

Introducing a framework-oriented approach to develop an intelligent tutoring system

Tho Toan Ly
Department of Computer Science and Software Engineering
Laval University
Quebec G1K 7P4, CANADA
ttly02@ift.ulaval.ca,

Ruddy Lelouche, associate researcher
LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, FRANCE
Ruddy.Lelouche@lirmm.fr

Abstract. This paper describes different design levels in the development of an intelligent tutoring system (ITS). Because the development of ITSs is difficult, time-consuming and costly, our ITS design proposal should make the process of ITS development less difficult and more efficient. Firstly, we present the knowledge types used in an ITS. Secondly, we describe our high level of modeling knowledge. Next, we define what we call an ITS-framework, and we show how such a framework can make the design and the development of an ITS more efficient. Indeed, the use of a framework gives us some interesting and important features: reusability, flexibility and extensibility. Lastly, we discuss our approach to the implementation of an ITS-framework and we introduce into that framework existing well-proven tools and technologies borrowed from different research results.

1 Introduction

An intelligent tutoring system (ITS) is basically a tutoring system that is designed and developed using artificial intelligent techniques. As to the tutoring domain, it is a particular type of expert system. Indeed, an ITS has some expertise in the domain that it is expected to teach: it can reason on the subject matter, it can solve problems, and it can explain or show a trace of its inferences (Lelouche, 1998; Sleeman & Brown, 1982). Thus, intelligent tutoring systems were proven to be highly effective as learning aids, and many ITS research and development efforts are taking place (El-Sheikh & Sticklen, 1998). However, few tutoring systems have made the transition to reach the commercial market so far. One main reason for this failure to deliver is that the development of ITSs is difficult, time-consuming, and costly (El-Sheikh & Sticklen, 1998): many tasks and procedures are repeatedly performed from scratch. By identifying these recurring procedures and standardizing them, we can define a common ground of development for all ITSs, and benefit from it.

To do so, we describe different levels of abstraction in the design and the development of an ITS: differentiating knowledge types; proposing a framework including guidelines and tools to ease its development; and finally describing how to build our proposed framework. Throughout our research and as we show in this paper, we try to promote reusability and extensibility in many aspects of the development.

2 Three knowledge types

In their recent research on an ITS in cost engineering, Lelouche and Morin classify the different learning domains into three groups, depending on the type of knowledge which is to be acquired by a student: know, know-how and know-how-to-be (Morin, 1998). Domains such as cost engineering, geometry and physics are categorized in the know-how type, or problem-solving type, since these domains require the student to learn how to solve domain problems. The authors state that the knowledge in an ITS in such a domain consists of three types: Domain Knowledge (DK), Problem-Solving Knowledge (PSK) and Tutoring Knowledge (TK).

Domain Knowledge is of the know type. It consists of static, descriptive knowledge to be acquired by the student, and it can be modeled in the system as concepts, relations, facts, rules, etc.

Problem-Solving Knowledge is of the know-how type. It consists of dynamic, procedural knowledge, including all the computational and inferential processes that may be used to solve a problem in the domain to be acquired by the student.

Tutoring Knowledge contains all tutoring entities enclosed in the ITS. It is not directly related to the teaching domain or to problem solving, for it is not to be taught to or acquired by the student. But it mainly helps him/her to understand, assimilate and master more efficiently the domain knowledge included in DK and PSK. It happens to include also some know-how-to-be knowledge, the quality of which will influence the quality and appropriateness of the actual tutoring strategies and actions.

3 An ITS framework proposal

For easing the development of an ITS, we propose to use a framework, that we call an Intelligent Tutoring System Framework (ITS-FW), to provide a common ground of all development aspects, including knowledge definition and reusability in both the descriptive knowledge and programming components. Thus, our proposed framework is an attempt to define a common model for ITS development and also to provide an application framework (Booch, 1991). It is a large project that will involve many research areas, such as knowledge representation, knowledge sharing, knowledge-based development, object-oriented frameworks, etc. To foster reusability, we are trying to reuse concepts found in many existing technologies from different domains: object-oriented development, distributed applications, etc. Their efficiency and strength have been proven useful in real-world systems (OMG Inc., 1997; Microsoft, 1999).

In short, our ITS-FW is composed of tools and guidelines to facilitate all facets of the development in its various stages. The following subsections present these two aspects of the framework in more detail.

