ITS Domain Modelling with Ontology

Brent Martin
(University of Canterbury, Christchurch, New Zealand
brent.martin@canterbury.ac.nz)

Antonija Mitrovic
(University of Canterbury, Christchurch, New Zealand
tanja.mitrovic@canterbury.ac.nz)

Pramuditha Suraweera
(Carnegie Learning Inc, Pittsburgh, USA
psuraweera@carnegielearning.com)

Abstract: Authoring ITS domain models is a difficult task requiring many skills. We explored whether modeling ontology reduces the problem by giving the students of an e-learning summer school the task of developing the model for a simple domain in under sixty minutes using ontology. Some students also used our tool to develop a complete tutor in around eight hours, which is much faster than they could be expected to author the system without the tool. The results suggest this style of authoring can lead to very rapid ITS development. We further extend the ontological approach with *domain schema*: high-level abstractions that describe the semantics of the domain model for a class of domains. Using domain schema reduces the authoring effort to one of describing only those aspects that are unique to this particular domain, and enables the ontology-based approach to model domains with different semantic requirements.

Keywords: intelligent tutoring systems, authoring systems, constraint-based modeling, domain models, Ontology
Categories: L.1.0, L.1.3, L.2.0, L.3.0

1 Introduction

Intelligent Tutoring Systems (ITS) increasingly show promise as a technology that will expand the horizons of education from those able to attend a bricks-and-mortar institution to anyone with an Internet connection. Acting as an enhancement to traditional distance learning offerings, they promise to augment laboratories and tutorials by allowing students to practice the skills they are learning from home. In recent years tutors such as the Geometry and Algebra tutors, and the Addison-Wesley database place suite (SQL-Tutor, ER-Tutor and NORMIT) have made it out of the lab and into the classroom [Koedinger, 97], [Mitrovic, 06a].

Despite this success, intelligent tutors have still not been adopted widely. One reason for this is the difficulty in building them. Recent research efforts have tried to address this shortcoming. The Cognitive Tutor Authoring Tools (CTAT) [Koedinger, 04] attempt to reduce the authoring effort for ITSs based on model tracing. The tools support the creation of two types of tutor: 'Pseudo tutors' and 'Cognitive tutors'. Authors can quickly build Pseudo tutors by developing the user interface for the tutor

and then demonstrating the solution to one or more problems. However, these are not "real" ITS: the resulting model is essentially a trace of the correct behavior for *that problem only*. Whilst it is possible to add multiple solution pathways (including incorrect behavior) and comprehensive feedback, the resulting tutor is nevertheless suitable only for the problem from which it was authored. To convert a pseudo-tutor into a full cognitive tutor the author must manually create the production rules that represent a general model of the domain, which is a formidable task. A refinement to this approach is to incorporate *bootstrapping*, in which novice data from several students is used to create a behavior graph that summarizes their collective behavior. Because it records not only (correct and incorrect) actions taken but also the frequency of each action, it provides rich information about likely student actions. For example, the Cool Modes system has been extended to record the collective behavior graph of collaborating students for this purpose [Harrer, 06]. REDEEM [Ainsworth, 04] takes a different approach. It allows educators to add pedagogy to e-learning delivery by tailoring the delivery of educational material to stereotypical student groups. REDEEM has been shown to increase student learning, but it is a very different authoring task that does not generate diagnostic models.

Constraint-Based Modeling (CBM) [Ohlsson, 94] is an effective approach for building Intelligent Tutoring Systems that supports the building of domain and student models. Constraint-based tutors are effective: students using SQL-Tutor have shown significant gains in learning after as little as two hours of exposure to this system [Mitrovic, 99]. Also, CBM seeks to minimize the authoring effort by requiring the author to model only states, rather than solution paths [Mitrovic, 03]. For domains such as design tasks where the number of possible solution states is huge this can greatly reduce the authoring effort. Nevertheless, the task of building an ITS is still far from trivial, and requires many different areas of expertise including cognitive science, software engineering and educational instruction. To reduce the authoring effort we developed WETAS (Web-Enabled Tutor Authoring System), a web-based tutoring shell that performs all of the common functions of text-based tutors, and thus obviates the software engineering requirement. To demonstrate the flexibility of WETAS we re-implemented SQL-Tutor and developed a new ITS for teaching spelling and vocabulary (LBITS). Although these domains share the property of being text-based, they have very different problem/solution structures. We evaluated LBITS in a New Zealand school and found it to be effective [Martin, 02a; Martin, 02c].

WETAS removes much of the effort required to build an ITS, but it does not directly facilitate the building of the domain model, which is arguably one of the most difficult tasks [Murray, 97]. In particular, the author must write the domain rules or "constraints", which requires programming skill and an understanding of artificial intelligence techniques. For complex domains the constraint set can quickly become large (SQL-Tutor has over 600 constraints), making it hard to manage. One way to overcome this is by modeling the domain at a higher level using ontology. We developed a tool, WETAS-Ontology, which allows authors to graphically model the domain as ontology. A constraint generator then creates the required constraints from the concepts in the ontology. The resulting constraints form a domain model that can be used to provide highly specific feedback tailored to the individual student's misconceptions, and to drive the pedagogical process, for example by selecting problems based on the concepts for which the student is currently violating

constraints. The ontology assists in this latter task by allowing the problem selector to infer which similar concepts a student is likely to find difficult when the problems applicable to the current concept have been exhausted.

WETAS-Ontology was used as a learning aid at the 2006 e-learning school at the National College of Ireland, which enabled us to test our hypothesis that modeling using ontology is easier and faster than writing constraints by hand. This paper reports on our experiences. The next section briefly introduces CBM and the WETAS authoring shell, and WETAS-Ontology is described in Section 3. Sections 4 and 5 describe how WETAS-Ontology was used at the e-learning summer school. Section 6 discusses limitations of this approach and introduces one potential solution, Domain Schema, which are described in detail in Section 7. Finally, we conclude in Section 8 and discuss future directions.

