards</li> </ul> |
|--|

Figure 4: Algorithm for generating constraints from a pair of solutions

New constraints are generated from a pair of solutions following the algorithm outlined in Figure 4. It treats one solution as the ideal solution and the other as the student solution. A constraint is generated for each element in the ideal solution, asserting that if the ideal solution contains the particular element, the student solution should also contain the matching element.

- E.g.      Relevance: IS.Entities has a Regular entity  
              Satisfaction: SS.Entities has a Regular entity

In addition, three constraints are generated for each relationship that an element participates with. Two constraints ensure that a matching element exists in SS for each of the two elements of IS participating in the relationship. The third constraint ensures that the relationship holds between the two corresponding elements of SS.

- E.g. 1. Relevance: IS.Entities has a Regular entity  
 AND IS.Attributes has a Key  
 AND SS.Entities has a Regular entity  
 AND IS Regular entity is in *key-attribute* with Key  
 AND IS Key is in *belong to* with Regular entity  
 Satisfaction: SS.Attributes has a Key
2. Relevance: IS.Entities has a Regular entity  
 AND IS.Attributes has a Key  
 AND SS.Attributes has a Key  
 AND IS Regular entity is in *key-attribute* with Key  
 AND IS Key is in *belong to* with Regular entity  
 Satisfaction: SS.Entities has a Regular entity
3. Relevance: IS.Entities has a Regular entity  
 AND IS.Attributes has a Key  
 AND SS.Entities has a Regular entity  
 AND SS.Attributes has a Key  
 AND IS Regular entity is in *key-attribute* with Key  
 AND IS Key is in *belong to* with Regular entity  
 Satisfaction: SS Regular entity is in *key-attribute* with Key  
 AND SS Key is in *belong to* with Regular entity

- a. If constraint set, C-set that does not contain violated constraint V, has a similar but a more restrictive constraint C then replace V with C and exit.
- b. If C-set has a constraint C that has the same relevance condition but different satisfaction condition to V,  
 Add the satisfaction condition of C as a disjunctive test to the satisfaction of V, remove C from C-set and exit
- c. Find a solution  $S_k$  that satisfies constraint V
- d. If a matching element can be found in  $S_j$  for each element in  $S_k$  that appears in the satisfaction condition,  
 Generalise satisfaction of V to include the matching elements as a new test with a disjunction and exit
- e. Restrict the relevance condition of V to be irrelevant for solution pair  $S_i$  &  $S_j$ , by adding a new test to the relevance signifying the difference and exit
- f. Drop constraint

Figure 5: Algorithm for generalising or specialising violated constraints

The constraints that get violated during the evaluation stage are either specialised or generalised according to the algorithm outlined in Figure 5. It deals with two sets of constraints (C-set): the new set of constraints generated by a pair of solutions and the main constraint base. The algorithm remedies each violated constraint individually by either specialising it or generalising it. If the constraint cannot be resolved, it is labelled as an incorrect constraint and the system ensures that it does not get generated in the future.

The semantic constraints generator of CAS produced a total of 151 constraints for the domain of ER modelling using the ontology in Figure 2 and six problems. The problems supplied to the system were simple and similar to the basic problems offered by KERMIT. Each problem focused on a set of ER modelling constructs and contained at least two solutions that exemplified alternate ways of solving the problem. The solutions were selected that maximised the differences between them. The differences between most solutions were small because ER modelling is a domain that does not have vastly different solutions. How-

ever, problems that can be solved in different ways consisted of significantly different solutions.

The generated constraints covered 85% of those found in KERMIT's constraint-base, which was built entirely manually and has proven to be effective. After further analysing the generated constraints, it was evident that the reason for not generating most of the missing constraints was due to a lack of examples. 85% coverage is very encouraging, considering the small set of sample problems and solutions. It is likely that providing further sample problems and solutions to CAS would increase the completeness of the generated domain model. Although the problems and solutions were specifically chosen to improve the system's effectiveness in producing semantic constraints, we assume that a domain expert would also have the ability to select good problems and provide solutions that show different ways of solving a problem. Moreover, the validation phase, which is yet to be completed, would also produce constraints with the assistance of the domain expert.

CAS also produced some modifications to existing constraints found in KERMIT, which improved the system's ability to handle alternate solutions. For example, although the constraints in KERMIT allowed weak entities to be modelled as composite multivalued attributes, in KERMIT the attributes of weak entities were required to be of the same type as the ideal solutions. However CAS correctly identified that when a weak entity is represented as a composite multivalued attribute, the partial key of the weak entity has to be modelled as simple attributes of the composite attribute. Furthermore, the identifying relationship essential for the weak entity becomes obsolete. These two examples illustrate how CAS improved upon the original domain model of KERMIT.

We also evaluated the algorithm in the domain of algebraic equations. The task involved specifying an equation for the given textual description. As an example, consider the problem "When I went to the shop to buy two loafs of bread, I gave the shopkeeper a \$5 note and he gave me \$1 as change. Write an expression to find the price of a loaf of bread using  $x$  to represent the price". It can be represented as  $2x + 1 = 5$  or  $2x = 5 - 1$ . In order to avoid the need for a problem solver, the answers were restricted to not include any simplified equations. For example the solution " $x = 2$ " would not be accepted because it is simplified.

