

Ontology-Based Knowledge Representation for a Domain-Independent Problem-Solving ITS Framework

Jean-François Lebeau, Mikaël Fortin, Amir Abdessemed and André Mayers

Département d'informatique, Université de Sherbrooke, Québec, Canada
{jean-francois.lebeau2, andre.mayers}@usherbrooke.ca

Abstract. In this paper, we present an original knowledge representation approach dedicated to ITS. It combines a strict ontological representation for authoring purposes and an ontology-based knowledge base used in domain-independent problem-solving. This approach benefits from recent developments in ontology engineering to propose an alternative to established formalisms. We explain the advantages of using this approach in the ASTUS framework through examples drawn from toy and real domains which illustrate both the formalism and its use in the framework.

Keywords: Knowledge Representation, Problem-Solving ITS Framework, Ontology-Based Knowledge Base.

1 Introduction

Among Intelligent Tutoring Systems (ITS), systems teaching factual knowledge can be discerned from systems teaching how-to knowledge. From the latter ones (problem-solving ITS), two different approaches can be distinguished: constraint-based modeling and cognitive modeling [12]. VanLehn recently proposed a taxonomy and a synthesis of the behavior [1] shared by most of the ITS based on a cognitive model. It can be summarized as an outer loop iterating at the task, or problem, level and an inner loop iterating at the step level, each step corresponding to an action in the learning environment (LE). Steps result from *learning events* or inferences¹, which correspond to the mental application of a knowledge component. Both the Cognitive Tutors [5], including their most recent derivate, CTAT[3], and VanLehn's Andes [11] fit this behavioral model of problem-solving ITS. As the ASTUS framework presented in this paper also follows this model, "ITS" is then used to denote such systems. In ITS built around a cognitive model, it is usual to have different types of knowledge components for both declarative and procedural knowledge. The latter embodies the skills the system is helping the learners master and the former represents facts manipulated by their procedural counterparts. As said by Anderson et al. [5], the task domain notions are limited to their *compiled* procedural form in ITS. For example, one would find, in a geometry task domain model, a procedure to calculate the perimeter of a triangle, but not a formal or

¹ "inference" was the term used in a draft version of [1] previously available online.

teachable definition of it. As expert systems have an historical influence on ITS, production rules are the usual formalism for procedural knowledge representation and declarative knowledge is usually represented by facts [3, 5] or more recently, with object-oriented formalisms [44]. Another distinction is that ITS can be domain specific, like Andes, or built from a domain-independent framework, like ITS built using CTAT.

In this paper, we show that an ontology-based approach has many advantages in representing declarative knowledge components in a domain-independent ITS framework. To that end, we need to put an emphasis on the role played by the knowledge components in ITS: enabling the system to understand and help the learner, not aiming for more efficient problem-solving. This is particularly relevant for declarative knowledge, its role being precisely to support the procedural knowledge by representing what the learner sees, manipulates and inputs in the learning environment. Our hypothesis follows the idea, stated by Mizoguchi [2], that while traditional knowledge bases fit problem-solving AI needs, ontologies are more suited to fulfill a supportive role that he defines as an “Intelligence Amplifier”. As an ontology helps a human being doing a request on the Semantic Web, it will support the system’s intelligent processes.

In the next sections, we describe our global approach; show how we use ontology-based semantic knowledge components in our framework; present different examples of them drawn from toy and real domains, discuss how an ontology-based approach helps the resulting ITS intelligent processes and finally conclude on the advantages of this approach that can lead to future works.