2 Constraint-Based Modeling and WETAS

CBM [Ohlsson, 94] is based on the theory of learning from performance errors [Ohlsson, 96]. It models the domain as a set of state constraints, where each constraint represents a declarative concept that must be learned and internalized before the student can achieve mastery of the domain. Each constraint represents a restriction on allowed solution states, and takes the form:

If *<relevance condition> is true for the student's solution,*
THEN *<satisfaction condition> must also be true*

The relevance condition of each constraint checks whether the student's solution is in a pedagogically significant state. If so, the satisfaction condition is checked. If it succeeds, no action is taken; if it fails, the student has made a mistake, and appropriate feedback is given.

In a constraint-based tutor the constraints are used to model the fundamental concepts of the subject (domain) being taught such that when a constraint is violated, *regardless of the broader context*, the student has made a mistake and the corresponding feedback is given. *Syntactic* constraints check that the solution is syntactically correct. For example, in the domain of SQL queries, any attributes listed in the "SELECT" clause must be separated by commas. Conversely, *semantic* constraints check whether the student's solution has solved the problem, usually by comparing it to an "ideal" solution supplied by the teacher. Again from SQL, one such semantic constraint tests that all of the tables required to retrieve the desired data are present in the student's answer. In the domain of English spelling syntactic constraints test that the word a student has given is correctly spelled (regardless of whether or not it answers the question), whilst semantic constraints test whether the answer requires certain letter combinations that children struggle with (e.g. "ough"). The semantic constraints test for all of the different possible encodings of the concept they are attempting to test; the student is thus permitted to use a different problem-solving strategy to the author, or even to mix strategies, provided no fundamental domain concepts are violated. For example, in SQL-Tutor the student's query may access the required tables using either a nested select or one of several kinds of table

join; the system will permit any of these provided they have correctly captured the semantics required (i.e. they correctly answer the question).

WETAS is a web-based tutoring engine that provides all of the domain-independent functions for text-based ITS. It is implemented as a web server, written in Allegro Common Lisp, and using the AllegroServe Web server [FranzInc]. WETAS supports students learning multiple subjects at the same time; there is no limit to the number of domains it may accommodate. Students interact through a standard web browser such as Firefox or Internet Explorer. WETAS implements as much of the ITS functionality as possible by providing generic processing capability that is expected to be applicable to a wide range of domains. In particular, it provides the following functions: problem selection, answer evaluation, student modeling, feedback, and the user interface. The author need only provide the domain-dependent components, namely the structure of the domain (e.g. any curriculum subsets), the domain model (in the form of constraints), the problem/solution set, the scaffolding information (if any), and possibly an input parser, if any specific pre-processing of the input is required. WETAS provides both the infrastructure (e.g. student interface) and the "intelligent" parts of the ITS, namely the pedagogical module and the domain model interpreter. The former makes decisions based on the student model regarding what problem to present to the student next and what feedback they should be given. The latter evaluates the student's answers by comparing them to the domain model, and uses this information to update the student model. Constraints are written in a custom pattern-matching language that is intended to be simple to author. The system reasons about the constraints in three ways: it may evaluate the student solution against constraints to decide what is wrong and give feedback, it may use the constraints to correct errors in the student's input (and thus show them how to proceed), and it may use constraints to generate new problems to present to the student. For more information see [Martin, 00; Martin, 02b].

WETAS has been used to build several tutors, including EER-Tutor for Addison-Wesley [Mitrovic, 06] and Collect-UML [Baghaei, 06]. It has also been used for the past four years by a graduate University class in Intelligent Tutoring Systems at Canterbury University. In this class students are assigned the task of building an ITS in WETAS. The first time it was used by this class it became apparent that further authoring tools are required: the students were able to build a tutor in the time allocated (three weeks) but their domain models were generally sub-optimal [Martin, 03]. We found that students make mistakes at all levels of the domain authoring process: they fail to model pedagogically significant states (i.e. model spurious concepts), do not always capture the intended states in their constraints, and make errors during constraint encoding or encode them inefficiently. Thus when working at the constraint level authors find it difficult both to conceptualize the domain and then to implement their model. Our proposed solution was to provide a high-level tool that automates the encoding of constraints based on an ontology that the author provides. We hypothesize that this will help in two ways: by removing the low-level steps from the authoring process (and thus preventing encoding errors being made) and by allowing the author to visualize the structure of the domain during authoring so that they are more likely to capture the intended pedagogical states.

3 WETAS-Ontology

The use of ontology in education systems is not new. Mizoguchi and Bourdeau advocate authoring intelligent instructional systems by engaging authors in knowledge *modeling* rather than knowledge *engineering* [Mizoguchi, 00]. They propose building education systems by creating *task* ontology (which models pedagogy) and *domain* ontology, which represents each individual domain. We are interested in the latter: how do we use ontology to develop domain/student models?
Modeling domains at the ontology level has other potential benefits too, such as the ability to re-use parts of a domain model and to obtain interoperability across different ITS platforms. For example, Barros et al propose using ontology to model computer supported collaborative learning (CSCL); the ontology provides a standardized vocabulary for collecting and analyzing students' collaborative behavior, and thus this part of the model could potentially be shared across multiple CSCL systems for different domains [Barros, 01]. As well as enabling reuse of such models, it potentially allows student data to be analyzed across multiple domains. For example, in [Soller, 03] students' collaborative discussions were analyzed and used to develop a model of effective patterns of interaction; by (re)using a shared ontology describing the collaborative model as described by Barros et al, this data could be used to build a global picture of a student's collaboration skills across otherwise unrelated education systems. Such an approach would provide the opportunity to coach them on this particular skill regardless of which system they are currently using. Finally, Ontologies also have the desirable property of representing domain information at increasing levels of abstraction. This is particularly useful if we wish to expose the domain/student model to the student; StyLE-OLM [Dimitrova, 03] uses this approach to provide an interactive open learner model.