- |  |
|--|
| <p>a) Relevance: IS LHS has a Constant (?Var1)<br/> Satisfaction: SS LHS has a Constant (?Var1)<br/> or SS RHS has a Constant (?Var1)</p> <p>b) Relevance: IS RHS has a +<br/> Satisfaction: SS LHS has a -<br/> or SS RHS has a +</p> <p>c) Relevance: IS RHS has a Constant(?Var1)<br/> and IS RHS has a -<br/> and SS LHS has a Constant(?Var1)<br/> and SS LHS has a +<br/> and IS Constant (?Var1) is in Associated-operator with -<br/> Satisfaction: SS Constant (?Var1) is in Associated-operator with +</p> |
|--|

Figure 6: Sample constraints generated for Algebra

The system was given five problems and their solutions involving addition, subtraction, division and multiplication for learning semantic constraints. Each problem contained three or four alternate solutions. CAS produced a total of 80 constraints. Although the completeness of the generated constraints is yet to be formally evaluated, a preliminary assessment revealed that the generated constraints are able to identify correct solutions and point out many errors. Some generated constraints are shown in Figure 6. An algebraic equation consists of two parts: a left hand side (LHS) and a right hand side (RHS). Constraint *a* in Figure 6 specifies that for each constant found in the LHS of the Ideal solution (IS), there has to be

an equal constant in either the LHS or the student solution (SS) or the RHS. Similarly, constraint  $b$  specifies that an addition symbol found in the RHS of the IS should exist in the SS as either an addition symbol in the same side or a subtraction in the opposite side. Constraint  $c$  ensures the existence of the relationship between the operators and the constants. Thus, a constant in the RHS of the IS with a subtraction attached to it, can appear as a constant with addition attached to it in the LHS of the SS.

#### 4 Conclusions and Future work

We provided an overview of CAS, an authoring system that automatically acquires the constraints required for building constraint-based Intelligent Tutoring Systems. It follows a four-stage process: modelling a domain ontology, extracting syntax constraints from the ontology, generating semantic constraints and finally validating the generated constraints.

We undertook a preliminary evaluation in two domains: ER modelling and algebra word problems. The domain model generated by CAS for ER modelling covered all syntax constraints and 85% of the semantic constraints found in KERMIT [7] and unearthed some discrepancies in KERMIT's constraint base. The results are encouraging, since the constraints were produced by analysing only 6 problems. CAS was also used to produce constraints for the domain of algebraic word problems. Although the generated constraints have not been formally analysed for their completeness, it is encouraging that CAS is able to handle two vastly different domains.

Currently the first three phases of the constraints acquisition process have been completed. We are currently developing the constraint validation component, which would also contribute towards increasing the quality of the generated constraint base. We also will be enhancing the ontology workspace of CAS to handle procedural domains. Finally, the effectiveness of CAS and its ability to scale to domains with large constraint bases has to be empirically evaluated in a wide range of domains.

#### References

- [1] DAML. DARPA Agent Markup Language, <http://www.daml.org>.
- [2] Jarvis, M., Nuzzo-Jones, G. and Heffernan, N., *Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems*. In: Lester, J., et al. (eds.) Proc. ITS 2004, Maceio, Brazil, Springer, pp. 541-553, 2004.
- [3] Koedinger, K., et al., *Opening the Door to Non-programmers: Authoring Intelligent Tutor Behavior by Demonstration*. In: Lester, J., et al. (eds.) Proc. ITS 2004, Maceio, Brazil, Springer, pp. 162-174, 2004.
- [4] Martin, B. and Mitrovic, A., *WETAS: a Web-Based Authoring System for Constraint-Based ITS*. Proc. 2nd Int. Conf on Adaptive Hypermedia and Adaptive Web-based Systems AH 2002, Malaga, Spain, LCNS, pp. 543-546, 2002.
- [5] Mitrovic, A., Koedinger, K. and Martin, B., *A comparative analysis of cognitive tutoring and constraint-based modeling*. In: Brusilovsky, P., et al. (eds.) Proc. 9th International conference on User Modelling UM2003, Pittsburgh, USA, Springer-Verlag, pp. 313-322, 2003.
- [6] Ohlsson, S., *Constraint-based Student Modelling*. Proc. Student Modelling: the Key to Individualized Knowledge-based Instruction, Berlin, Springer-Verlag, pp. 167-189, 1994.
- [7] Suraweera, P. and Mitrovic, A. *An Intelligent Tutoring System for Entity Relationship Modelling*. Int. J. Artificial Intelligence in Education, vol 14 (3,4), 2004, pp. 375-417.
- [8] Suraweera, P., Mitrovic, A. and Martin, B., *The role of domain ontology in knowledge acquisition for ITSs*. In: Lester, J., et al. (eds.) Proc. Intelligent Tutoring Systems 2004, Maceio, Brazil, Springer, pp. 207-216, 2004.
- [9] Suraweera, P., Mitrovic, A. and Martin, B., *The use of ontologies in ITS domain knowledge authoring*. In: Mostow, J. and Tedesco, P. (eds.) Proc. 2nd Int. 2nd International Workshop on Applications of Semantic Web for E-learning SWEL'04, ITS2004, Maceio, Brazil, pp. 41-49, 2004.
- [10] van Lent, M. and Laird, J.E., *Learning Procedural Knowledge through Observation*. Proc. International conference on Knowledge capture, pp. 179-186, 2001.