2 The ASTUS Framework

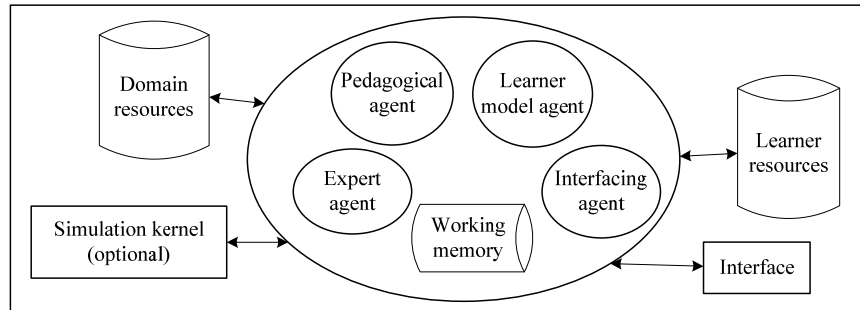


Fig. 1. The ASTUS framework. The ellipse encompasses the domain-independent components.

The ASTUS framework is based on the archetypal, four-modules ITS architecture [8] (see Fig. 1). The expert agent’s (EA) main jobs are step generation and plan recognition (see section 2.1). EA can also be used by authors to test the modeled expertise against encoded problems. The *learner model agent* (LA) is responsible for

the learner' assessment, estimating his capacities before and after his steps, and deducing the inferences he derives them from (when plan recognition faces ambiguities). The pedagogical agent (PA) uses information from both EA and LA to decide when and how to provide feedback to the learner. The interfacing agent (IA) carries out these feedbacks which range from simple messages to sophisticated manipulations of the LE. However, its main jobs are step recognition from the learner's actions and knowledge communication in the LE [6]. The agents, implemented as Java processes, can communicate with each other directly or by accessing a shared working memory (WM) built over the Jess rule-based engine². For example, the plan representing the learner's progress built by EA is stored in the WM where it can be augmented by LA, updated by IA and used as a criterion for PA decision making. A *lab* is task domain package combining the represented knowledge, the LE (user interface and simulation kernel), the problems, and others domains' or learners' specific data (domain and learner resources).

2.1 Expert Agent and Knowledge Representation

To understand our usage of ontology-based declarative knowledge components, one must see how these components will be manipulated by the procedural ones. Our knowledge representation model comes from preliminary works on the use of *Miace*[9], a cognitive architecture drawing largely from ACT-R[13], but proposing some original twists, in the context of ITS. A previous implementation of this model was used to simulate the reduction of Boolean expressions [10], but was not integrated into a complete ITS. The ASTUS model divides declarative knowledge in both semantic (factual) and episodic (autobiographical) knowledge [14]. Semantic components are detailed in the next section while the episodic components are simply defined here as results from the application of procedural knowledge. In the current implementation of this model, three different basic types of procedural knowledge components are used.

First, *complex procedures* are mental plans that produce a set of goals, a special semantic component that represent an intention (not a state), according to a scripted behavior (a sequence, an iteration or a selection). Second, *primitive procedures* are atomic actions that represent mastered skills which manipulate the LE, e.g. typing the sum of two numbers; they correspond to steps in VanLehn's lexicon [1]. Third, *queries and production rules* represent basic or mastered skill which occurs mentally, such as pattern-matching and mental arithmetic. Along with complex procedures, they fit VanLehn's definition of inferences. A problem is associated with an overall goal that can be satisfied by different procedures, including some erroneous ones. As complex procedures specify sub-goals, the resulting graph has a root goal and a set of primitive procedures as its leaves. For example, the multi-columns addition task domain can be basically modeled with two complex procedures; the root goal *G_AddColumns* is satisfied by a first complex procedure *CP_AddColumns* that queries the columns and iterate over it, creating the required number of *G_AddColumn* sub-goals. The latter are satisfied by *CP_AddColumn*, a complex procedure which

² More on Jess can be found at <http://herzberg.ca.sandia.gov/>

behavior is a relaxed sequence of two sub-goals, each depending on a query result (see Fig. 2). In the next section, we show how the columns, sums, carry digits, etc. are represented.