3.1 The ITS framework as guidelines

Firstly, an ITS-FW is a set of guidelines. These can be used to define a common knowledge representation scheme, to define a structure to locate and access a specific piece of domain knowledge, to design and implement an ITS using the ITS-FW components, and to define a way to communicate between components.

Defining a Common Knowledge Representation (CKR). CKR is a flexible and extensible scheme to define and represent any domain knowledge. It is based on the Extensible Markup Language (XML) notation, and is a subset of the existing predefined semantic web markup language (Consortium, 2003), which is well defined, flexible and easy to use. As a result, it supports different knowledge structure types, at the meta-knowledge level as well as at the various implementation knowledge levels. Knowledge and meta-knowledge can thus be represented in different ways: procedures, concepts, production rules, propositions, frames and scripts, semantic networks, etc.¹ Another benefit of the XML notation is that, ideally, being a simple human-readable language, it allows different stakeholders such as domain experts or computer agents to read and express their knowledge, if they are willing to do so, without needing a knowledge engineer. Thus CKR provides a mean to define, reuse and extend knowledge or meta-knowledge. Finally, CKR also facilitates knowledge sharing: the knowledge in any given domain can be exchanged from one ITS to another.

Defining a structure to locate and access a specific piece of domain knowledge from various locations. Our proposed framework is an attempt to distribute domain knowledge throughout a distributed environment such as the World Wide Web. Without this structure, knowledge engineers would not be able to access a piece of existing

¹ Note that, although much knowledge is *heuristic*, this characteristic refers to the deep structure of the considered knowledge, and not to some representation technique.

knowledge in one domain to reuse it within another. Although not essential, this ability enhances the capability to exchange and access existing knowledge.

Designing and implementing an ITS using the ITS-FW components. The ITS-FW framework provides documentation to permit the integration of ITS-FW programmed components within an ITS. It also documents how to use, extend and personalize many aspects of the development, including the expression of different levels of abstraction for the system design and implementation, for both components and knowledge. This documentation should enable stakeholders such as developers and knowledge engineers to integrate and use ITS-FW features in production.

Defining a way to communicate between components: the Common Component Interaction (CCI). This interaction capability is borrowed from existing similar ones which are employed in many current technologies such as the Component Object Model (COM) (Microsoft, 1999), the Common Object Request Broker Architecture (CORBA) (OMG Inc., 1997), etc. CCI is a guideline which specifies how each module should be connected and communicate with the others. As long and as soon as all involved components conform to this predefined specification, any component can communicate with the other existing components within an application. For instance, at a high level of design, the main purpose of using CCI is to integrate relatively easily, into some ITS, components developed and sold by independent vendors. This leads to the ability to construct modules disregarding physical development sites.

Moreover, CCI provides the flexibility to integrate or substitute components into two or several ITS built for related learning domains. If two domains have some common characteristics, e.g. in user interface presentation, student monitoring techniques, or reasoning techniques (Sleeman & Brown, 1982), etc., some components used in one domain can be reused or inserted into another. For instance, if we suppose that geometry is a subset of mathematics, the domain expert module in geometry can be replaced by the one in mathematics (assuming that one exists, which is a major endeavor in itself!) without affecting the overall functionalities of the geometry ITS.

3.2 ITS framework as tools

For the practical design and implementation aspects, the ITS-FW provides (1) a set of low-level operations or domain-independent knowledge activities, called runtime knowledge, and (2) an object-oriented application framework (OO-FW).

The runtime knowledge. Any problem-solving activity or procedure, whether it is performed by the system or by a human (teacher or learner), needs, at the lowest level, to execute a simple “atomic” operations such as adding, subtracting or comparing two simple values, displaying a point, a line segment or a number, etc. Such operations are usually necessary in all domains. This is what we call the runtime knowledge, or runtime procedures. This knowledge is part of, and is used at, the execution-level of a problem-solving activity carried by an ITS. These runtime routines are implemented directly as low-level executable components. We consider the knowledge they implement as domain-independent, but we shall not focus our research to prove whether all of this knowledge is truly domain-independent.