WETAS-Ontology is an experimental tool for authoring ITS domains. It consists of two parts: a graphical editor for creating the ontology and a constraint generator. The latter parses the ontology and creates constraints for testing the student solution based on the concepts in the ontology. One goal of this research is to develop authoring tools that are easy for ITS "lay people" to use, i.e. teachers in general. Many tools already exist that facilitate the development of ontology (e.g. Protégé [Puerta, 92], Protégé-Owl [Knublauch, 04]), however these tools are typically aimed at experienced knowledge engineers, and we considered they would be too difficult for non-experts to use. In particular, our tool attempts to visualize the entire model in a clear, graphical manner.

Figure 1 is a screen shot of the ontology editor showing the developed ontology for the domain of search engine queries that was used for the case study. In this domain students are given the criteria for a search engine query, which they write using a hypothetical language that consists of logical expressions containing the words and strings they are looking for. The ontology is a combination of taxonomy ("kind-of" relationships) and partonomy ("part-of" relationships). The graphical representation adopted was chosen to visualize both of these views simultaneously. Diamonds represent alternative constructs/concepts (kind-of relationships). For example, a *search expression* consists of a *negative expression* or a *positive expression*. Conversely, child nodes of rectangles represent a strict sequence of required sub-parts.

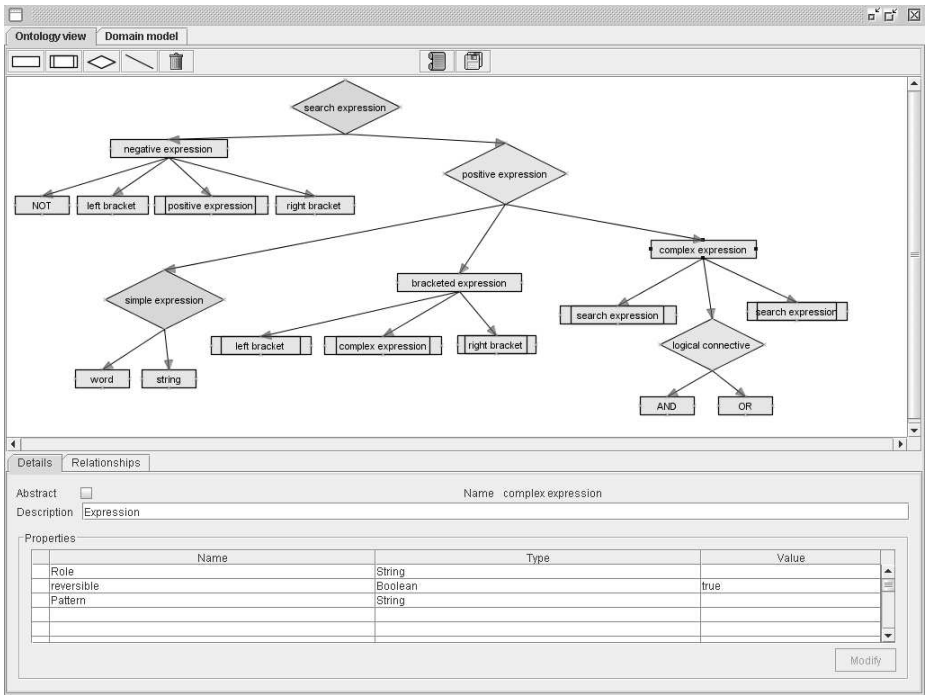

Figure 1: WETAS-Ontology interface

In this ontology, a *negative expression* consists of NOT, followed by a *left bracket*, followed by a *positive expression* followed by a *right bracket*. Rectangles with double-lined sides represent concepts that have already been defined elsewhere in the ontology; domains may thus be recursive, as in the example given. A concept may optionally have three properties: *role*, which identifies their purpose in the parent concept (for example, the role of search expression in a complex expression could be "first argument"); *reversible*, which indicates whether or not this concept has the same meaning when parsed backwards; *pattern*, which describes how this concept is identified in the solution. Pattern may be a string, or the name of a macro if complex processing is required to determine membership of the concept. The purpose of the ontology is to capture the fundamental concepts of the domain so that these can be tested in the student solution. There is no standard process for creating ontology, however we have found that for many domains (including the one in Figure 1) a useful approach is to begin with the grammar of valid solutions and add further concepts as required.

The constraint generator uses the ontology to create a set of constraints that can be loaded into WETAS and used to evaluate student solutions. For the purpose of this study we generated only semantic constraints. Templates are used to create a set of constraints from each concept in the ontology. These constraints test for the presence/absence of any examples of each concept (i.e. is this concept used at all), that all of the required instances of each concept are present in the solution, and that the subcomponents of each instance are correct (e.g. does the student's logical

connective have the correct arguments). Feedback messages are also generated automatically based on templates. This fairly simple set of templates yields a plausible domain model. Note however that it is not intended to deliver the final set of constraints; typically authors will modify the feedback messages, add additional constraints for complex concepts and edit the generated constraints, perhaps to make them more general. Figure 2 shows two examples of generated constraints. The first checks whether or not a string is needed. The test for a string is complex, so a macro has been used. Writing the macro is an additional task to producing the ontology; in practice few (if any) macros tend to be required. The second constraint checks whether or not a complex expression is needed. In this case there is no easy way to test for this concept because it consists only of two alternative sub-concepts and no literal components. The generator therefore descends the tree until it finds sub-concepts with literal components (in this case "AND" and "OR") and creates alternative tests for each alternative sub-component.

WETAS-Ontology was evaluated at the e-learning summer school in June 2006 at the National College of Ireland, Dublin. This forum was considered an ideal testing ground because the participating students were of mixed backgrounds, with less than half being Computer Scientists. The first author gave instruction at this school, which consisted of two hours of lectures about ITS (and domain/student modeling in particular), followed by a 90 minute practical session. Instructors were also asked to contribute a potential project idea, from which the students would choose one for a one-day practical project. WETAS-Ontology was used for both of these purposes.