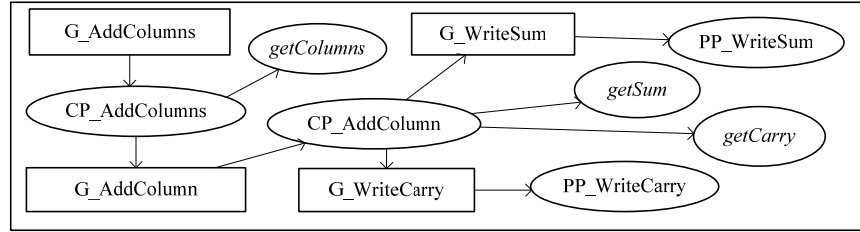


Fig. 2. A simplified procedural knowledge graph for the multi-columns addition problem. Usually, queries (in *italic*) are not visually represented but are added here for clarity.

3 Ontology-Based Semantic Knowledge

To further develop the addition task domain example, we introduce our types of semantic knowledge components: *chunks* and *attributes* are the low-level tools used to build *concepts*, *relations*, *functions* and *contexts*. We use the common term “chunk” to describe all the semantic knowledge components we assume are stored in long-term memory [14]. Concepts represent abstract or tangible, atomic or complex notions. Functions and relations associate *instances* of those concepts. Compared to the Hozo ontological framework [7], most of the chunks correspond to *basic concepts*, but relations or functions can act like *role concepts*. Chunks are described both with “is-a” relationships and attributes with values containing either primitive data or *links* to other chunks’ instance(s). While attributes are used to represent the most defining features of a chunk, like a column’s operands, functions and relations represent more subtle associations, like a sum function instance associating a column and a digit instance. Attributes play different roles depending in which chunks they are used, they define: *properties* or *parts* in concepts, *arguments* and *image* in functions and “*tuples*” in relations. Attributes can be shared among chunks coming from different taxonomies. Also, attributes’ range and domain can be restricted at the chunk level. Integrity constraints can be set on links to enforce transitive part/whole relationships. For example, removing a column would trigger the removal of its operands from the knowledge base (see section 3.2). At the function or relation level, we adopt a closed-world assumption. However, at the attribute level, an open-world assumption is adopted through the meaning we give to an *unknown value*. A value must exist even if it is not currently known in the knowledge base. Instances with unknown values usually represent empty forms a learner has to fill to produce his solution. An attribute can link an instance to a *set of instances* when the *multi-valued* aspect is defining. For example, an *addition* can be defined as a set of columns to add (see Fig. 3).

Unlike with Description Logics (DL) [15] or OWL [16], within the ASTUS framework, it is not possible to reason at the chunk (DL-concept/OWL-class) level at *runtime* (when the problem is being solved). A previous ASTUS implementation [10]

used OWL as a foundation; however, we now consider that it is more suitable to use a generated OWL ontology during the task domain authoring process (see section 3.3). At runtime, our current knowledge representation architecture limits itself to identify which chunks an instance depicts. To achieve this, a concept can be associated to a *classification rule* instead of a set of attributes; the knowledge base will then assert the appropriate “instance-of” relationships when the rule’s conditions are met. Domain-specific forward-chaining inference can be realized by associating *instantiation rules* to both relations and functions. For the first, the rule looks for relevant instances in the knowledge base that verify the relation and for the second, the rule can either find or create the image value (an instance / set of instances) that corresponds to the arguments. Even if function or relation instances are mostly created this way, they can be asserted directly like concept instances. Coupled with queries, the classification rules which represent pattern-recognition and similar skills along with instantiations rules, implement the third basic type of procedural knowledge components (see section 2.1). To represent constants of the task domain such as decimal digits (see Fig. 3), a specific knowledge component is needed. While range restrictions can be applied on attributes at the chunk level, attributes’ values are set at the instance level. *Static instances*, which attributes’ values are immutable, are modeled at the same level than chunks and attributes.