The object-oriented framework. Jacobson, Booch and Rumbaugh define as an object-oriented framework as a micro architecture that provides an incomplete template for systems within a specific application domain (Gamma, Helm, Johnson, & Vlissides, 1995; Jacobson, Booch, & Rumbaugh, 1999). For example, it can be a system built with the explicit purpose to be extended and/or reused. The benefit of such a framework is the ability to reuse its components (Booch, 1991). In our case, such an object-oriented framework OO-FW is a part of the overall ITS-FW described in this paper. It is composed of (1) a *high level architectural view*, which is based on a common component architectural view for all existing ITSs, and (2) a set of *unfinished programmed components* (UPC). The UPC set consists of object-oriented abstracted classes and their relationships. These relationships contribute to the “core” of software execution and component integration. Also the UPCs are developed based on software design patterns to maximize flexibility and reusability. The UPC set is in charge of defining the mechanism used to plug different components to a main component; it also provides the mechanism for an ITS to be executed within a given operating system. Because all UPCs are developed in conformity with the CCI, they should be replaced or evolve without affecting the structure and functionality of the overall ITS-FW.

4 An ITS framework implementation proposal

In this paper, following the separation of the ITS knowledge into three knowledge types, we are proposing a framework that, we believe, should ease the development of ITSs in any teaching domain, because it is based on the reusability, extensibility, flexibility and sharing of knowledge structures, ontologies, tools and techniques. In this section, we focus on the low-level implementation of the framework: how the separation of knowledge types is effected in relationship to the framework, how the framework should be implemented, and which tools should be included in the framework. In the following subsections, we focus on the implementation of the proposed guidelines (section 4.1), on the interaction between knowledge components (section 4.2), on tools to model knowledge and acquire knowledge entities (section 4.3) and, lastly, on the ITS environment execution (section 4.4).

4.1 Knowledge component design and implementation guidelines

In regard to the top view of an ITS, we need to implement the system in such a way that the three knowledge types must be clearly defined and independent from one another. Having these principles in mind, we approach our simple architecture of the system: defining a knowledge component for each knowledge type: a domain knowledge component (DKC) for DK, a problem-solving knowledge component (PSKC) for PSK and a tutoring knowledge component (TKC) for TK. In our context, a knowledge component could and should be more than just one 'programmed' component, the size of which would depend on the complexity of the knowledge encoded in a domain. Indeed, a knowledge component has its own ontology, its own defined structures and its own sub-components. It is independent from other components in the same system.

By organizing each type of knowledge as its own knowledge component, such a knowledge component can be reused and integrated into different systems without knowing the structures and components of the system in which it executes. We can define an ontology and knowledge representation for each knowledge component. For example, we could use a frame representation for domain knowledge, an object-oriented representation for tutoring knowledge, e.g. to build a student model, and a task-oriented representation for problem-solving knowledge. The following subsections describe our approach to the implementation of each of these knowledge components.

4.1.1 Domain knowledge component (DKC)

As mentioned earlier (section 2), the domain knowledge consists of static, descriptive knowledge which, at the low-level of the implementation, is subsequently represented as concepts, relations, facts, rules, etc. When discussing further about our framework, we proposed a Common Knowledge Representation (CKR) (section 3.1) to provide a mean to reuse and extend knowledge or meta-knowledge. To do so, we need to conceptualize a common set of vocabulary and structures, known as ontology (Devedzic, 2001; Mizoguchi & Bourdeau, 2000), in teaching domains. We believe that by defining the DK ontology, we can enhance the reusability in many different teaching domains. Within the German Research Center for Artificial Intelligence (DFKI), Ullrich's research group has been focusing on "instructional ontology". Their goal is to provide an ontology to describe a learning resource from an instructional perspective (Ullrich, 2004).

In our research model, we use "instructional ontology" as a CKR for DK to define any teaching domain knowledge structures. We believe that the modeling of the teaching domain knowledge for an intelligent educational system (IES) and for an ITS is very similar: both systems are educational software and contain artificial intelligent components (AAAI, 2000-2005). Extending this small and common ontology, we can easily define, in a teaching domain, a specialized ontology; then we may use it as a basis to acquire knowledge entities: for example, in physics, Newton's law is a theorem and falling gravity is 9.8m/s is a fact.