4 Case Study 1: A Domain Model in Sixty Minutes

To determine the feasibility of using WETAS-Ontology we asked the students to use it during their 90-minute practical session to develop the ontology for the domain of search engine queries. The students were first lectured about the ontology tool, WETAS and the domain; this took approximately 30 minutes. They then had a further 60 minutes to develop their model. The WETAS tutor authoring shell was installed on each of their computers along with WETAS-Ontology. The other necessary components of the search engine query tutor (e.g. the problem/solution set) were also set up for them. The students could therefore test their ontology at any time. First,

```
(5 "Check whether you need one or more string(s) in your answer."
  (MATCH IS ANSWER (?* (^string ?IS_1) ?*)) ;; relevance

  (MATCH SS ANSWER (?* (^string ?SS_1) ?*)) ;; satisfaction
"ANSWER")

(16 "Are you sure you need complex expression(s) in your answer?"
 (OR (MATCH SS ANSWER (?* "AND" ?*)) ;; relevance
     (MATCH SS ANSWER (?* "OR" ?*)))

 (OR (MATCH IS ANSWER (?* "AND" ?*)) ;; satisfaction
     (MATCH IS ANSWER (?* "OR" ?*)))
"ANSWER")
```

Figure 2: Generated constraint examples

they would use the ontology editor to begin creating the ontology. They then instructed the tool to generate the constraint set. Finally, they loaded the constraints into WETAS and tried out the tutor. They were able to repeat this procedure as often as desired until they had completed the model or ran out of time. When they first used WETAS-Ontology it contained just the definition of "simple expression" from Figure 1. This is a working model, but it only recognizes search expressions that consist of a single word or string. Their task was to extend the model to cover the entire search engine language. Twelve students attempted to complete the task.

We categorized each model by comparing it to that in Figure 1. The categories were: *complete* – the model leads to as good a constraint set as ours; *useable* – the model generated a significant subset of the constraints, such that the resulting tutor gave useful feedback, but would not recognize all legal expressions; *good attempt* – the model had a significant number of the relevant concepts but contained substantial errors or omissions; *poor* – some attempt had been made but there had been little progress (these latter models might also be unusable because they were not syntactically correct).

Half of the students produced useable domain models, of which one was almost identical to that produced by this paper's authors, and was both of high quality and complete. Figure 3 shows a useable (but not complete) model. The main problem is that the student has not made the model recursive; the resulting constraints are therefore unable to deal with complex solutions. For example, because the arguments to "conjunct" are simple expressions only, the model generated from this ontology will be able to recognize "fish and chips" but not "fish and chips and salt," which contains a nested conjunct. There were also other differences, such as whether or not the author had grouped conjuncts and disjuncts into a high level concept (e.g. "complex expression" in Figure 1). Some of the "useable" solutions also missed whole parts of the ontology (such as bracketed expressions) or duplicated parts of the ontology instead of abstracting out common concepts. Of the other six participants, three were classified as "good attempt". These students had produced reasonable ontologies but they were still some way from being complete, and would hence generate constraint sets that failed to test significant features of the solution. The remaining three were "poor"; these participants appeared to have struggled with the

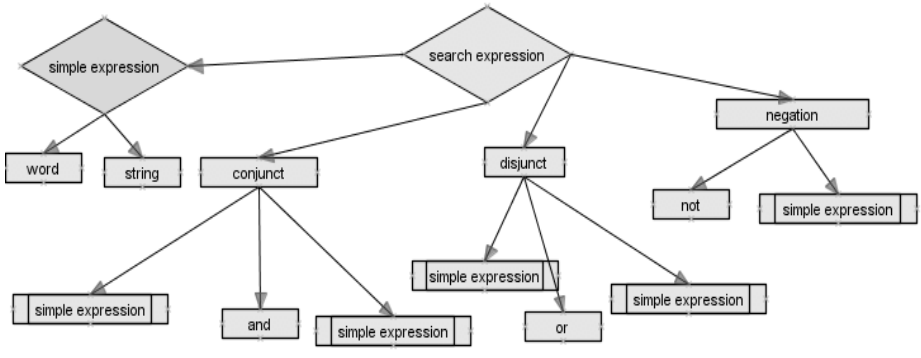

Figure 3: Example of a "useable" ontology

whole task of creating ontology.

When asked informally for their comments the students were generally very positive about the experience. In particular, they were impressed that they had produced an ITS that generated useful feedback in such a short space of time (less than 60 minutes). They also commented that they found the tool easy to use and that the ontology representation, once explained, was easy to understand. Some participants also commented that they were impressed with the level of generality of the tool, and that they could see how it could be used to develop ITS across a wide range of domains.

However, it appears that this approach to authoring does not suit everyone. In particular, the concept of recursion appeared not to be obvious to most participants. For those participants who scored "poor" it is likely that they have not had to perform similar abstraction tasks before; at least one such student commented that the modeling task was so foreign to them that they had simply not known where to begin, despite the tuition they had received. This may be a feature of the students' background. For example, the student who developed a complete ontology was a Computer Scientist studying in a similar area (collaborative e-learning), and who would therefore be familiar with the concepts behind the task. For authors of other background some tuition in developing ontology is likely to be needed.

Despite these limitations the results were sufficiently positive that we proposed WETAS-Ontology as a potential subject for a group project.

5 Case Study 2: A Tutor in a Day

The participants at the summer school were all required to contribute to a group project, which would be assessed. The students were given a list of seven potential projects spanning a variety of subjects in the general area of e-learning. Eight of the students (more than half the class) chose to use WETAS-Ontology to build an ITS. They separated into two groups, both of which worked on tutors in the domain of English spelling, a similar domain to an example they had been given. The goal was to build a complete tutor from scratch. They were allocated around six hours of class time to complete the project, although they could work outside class hours if they wished. The first group critiqued WETAS-Ontology and determined (incorrectly) that the ontology representation was too weak to support their chosen domain. As a result they suggested an alternative approach whose representation was conceptually similar to Object Oriented software design, for which they built a simple prototype. The second group developed a complete tutor using WETAS-Ontology. We turn our attention to this second group.

To develop a complete tutor the students were required to author a set of problems and ideal solutions (the latter being used by the constraints to check semantics), as well as building a complete domain model. To author the domain model, in addition to creating the ontology they would need to edit the generated constraints to provide better feedback and add any additional semantics that were too difficult (if not impossible) to model in the ontology. The group chose the domain of pluralization of nouns.