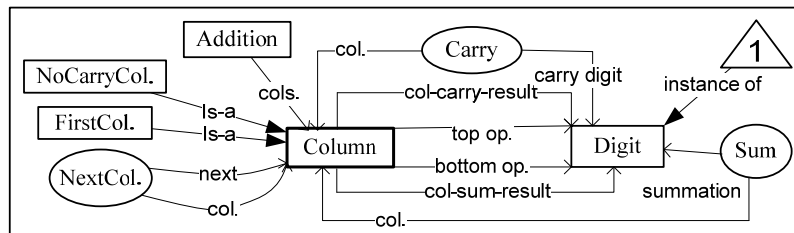


Fig. 3. A simplified semantic knowledge graph for the multi-columns addition task domain. Rectangles are concepts, ellipses are relations or functions and triangles are static instances. Link-attributes are shown with empty head arrows, e.g. *next* is an attribute of the NextColumn function. Primitive-valued attributes are not shown. For example, a digit has an integer *value*.

3.1 Usage in Procedural Knowledge Components

Intentions of goals and applicability of procedures are specified with parameters and queries. They are referred in the chosen procedure (from a goal) or in a sub-goal (from a complex procedure). Queries find instances in the KB according to domain-independent requests such as “get the unique instance of a concept” or “get the image value of a function”. They are specified with parameters and previous queries as they are executed sequentially. Both queries and parameters specify a chunk that must be depicted by the argument (instance/set of instances) at runtime, like a type matched by an argument in typed programming languages. For example, *G_AddColumn* has a *<Column c>* parameter that receives its argument from the execution of *CP_AddColumns*. This complex procedure queries all the column instances linked to its *<Addition a>* parameter. This parameter will receive its argument from *G_AddColumns*, the root goal that first queries it (see Fig. 2).

3.2 The Knowledge Base and Contexts

The knowledge base (KB) is a fundamental part of the system's WM; it includes all the instances and the rules testing and updating it. Every instance added to the KB is associated with a context's instance. Contexts are special chunks because we assume that the learner is not consciously representing them like the others. Contexts are a way to represent as semantic knowledge components the different sub-problems and tools needed to solve them in a given task domain. Usually, the LE user interface presents a different kind of window for each context. For example, in the addition task domain a dialog reifying the addition of a column with the counting method would be represented as a context separated from the main context window. Each task domain contains at least one domain-specific context, whereas some may define many of them or require many instances of the same. In order to organize the instances, we need a domain-independent hierarchy of contexts.

Thus, three special contexts are created during KB initialization: the *ontological context*, which contains the static instances; the *reasoning context* which includes the instances created by instantiation rules and the *meta-context* which holds all the context instances. Attributes in contexts play two different roles: *anchor points* that define the top-level instances and *bridges* that specify the connection between two context's instances. For example, in the addition task domain, the sole *addition's* concept instance is linked to the main context by an anchor point and a column's instance would act as a bridge between the main and the counting method contexts. The KB is directly modified by *kb scripts* that are associated to primitive procedures, context transitions and context initializations. The latter is usually the most important source of live³ instances. For example, the addition task domain's main context initialization kb script creates the addition and columns instances.

3.3 OWL Ontology Generation

Using Pellet reasoning API [18], we generate an OWL ontology from a basic template containing ASTUS's top-level ontology (chunk class subsuming concept and such). The domain-ontology is formed by concepts, relations, functions and contexts as classes; attributes as properties (roles in DL) and static instances as individuals. This development-time parallel representation of the task domain serves two purposes: to detect inconsistencies, such as concepts from which it is logically impossible to create instances and to discover, through reasoning, the complete taxonomy. Thanks to Pellet reasoning services, we can avoid (as much as DL-based representations makes it possible) the classic problems of simple/multiple-inheritance without implementing mechanisms to verify the logical soundness of our XML encoded semantic knowledge components. We visualize the generated ontology with Protégé-OWL [17]. We rejected using SWRL⁴ rules to augment the OWL reasoning offered by Pellet, because while they are more formal than Jess rules, their interpretation by higher-level processes such as PA would still be difficult.