4.1.2 Problem-solving knowledge component (PSKC)

The PSKC is a programmed component that encodes the problem-solving knowledge. It is responsible for solving a domain problem through a sequence of computational and inferential procedures. Throughout the years, many techniques and programming languages have been used to model this type of knowledge: command-based presentation in imperative programming languages (Fortran, Algol, C, etc.), rule-based presentation in Prolog and LISP, object-based presentation in C++, Java, etc. (Morin, 1998). How it should be implemented is based on the

preference of the developers who build the component. In our framework, we simply provide a mean for a component of this type to interact with other components in the same system. In our endeavor, we focused on the Unified Problem-Solving Method Language (UPML) (Fensel, Motta et al., 1999), a complete framework providing means to reuse and integrate *problem-solving methods* (Schreiber, Wielinga, Akkermans, Velde, & Hoog, 1994) in the development of a knowledge-based system. It is a software architecture for knowledge-based systems providing *components*, *adapters* and a *configuration* of how the components should be connected using the adapters. In addition, it is a set of design guidelines specifying how to develop a system constructed from components and connectors (Fensel, Motta et al., 1999). Thus, UPML thoroughly defines how components interact and adapt to one another in a knowledge-based system. We believe that it is a solid base that we can use and customize to our ITS-framework.

The UPML framework defines not only how to reuse parts or all of the knowledge-component, but also how to interact with such components. Moreover, it provides a well-defined ontology for problem-solving methods (Fensel, V. R. Benjamins et al., 1999). In the previous sections, we defined our framework characteristics as similar to those of the UPML framework. For instance, our framework is aimed at fostering knowledge reuse and a component-oriented development, and also proposes guidelines and tools to ease ITS development. So does the UPML framework, but for more general knowledge-based systems. In consequence, we believe that, by reusing the UPML framework and specializing it, we may devote less effort to develop our framework, while providing eventually an efficient and well-tested framework.

For our PSKC, similarly we reuse the UPML problem-solving method ontology to define problem solving in our component. By conforming to this ontology, we believe that the PS knowledge can be reused and extended for any ITS without worrying about how it is encoded or implemented.

4.1.3 Tutoring knowledge component (TKC)

As for the tutoring knowledge, due to the limited time devoted to our work (a master's thesis), we decide not to concentrate on this component, but rather to focus on the teaching domain knowledge component and on the problem-solving knowledge component. Depending on the underlying teaching and learning theories, this component could be very complex and very ad-hoc. However, how this component is built is not our concern here, and would not cause any problem in a system that is constructed with our framework, because knowledge components are independent from one another.

4.2 Common Component interaction (CCI)

So far, we have three independent knowledge components. Each one has no knowledge about the existence of other components in the system. It has its own ontology and some protocol communication: namely, a specification of how external components interact with it in a system. Because we do not impose any common protocol of communication that all components should adhere to in order to interact with one another, we must define a way to 'map' the ontology of a given component with those of the external components interacting with it. Thus a set of mapping relations express the connections between the ontologies of two components (M. Crubézy, Pincus, & Musen, 2003). For that purpose, the UPML framework defines knowledge *adapters* (Monica Crubézy & Musen, 2003), expressing how to map a domain knowledge component to interact with problem-solving methods in a knowledge-based system. That approach is a great source of inspiration for us to define how to map a DKC to a PSKC in our framework. In fact, we believe that we can reuse the UPML framework to design the mapping for DKC and PSKC, and that we could also propose a similar mapping for TKC to interact with DKC and PSKC.

To help the reader understand this concept better, we demonstrate it through a simple example. Suppose that our projected ITS consists of a DKC specialized in physics and a PSKC specialized in arithmetic calculation. A simple problem statement is given: "2km + 2km = ? km". PSKC only knows that addition is the sum of one or many values which must be numeric. As consequence, we must map "2" from the problem statement into a numeric value 2 for PSKC, so that PSKC can do its calculation. When the PSK component terminates its calculation, it will return the numeric value 4, and the mapping will then tell the DSK-component that the (physics) result is "4 km".

The mapping process becomes important if we want two components to communicate with each other. Certainly, defining a mapping is an extra step that needs to be performed in the development process; however, its cost should become reasonable if we have access to many ready-to-use components: the cost of developing a mapping is lower than that of adapting an existent component to make it work with other components. Moreover, our ITS framework does not impose any kind of structure on any of its components.