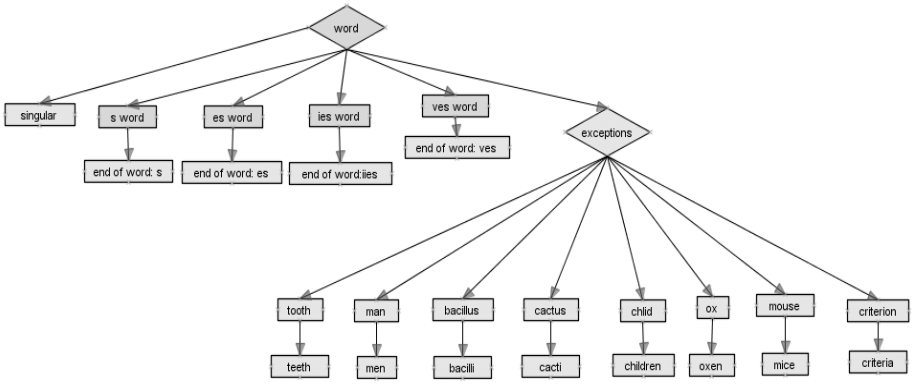

Figure 4: The pluralization ontology

Figure 4 shows their completed ontology. This ontology is generally similar to what we would have produced, the main difference being that the final leaf nodes are not actually required (i.e. the nodes above could serve as the leaf nodes). There were also some other minor errors (e.g. bacillus and cactus are two examples of the same rule). The leaf nodes on the left represent regular nouns that can be grouped into "rules" of pluralization, while those on the right are irregular nouns that can only be learned individually. The semantics for the regular noun groups can either be modeled in the ontology (e.g. by enumerating all of the words belonging to each group) or via macros. The group chose to use this latter solution, and two of the group members paired up to perform this job. Further, there are two ways that macros can be used to represent the required concepts: by testing the letters on the end of the word for the required regular form, or by enumerating the words that belong to each group. The former is more robust and efficient but requires a greater knowledge of WETAS' pattern-matching language, while the latter is brittle; the macro will require modification every time new vocabulary is introduced into the problem set. The students chose to enumerate the members of each group, which is understandable given the limited time available and given that they were not taught how to write complex macros. It does however illustrate that greater familiarity with the tool would be essential for developing high quality domain models.

The group produced a fully working tutor in around eight hours. Whilst the domain chosen was not particularly complex, this is nevertheless an impressive achievement. The final tutor consisted of 108 constraints. If all of the group's time was spent purely on this task, this result equates to less than five minutes per constraint. Given that there were four group members, this equates to around twenty person-minutes. This is significantly less than the 10 hours per rule reported for model tracing tutors [Anderson, 95] or even the 1.1 hours that it may typically take to hand-write a constraint for CBM [Mitrovic, 03]. In practice the participants also had to perform other tasks such as authoring the problem/solution set, so these results are conservative. For simple domains such as this, authoring by ontology delivers a major improvement in efficiency. The quality of the domain model they produced is comparable to what this paper's authors would have created.

6 Discussion

WETAS-Ontology reduces the effort required to build an ITS by abstracting the authoring process to one of graphically modeling domain ontology rather than encoding individual constraints, but nonetheless it is still a formidable task. In particular, it appears that developing domain ontology is a process that does not come naturally to all authors. Whilst some of the students in the study produced usable domain models, only one was completely correct. The rest appear to have had difficulty grasping the complexity of the modeling task, and nearly all participants were unable to model the recursive nature of the domain. Naturally we would expect this situation to improve with experience, however the fact that some participants found the whole idea of modeling ontology too difficult to grasp is of some concern. It is thus desirable to somehow lower the bar further for those new to conceptual modeling.

One of WETAS-Ontology's strengths is that it automatically generates a plausible set of constraints from the ontology. However, to do this it must make many assumptions regarding the semantics of the domain model being created, and hence the domain being built must be modeled in such a way that the tool's reasoning will be correct. For example, ASPIRE is a mature authoring system (based on WETAS) that also generates constraints for CBM domains using ontology [Mitrovic, 06b]. The generation process for this tool was based on the semantics of the domain model for entity-relationship modeling; in this domain each concept in the model represents an object the student may create in the solution (e.g. an entity or attribute), with relationships in the ontology representing relationships between objects in the final diagram (e.g. a one-to-many "has" relationship between "customer" and "order" entities). In contrast, WETAS-Ontology is used for language-based learning tasks such as programming languages. This requires a completely different semantic view of the ontology because program code has "part-of" relationships (e.g. the word "WHILE" is part of a "while-loop" concept). Both systems thus exhibit the same limitation: they will only be effective for domains that conform to their semantic view of the ontology.

Our goal is to create a tool for authoring ITS for *any* domain. To do this its semantics must be easily extensible. Since different authors will have differing semantic requirements it must be possible for support for new domain types to be added without changes to the core system. To facilitate this we have developed an additional abstraction layer, *domain schema*. Domain schema define the system's behavior for a subset of domains that share a common structure and task type. New schema can be added to ASPIRE at any time by creating the appropriate XML documents and uploading them.

The schema automates the authoring process still further by reducing the vocabulary of the ontology to only those constructs required to author this domain type, making ontology authoring a much more well-defined task. We hypothesize that this will make domain modeling more feasible for novice modelers.