³ "dynamic" is not used, because live instances, even if created at runtime, can be immutable.

⁴ W3C Member submission for SWRL : <http://www.w3.org/Submission/SWRL/>

4 Modeling Semantic Knowledge Components

Currently, four toy task domain labs have been developed and two classroom-ready ones are functional but still in development. The first are: the addition lab described the previous section, a fraction addition lab, a block-world lab and a trivial apparatus lab that stands for task domains involving machine operation through simulation; the second are: a genetics lab that provides the simulation tools to exercise a DNA analysis method and a hexadecimal subtraction lab. Both are planned for experimentation with undergraduate students.

4.1 KB as Model (in MVC Design Pattern)

The instance level is a recent addition to the ASTUS model. Its first motivation is to replace most of the LE domain-specific data structures that were needed in the previous implementations. Therefore, the KB can be considered as the *model* of the Model-View-Controller⁵ design pattern while the IA is responsible for the *viewing* mechanisms. The *controllers* are divided into low-level ones (such as UI event handlers) handled by IA while the KB and kb scripts are the high-level ones [6]. Instances represent both given elements and elements created by the learner to build his solution; as such, in the fraction lab there are two fractions operands and three forms (equivalent fraction, intermediate sum and final answer). Other instances, such as the various possible common denominators (*cd*) in the fraction lab, are never shown; they exist only in the reasoning context. This means that different chunk-instance pairs represent the mental calculation result and the learner's typed answer. In some labs, instances are insufficient to model the LE; a simulation kernel is added, associating relevant instances to a domain-specific behavior. For example, a button in the apparatus lab needed to be pressed a variable number of times before an indicator's color changes, or the calculations to place DNA fragment instances on a given "agarose gel" instance in the genetics lab⁶.

4.2 Dealing with Learner's Inputs

IA gives meaning to the data input by the learner, by building instances from it. EA's role, through the KB, is to evaluate those instances. *Qualification rules* and *comparators* (simple ones are built-in) are used to do so. Those rules create appropriate "instance of" relationships. For example "24" is a *valid cd* for "1/8" and "3/4" but is neither the *least cd* nor the *denominators-product cd*. While these rules provide a method to handle various situations, including inexact measures evaluation in our genetics lab, comparators handle simple cases, like exact comparison. For example, in the "duplicate stack" problem of the block-world lab, the learner's created blocks' name and color attributes' values can be compared to the values of the original blocks. The same applies to the sum-result of a column in the addition lab.

⁵ Sun takes on MVC at <http://java.sun.com/blueprints/patterns/MVC-detailed.html>

⁶ Readers can have a glimpse of this method at <http://www.life.uiuc.edu/molbio/geldigest/electro.html>

4.3 Block-World Knowledge Base for a Simple Problem

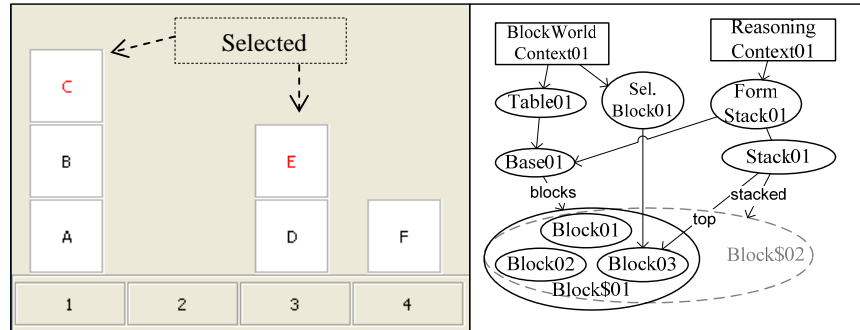


Fig. 4. A screenshot of the block-world lab user interface and a partial view of the KB.