4.3 Tools to ease the knowledge implementation

To make a new proposal framework for ITSs, we effectively promote reusability and extensibility by applying the UPML existing framework and concepts, and by adding more functionality to it. Thus, our framework becomes an extension of the UPML framework. We reuse the concept knowledge component and adapter of UPML. One advantage of reusing the UPML framework is that UPML is perfectly integrated with *Protégé* (Gennari et al., 2003). *Protégé* is an ontology editor and knowledge acquisition system developed by a group of researchers at Stanford University. *Protégé* is a tool, which allows users to construct domain ontologies, to customize data entry forms, and to enter data. It is also a platform, which can easily be extended to include graphical components such as graphs and tables, various media such as sound, images, and video, and various storage formats such as OWL, RDF, XML, and HTML (N. F. Noy, 2001).

Thanks to the extensibility built in *Protégé*, UPML communities have created a plug-in which provides an editor to edit a problem-solving method and an editor to edit its mapping. Thus, we can use those very editors to add and edit our problem-solving methods for ITS, and to define their mappings. Also, *Protégé* is a perfect tool for modeling our teaching domain ontology: it has been used to model instructional ontology and UMPL ontology. Thus, with *Protégé*, we expect to be able to define in a seamless fashion our teaching domain knowledge, be it mathematics, physics or geometry.

4.4 ITS environment execution

As we mentioned earlier in section 3.2, the ITS-FW provides a runtime knowledge which is a set of low-level operations. At the implementation level, the runtime knowledge is a process which translates a high level programming language into a machine language. Thus it takes place in an execution environment where an “atomic” operation, as mentioned above, can be translated into the machine code. Moreover, it is in charge of connecting different knowledge components in an ITS (see section 4.2), thanks to their mapping relationship(s). However, defining a mapping is not enough: we need to add to the execution environment an *interpreter* for translating a given knowledge instance from one knowledge component to the others.

The execution environment provides the executed steps of a problem-solving method. The result of each step can be mapped back to some domain knowledge instance. This “back-mapping” provides a mean, especially for a tutoring component, to examine the reasoning process for the purpose of suggesting a student certain execution steps during a tutorial session if necessary.

In order to maximize reusability, flexibility and extensibility, we carefully design the execution environment using the object-oriented framework concepts. This execution environment should be flexible to replace any programmed component in the environment without corrupting it; it should be reusable to customize any component for the needs of a specific domain. To make it extensible, as mentioned earlier, we are developing a UPC (see section 3.2) and the associated plug-in mechanism so that any programmed component can be interacted with the environment at runtime.

4.5 In summary

Using our ITS framework to build an ITS, one will need to define the domain-dependent knowledge, the problem-solving knowledge and the mapping knowledge. Because these knowledge are independent from one another, we can always reuse them in the development of an ITS specializing in different domains. As we look back at the example in section 4.2, we can reuse the PSKC specialized in arithmetic calculation in the development of an ITS in

geometry by simply defining the mappings necessary for the DKCs and PSKCs to communicate with one another. Throughout the paper, we have tried to show that our proposal framework relates to many fields such as knowledge representation, knowledge engineering, object-oriented development, etc. Because it is too large for the scope of our research (a master's project), we hope to establish mainly:

- a common ontology for DK and PSK,
- a guideline on how to design DKC and PSKC based on our common ontology and how to define the associated mappings,
- an ITS environment execution, as mentioned in section 4.4, that is highly extensible and customizable to meet most, if not all, of the needs to build any type of ITS.

We hope that our proposal framework could be a subject of interest to many researchers, so that it can be carried on and be completed in some near future.

5 Conclusion

As we described our approach to build an ITS, we progressively mapped our design activities into different levels of abstraction. At the highest level of abstraction, which is the ITS knowledge modeling level, we separated ITS knowledge into three distinctive knowledge types. Next, at the design level, we proposed an ITS-framework, the purpose of which is to ease the process of ITS development. That ITS-framework comprises: (1) a set of guidelines, including common knowledge presentation, knowledge sharing and common component interaction, to help us reuse existing knowledge structures and entities; (2) a set of tools, that take care of common operations in all domains, to help us focus on developing and extending knowledge components for a specific need. Lastly, at the implementation level, we proposed to use rather more concrete existing well-proven tools and guidelines to develop the framework. Thanks to the reuse of these existing tools and guidelines, the development of our ITS-framework is effective and cost-saving. Because of their inclusion in our ITS-framework, the latter should be reliable, well-tested and ready to be used in the development of a real-world ITS.