7 Domain Schema

A domain schema is a collection of XML documents that describe parts of the domain model that will be common to all domains of the same general type, such as critiquing a set of images. These documents tell ASPIRE how to perform many parts of the authoring process that would be otherwise performed manually. The documents are:

- Ontology schema (XML) and ontology generation rules (XSLT)
- Constraint generation rules (XSLT)
- Solution structure generation rules (XSLT)
- Student interface (HTML, with optional Java applets)

In the following sections we will use an example domain type to illustrate how domain schema work: for this domain type the student is shown a set of two or more images and is asked to choose the one with a particular characteristic and to identify features in the image that support their choice. This domain type could apply to many different subject areas (domains), such as: which of two buildings is Ionian; which x-ray image is better quality; which forest is the most damaged by acid rain; which painting is by Van Gogh; which x-ray shows an intestinal stricture. The interface consists of an applet for displaying, panning and zooming images, a control for selecting one of the images and a list of *features* that may or may not contribute to the decision; for each feature the student will select an appropriate *feature value*. Figure 5 shows this interface in action for an example of this domain type: x-ray power.
For each domain type the ontology will have the same basic form. The ontology schema defines this form by specifying concepts common to all domains of this type (typically the top of the ontology hierarchy), and describing the types of other concepts that the author can create and the relationship between these and the common concepts. Figure 6 shows the ontology for an ITS of the domain type "critique images", in this case the x-ray power domain, viewed using ASPIRE's

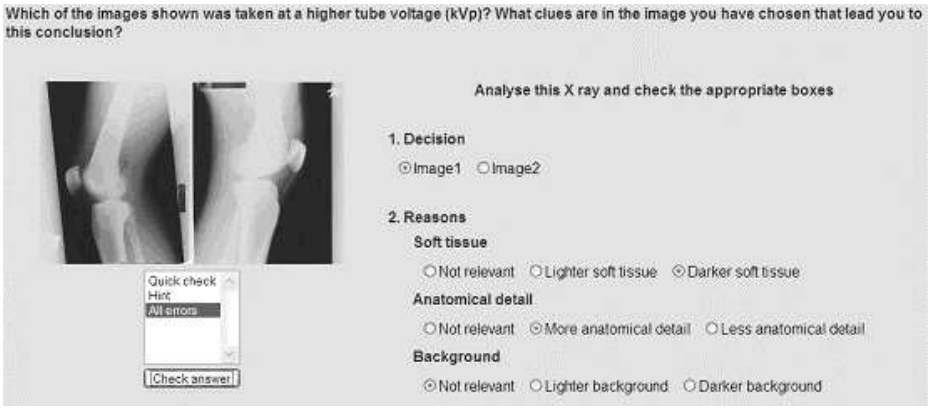

Figure 5: Example tutor for critiquing x-ray images

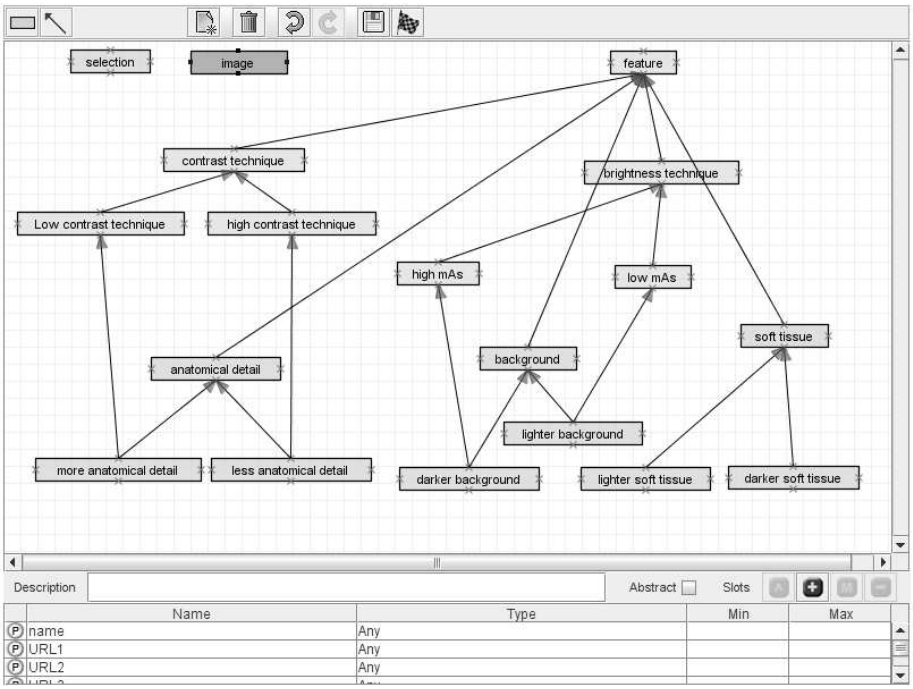

Figure 6: Ontology for x-ray power

ontology editor. All ontologies for this domain type contain the “feature”, “image” and “selection” concepts. The “feature” concept is then specialized for the actual features that the student will look for in this domain. The author can also specify *abstract* features if they wish; these are used for adding information that is common to more than one of the actual features. In Figure 6 the actual features are “anatomical detail”, “background” and “soft tissue”; abstract features are “contrast technique” and “brightness technique”. Each feature is then further specialized into *feature values*, which are the values the student can choose between, such as “more anatomical detail” and “lighter soft tissue”. The “image” concept is used to describe the images being shown to the student, in terms of the features present in this image (whether or not they contribute to the correct answer). Finally, the “selection” concept represents the choice the student must make between images.

The ontology schema describes the concept types the author can create (in this case *feature* and *feature value*). For each it also describes the attributes of that concept the author will be required to provide; for this domain a *feature* can have two feedback messages; *hint*, used when the student has overlooked this feature, and *wrong* for when the feature has been erroneously used. Similarly a feature value has a summary and detailed feedback message, and another (positive) to be displayed as reinforcement when the student has correctly answered the question. Finally, the author can specify that one concept is an example of another; in Figure 6 “les anatomical detail” is an example of “high contrast technique”. Once the author has

filled in the details for the features and their possible values, the information is saved as an XML document and converted to a standard ASPIRE ontology using XSLT.

The ontology is then converted to constraints using an XSL transform. This XSLT encodes the semantic interpretation of the ontology by specifying how each concept should be turned into one or more constraints. For the domain type under discussion the constraint generation rules are as follows:

1. **Correct selection:** for each "selection" concept check the student has supplied the correct selection value
2. **All features specified:** For each feature, if a value is specified in the ideal solution, the student must also have specified a value
3. **No extraneous features:** for each feature, if the student has specified a value, the ideal solution must also specify a value
4. **Correct feature value:** If the student has specified a feature value, and one was required, is it the same as that in the ideal solution
5. **Feature value supports selection:** if the student has selected a feature value that is present in their chosen selection, check that the selection is correct.