The learner is asked to switch the position of two selected blocks (see Fig. 4), under the usual constraint that only top blocks can be moved, one at the time. During the block-world context initialization, many instances are created: a table, its four bases, a set of block instances for each of them, six blocks and two *selected block* relation instances. Other instances are created by rules, such as a *stack* and a *stack height* for each *occupied base*, which is a concept depicted by three of the four bases in Fig. 4. Both *blocks* and *free bases* are *surfaces* on which blocks can be moved. A distinct set of instances (Block\$02 in Fig. 4) is created by the instantiation rule of the *form stack* function because its image is a mental construction that has a specific meaning (a stack) which the existing set (Block\$01 in Fig. 4) does not have.

In the “duplicate stack” problem, the learner has to build a stack using new blocks created in a different context shown over the block-world. To create those blocks, he must: type a name, select a color, and then press a button. The instance that represents its input, a *block blueprint* created by IA, is sent as an argument to the kb script that creates the corresponding block and adds it to the KB. Thus, as mental results are distinguished from typed answers, input data and their effect in the KB are also distinct.

5 The Advantages of an Ontology-Based Approach

We can summarize the ASTUS approach as the combination of a supporting OWL-DL ontology at authoring time and an ontology-based KB at runtime. The first helps the authors formalizing the notions required to solve problems in the task domain, whereas the second intrinsically helps the system solving process, but more importantly its recognition process done by IA and EA. Ontology engineering is about standardization, reusability and sharing. Keeping the two last aspects for the conclusion, we now discuss the advantages for a domain-independent ITS framework to work with standardized models of the task domains. To do so, we need to explain key ideas of our knowledge representation approach. First and foremost, task domains

modeled for the ASTUS framework must embrace a specific ontological commitment: the ontology is a pedagogical reflection of the domain and must follow cognitive modeling constraints drawn from works in cognitive psychology and architectures. Second, our approach for semantic knowledge components illustrates our global strategy for all the knowledge components. It can be described with three fundamental principles: 1) knowledge components must have a single, precise meaning; 2) knowledge components are glass-box formalisms when they are of pedagogical interest, and efficient black-box formalisms when they are not; 3) knowledge components must facilitate their interpretation by the higher-level processes.

All agents benefit from the ontology-based approach, but we will focus on the advantages for EA as it is the most developed agent (IA was treated in [6]). The recent inclusion of reasoning at the semantic knowledge components level enables a clear distinction between skills that we think should be handled directly at the tutoring level; for example, the *CP_AddColumn* mental plan, the *sum* function mental calculation/retrieval and the *PP_WriteSum* action on the LE. This is a distinction not seen in other ITS. As the applicability of procedures relies on chunks and instances taxonomy, the flexibility of OWL-DL subsumption relationships frees us from awkward constraints of “is-a” relationships. We offer a technique to deal with instances added in the KB: both the learner’s input which must be evaluated, and the problem’s specific instances that are augmented. This augmentation can be taxonomy updates or rule-instantiated functions or relations that include instances input by IA or created by kb scripts. This is crucial to keep the size of domain-specific kb scripts manageable. It would be tedious to declare each couple of *same height* blocks relation in a block-world lab initialization script. Developing a domain-independent LA or PA is the overall goal of our global approach, ontology-based semantic knowledge components being an important part of it. For LA, precise and examinable knowledge components lead to a more reliable modeling of the learner’s mental efforts. For PA, it allows to generate simple, but precise communication content in many situations where other ITS rely on domain-specific messages.

As we don’t use DL-based representation and reasoning at runtime, we avoid performance issues; however we don’t exactly get the best of both worlds. For example, we know that the agents could take advantage of formal chunk definitions at runtime. As long as the semantic knowledge has a supportive role and is not the primary object of teaching, we consider this loss of expressivity and reasoning acceptable.