6 References

- AAAI. (2000-2005). Intelligent Tutoring Systems. From <http://www.aaai.org/AITopics/html/tutor.html>
- Booch, G. (1991). *Object-Oriented Design with Applications*. Redwood City (California, USA) and Don Mills (Ontario, Canada): Benjamin/Cummings Publ. Co.
- Consortium, W. W. W. (2003). Web Semantic. From <http://www.w3.org/2001/sw/>
- Crubézy, M., & Musen, M. A. (2003). Ontologies in Support of Problem Solving. In S. Staab & R. Studer (Eds.), *Handbook on Ontologies* (pp. 321-341): Springer.
- Crubézy, M., Pincus, Z., & Musen, M. A. (2003). *Mediating Knowledge between Application Components*. Paper presented at the Proceedings of the Semantic Integration Workshop of the Second International Semantic Web Conference (ISWC-03), Sanibel Island, Florida.
- Devedzic, V. (2001). *The Semantic Web - Implications for Teaching and Learning*. Paper presented at the ICCE 2001 Conference, Seoul, Korea.
- El-Sheikh, E., & Sticklen, J. (1998). *Framework for developing intelligent tutoring systems incorporating reusability*. Paper presented at the IEA/AIE-98:11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Castellón (Spain).
- Fensel, D., Motta, E., Benjamins, V. R., Decker, S., Gaspari, M., Groenboom, R., et al. (1999). The Unified Problem-solving Method development Language UPML. *ESPRIT project number 27169, IBROW3, Deliverable 1.1, Chapter 1*. From <http://www.aifb.uni-karlsruhe.de/WBS/dfe/publications99.html>.
- Fensel, D., V. R. Benjamins, Decker, S., Gaspari, M., Groenboom, R., W. Grosso, M., et al. (1999). *The Component Model of UPML in a Nutshell*. Paper presented at the WWW Proceedings of the 1st Working IFIP Conference on Software Architectures (WICSA1), San Antonio, Texas, USA.

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Software Architecture*. Reading (Massachusetts, USA): Addison-Wesley.
- Gennari, J. H., Musen, M. A., Ferguson, R. W., Grosso, W. E., M. Crubezy, Eriksson, H., et al. (2003). The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. *International Journal of Human-Computer Studies*, 58(1).
- Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process*. Reading (Massachusetts, USA): Addison-Wesley.
- Lelouche, R. (1998). The successive contributions of computers to education: a survey. *European Journal of Engineering Education*, 23, 297-308.
- Microsoft. (1999). Component Object Model. From <http://www.microsoft.com/com/resources/comdocs.asp>
- Mizoguchi, R., & Bourdeau, J. (2000). Using Ontological Engineering to Overcome Common AI-ED Problems. *International Journal of AIED*, 11(2), 107-121.
- Morin, J.-F. (1998). *Conception of an Intelligent Tutoring System in Cost Engineering: Knowledge Representation, Pedagogical Interactions, and System Operation*. Université Laval, Québec, Canada.
- N. F. Noy, M. S., S. Decker, M. Crubezy, R. W. Ferguson, M. A. Musen. (2001). Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems*, 16(2), 60-71.
- OMG Inc., (1997). OMG CORBA/IIOP Specification, 1997-2003.
From http://www.omg.org/technology/documents/corba_spec_catalog.htm
- Schreiber, A. T., Wielinga, B., Akkermans, J. M., Velde, W. V. D., & Hoog, R. d. (1994). CommonKADS. A Comprehensive Methodology for KBS Development. *IEEE Expert*, 9(6), 28-37.
- Sleeman, D., & Brown, J. S. (1982). Introduction: Intelligent tutoring systems. In D. Sleeman & J. S. Brown (Eds.), *Intelligent Tutoring Systems* (pp. 1-13). London, UK and Orlando, Florida, USA: Academic Press.
- Ullrich, C. (2004). *Description of an Instructional Ontology and its application in Web Services for Education*. Paper presented at the Poster Proceedings of the 3rd International Semantic Web Conference ISWC2004.