For each constraint the hint and feedback messages (for the concept from which it is generated) are incorporated into boilerplate text to give the actual messages the user will see when the constraint is violated. By carefully wording the feedback messages for each concept the author can ensure that the messages in the generated constraints are as required; this obviates the need to edit the constraints directly at any time. For this domain type the semantics are very straightforward. We are using domain schema to develop VIPER (Virtual Instructional and Practice Educational Resource) in conjunction with the Canterbury Polytechnic and Institute of Technology (CPIT). For this project there are five domain types, all of which are visual: critique images; label an image; identify a feature in the image (i.e. point to it); perform measurements on an image; experiment with the parameters of an image. In all cases the domain model is feature-based, and the semantics are straightforward as a result. The semantics for other domain types may be more complex.

Another domain type we are developing is programming languages. In this type of tutor the student is given a task to perform where they must write a snippet of code in free text form. The ontology for this type of ITS describes the grammar of the language being used. For example, consider the "logical expressions" domain described previously. In this domain each concept represents some part of the language (e.g. "conjunct"); concept properties represent the "part-of" relationship between a concept and the language constructs that make up that concept; for example a conjunct consists of an expression, followed by "and" followed by a second expression). The constraint generation rules for checking semantics of this domain type are as follows:

1. **Concept necessary:** for each concept, if it appears at least once in the ideal solution, it must also appear in the student solution;
2. **Concept superfluous:** for each concept, if it appears at least once in the student solution, it must also appear in the ideal solution;

3. **All concept instances present:** for each instance of each concept in the ideal solution where the student solution contains at least one instance of this concept, there must exist an equivalent instance in the student solution;
4. **No concept instances superfluous:** for each instance of each concept in the student solution where the ideal solution contains at least one instance of this concept, there must exist an equivalent instance in the ideal solution;
5. **Correct components:** for each concept instance in the student solution, if all but one component is equivalent to an instance in the ideal solution, the remaining component must also be equivalent.

For this domain the author describes each of the concepts in the same way as they would describe a grammar (e.g. in BNF). However, this is not sufficient because they also need to define equivalence. For example, "dog and cat" is equivalent to "cat and dog". They do this by defining additional concepts. In the previous example, conjunction is defined twice, with one definition being the exact reverse of the other. Each concept can then specify an "is equivalent to" relationship with another. In more complex cases the concept may be one that does not already appear in the grammar. For example, for logical expressions we can define de Morgan's law:

$$\overline{(A \wedge B)} \equiv \overline{A} \vee \overline{B} \quad (1)$$

We specify this law by defining both de Morgan forms and indicating they are equivalent. The constraint generation rules then use this information as follows. First, whenever a concept detected in one solution (e.g. the ideal solution) is being looked for in the other solution (i.e. the student solution), the default logic is to look for the exact same concept instance in both solutions. However, if the concept is one for which an equivalent form exists, the constraint will instead check that either the same concept instance exists in the other solution *or* an *equivalent* concept instance exists. Second, when checking for a particular concept instance, the constraint will also check whether it *forms part of* another concept that takes part in an equivalence relationship, and the alternate form exists in the other solution. If so, the check is dropped. For example, when checking for all "and"s, if the "and" in question is part of a De Morgan form and the student used the alternate form, the check for "and" will be dropped. We are currently evaluating this domain type in the areas of logical expressions, Java and SQL. This approach is also potentially useful for natural languages, provided the domain is sufficiently constrained. We are also exploring this possibility.

Another example of a completely different domain type is arithmetic procedural domains, such as multi-column addition. These can be catered for by extending the framework described as follows. First, for such domains the properties of a concept must be able to be collections. For example, an addition problem is made up of a collection of columns; each column contains a carry, a collection of addends and a sum. Second, the author must be able to specify arithmetic value restrictions for properties. For example (again from multi-column addition):

$$sum(n) = [carry(n) + SUM(addends(n))] \; MOD \; 10 \quad (2)$$

$$carry(n) = [carry(n+1) + SUM(addends(n+1))] \; DIV \; 10 \quad (3)$$

Note that *n* is the column number (more generally, *n* is the instance number of the object being considered). SUM and DIV are built-in primitives. As well as giving the formula for the restriction, the author also specifies two associated feedback messages: one that describes what the restriction means in words (used to correct the student when they violate the restriction) and one that describes the dependencies implied by the RHS of the restriction (used to indicate why the student should not be specifying this value yet, because the restriction cannot be tested). The constraints are now generated from both the concepts in the ontology plus the restrictions, as follows:

1. **All values specified:** For each concept instance, check whether this instance has been completed, e.g. "*You have not filled in the sum for column 3.*" Note that the restrictions imply dependencies between concept instances, which also need to be checked. If the dependent concept instances are not complete yet this constraint will not be relevant.

2. **Ordering:** For each concept that is on the LHS of a restriction, if the student has supplied an instance of this concept, check that the necessary parts in the RHS have been specified and give the "dependency" error if not, e.g. "*You cannot compute the carry for a column until you have completed the column to the right*", and "*You cannot compute the sum for a column until you have completed the column to the right.*"

3. **Correct value:** For each concept that is on the LHS of a restriction, test its value, and give an error if wrong, e.g. "*Check your sum in column 3. The sum should add up to the sum of addends in this column, plus the carry, if any*", or "*Check the value of the carry in column 2. The carry should be 1 if the addends and carry in the next column to the right add up to 10 or more.*"

This logic is sufficiently general to apply to other arithmetic domains, such as fraction addition.

These examples illustrate how a variety of different domains can be grouped into higher level "meta-domains" that share a similar task and can have the same form of constraints generated for them. We intend to investigate other domains to see how general this approach is. In particular we are interested in whether a domain type can be produced that can be generally applied to programming languages; the difficulty here is being able to capture alternative ways to solve the same problem and, in particular, how to cater for concepts such as variables, which can be arbitrarily named and may be used in very different ways to solve the same problem. These are known issues with developing domain models for programming languages, and it remains to be seen how much of a challenge they pose to the domain schema approach. In particular, unlike with WETAS-Ontology, it is the intent with domain schema that the generated constraints are final; the author should never need to view or change them. This may prove infeasible for some domains, particularly if they are more open-ended and hence the scope of potential domain models cannot be so readily constrained compared to more formal domains.