6 Conclusion

Now that we have shown some advantages of an ontology-based representation for semantic knowledge, more advanced ones can be outlined. If the ontology developed for a task domain is consensual among a teaching community, it is possible to link different labs such as one for a physics task domain and one for systems of equations algebra. On the other hand, multiple task domain ontologies based on an evolution of the current top-level ontology may allow more powerful and abstract problem-solving

approaches. It could lead to the application of *problem-solving-methods* [19] to ITS, a means to tackle less well-defined task domains. Semantic knowledge would then need to encompass knowledge that is currently only represented in its procedural form.

References

1. VanLehn, K. The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*. 16, 3, 227-265 (2006).
2. Mizoguchi, R. Le rôle de l'ingénierie ontologique dans le domaine des EIAH, Interview realized by Jacqueline Bourdeau, *Revue STICEF*, Volume 11, 2004.
3. Alevan, V., McLaren, B. M., Sewall, J., & Koedinger, K. The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. *Proceedings of the 8th International Conference on Intelligent Tutoring System (ITS 2006)*. pp. 61-70.
4. Ritter, S. Authoring model-tracing tutors. *Technology, Instruction, Cognition & Learning (TICL)*, 2005, 2, 3, 231-248.
5. Anderson, J. R., Corbett, A. T., Koedinger, K. R. and Pelletier, R. Cognitive tutors: Lessons learned. *Journal of Learning Science* 4(2), 167-207 (1995).
6. Fortin, M., Lebeau J.F., Abdessemed A., Courtemanche F. and Mayers A. A Standard Method of Developing User Interfaces for a Generic ITS. *Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS 2008)*. in press.
7. Kozaki K., Sunagawa E., Kitamura Y. and Mizoguchi R. Fundamental Consideration of Role Concepts for Ontology Evaluation. *Proc. of Evaluation of Ontologies for the Web (EON2006) 4th EON Workshop*, Edinburgh, United Kingdom, May 22, 2006.
8. Wenger, E. (1987). *Artificial intelligence and tutoring systems: computational and cognitive approaches to the communication of knowledge*. Los Altos, Morgan Kaufmann Publishers.
9. Mayers, A., Lefebvre, B. & Frasson, C. Miace, a human cognitive architecture. *SIGCUE Outlook*, 27(2), 61-77 (2001).
10. Fournier-Viger, P., Najjar, M., Mayers, A. & Nkambou, R. A Cognitive and Logic based Model for Building Glass-box Learning Objects. *Interdisciplinary Journal of Knowledge and Learning Objects*, 2: 77-94 (2006).
11. VanLehn, K., Lynch, C., Schulze, K. Shapiro, J. A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., & Wintersgill, M. The Andes physics tutoring system: Lessons Learned. *International Journal of Artificial Intelligence and Education*, 15 (3), 1-47 (2005).
12. Mitrovic A., Koedinger K. R., Martin B. A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling. *User Modeling 2003*.
13. Anderson, J.R. (1993). *Rules of the mind*. New Jersey, Lawrence Erlbaum Associates.
14. Reed K. S. (2007) *Cognition: theory and applications*, 7e Edition. Thomson Wadsworth.
15. Baader, F. & Nutt, W. (2003). Basic description logics. In F. Baader, D. Cavanese, D. McGuinness, D. Nardi, & P. Patel-Schneider (Eds.), *The description logic handbook: Theory, implementation and applications* (pp. 47-100). Cambridge University Press.
16. W3C OWL Web Ontology Language Overview <http://www.w3.org/TR/owl-features/>
17. Knublauch H., Fergerson R. W., Noy N. F., & Musen M. A. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. *Third International Semantic Web Conference*, Hiroshima, Japan, 2004.
18. Sirin E., Parsia B., Cuenca Grau B., Kalyanpur A. and Katz Y. Pellet: A practical OWL-DL reasoner, *Journal of Web Semantics*, 5(2), 2007.
19. Fensel, D. et al.: The Unified Problem Solving Method Development Language UPML. In *Knowledge and Information Systems Journal (KAIS)* 5(1), 2003.