We are also interested in whether mathematics-based domains such as geometry, projectile physics and structural engineering can potentially be authored via a single domain schema. The limits of the approach are currently unknown (in particular, for what classes of domain is it *not* possible to generate all of the required constraints from ontology). We will gather evidence to answer this question as we explore more domains.

So far we have attempted to craft domain schema (and the constraint generation process in particular) by hand. Another interesting research question is whether the mapping from ontology to constraints might be able to be bootstrapped by inferring it from user data in a manner similar to [Harrer, 06] or [Soller, 03]. This might lower the skills bar for creating new domain schema.

Conclusions

ITS authoring is a difficult task. The WETAS tutoring shell dramatically reduces the effort required to build a tutor in that it reuses the "nuts and bolts" of the ITS implementation, but this still leaves the most difficult task, domain authoring. We introduced WETAS-Ontology, a tool that enables ITS authors to model the domain graphically using ontology. A pilot study at an e-learning summer school showed whilst this approach did not suit everyone, some students were able to develop domain models extremely quickly using this system; one group of students developed a fully working tutoring system in around eight hours. This represents a significant leap in authoring efficiency compared to more traditional methods of tutor development. In both the practical exercise and the project students developed simple tutors in a very short space of time. The reasons for this are threefold. First, WETAS removes all of the domain-independent authoring tasks. Second, the ontology helps the students visualize the model as they build it. Finally, the constraint generator removes the need to encode the constraints, which requires programming knowledge and can cause the author to waste time debugging errors. However, the generator has another important benefit; it removes the need for the author to decide what aspects of each concept need to be tested. This has the effect of reducing the task by a factor of five, this being the average number of constraints generated per node. Unfortunately this advantage comes at a cost: the semantics that can be represented in the ontology are somewhat limited. More importantly, they assume a particular semantic interpretation of the ontology. To overcome this limitation we extended our approach by adding "domain schema", a framework that allows the generic ITS authoring tool to be tailored to specific domain types. This approach allows the system to apply the specific semantic interpretation required for a particular domain, but the tool is still general in the sense that it can be readily extended to support new domain types. Domain schema also ease the task of ontology authoring by restricting the ontology vocabulary to just those concept types required for the type of domain being modeled.

Developing new domain schema is itself a difficult task; there is no "right" way to develop ontology, and correspondingly it is not always immediately obvious what the ontology schema for a class of domains should be. It is therefore envisaged that the creation of new domain schema will always be the job of an expert in Intelligent Tutoring Systems, while the task of creating actual domains will be that of domain experts. In the medical examples given the first author developed the ontology

schema required in consultation with domain experts at CPIT, and the Intelligent Computer Tutoring Group at the University of Canterbury provided the software required to implement the rest of the domain schema (Java applets etc); the domain models themselves were developed by CPIT medical training staff. By adopting this approach of creating new domain schema every time one is needed to implement a new ITS, we will develop a growing library of schema that will be used many times over and more than pay back the effort required to build them. In doing so we hope to empower an increasing number of educators to create ITS for their classrooms.

Intelligent tutoring systems are a promising tool for delivering education remotely. To date a key problem has been the effort required to build such systems. This can only be overcome by building tools that decrease both the effort and complexity of the task. Authoring systems like WETAS-Ontology have the potential to bring ITS authoring within the reach of teachers and thus to make widespread deployment of ITS feasible in the near future.

Acknowledgements

Research into domain schema was supported by a grant from the New Zealand Tertiary Education Commission Innovation Development Fund.

References

[Ainsworth, 04] Ainsworth, S. E., Grimshaw, S.: Evaluating the REDEEM Authoring Tool: Can Teachers Create Effective Learning Environments? International Journal of Artificial Intelligence in Education 14(3): 279-312.

[Anderson, 95] Anderson, J. R., Corbett, A. T., Koedinger, K.R., Pelletier, R.: Cognitive Tutors: Lessons Learned. Journal of the Learning Sciences 4(2): 167-207.

[Baghaei, 06] Baghaei, N., Mitrovic, A.: A Constraint-based Collaborative Environment for Learning UML Class Diagrams, In Proc. ITS2006, Taiwan, 2006, 176-186.

[Barros, 01] Barros, B., Mizoguchi, R., Verdejo, M. F.: A platform for collaboration analysis in CSCL. An ontological approach, In Proc. Tenth international conference on Artificial Intelligence in Education, San Antonio, USA, 2001, 530-532.

[Dimitrova, 03] Dimitrova, V.: STyLE-OLM: Interactive Open Learner Modelling. International Journal of Artificial Intelligence in Education 13: 35-78.

[FranzInc] http://allegroserve.sourceforge.net/

[Harrer, 06] Harrer, A., McLaren, B. M., Walker, E., Bollen, L., Sewall, J.: Creating Cognitive Tutors for Collaborative Learning: Steps Toward Realization. User Modeling and User-Adapted Interaction 16: 175-209.

[Knublauch, 04] Knublauch, H., Fergerson, R. W., Noy, N.F., Musen, M.A.: The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications, In Proc. Third International Semantic Web Conference, Hiroshima, Japan, 2004.

[Koedinger, 97] Koedinger, K. R., Anderson, J. R., Hadley, W.H., Mark, M.A.: Intelligent Tutoring Goes To School in the Big City. International Journal of Artificial Intelligence in Education 8: 30-43.

[Koedinger, 04] Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B., Hockenberry, M.: Opening the Door to Non-programmers: Authoring Intelligent Tutor Behavior by Demonstration, In Proc. 7th Int. Conf. Intelligent Tutoring Systems, Maceio, Brazil, 2004, 162-174.

[Martin, 00] Martin, B., Mitrovic, A.: Tailoring Feedback by Correcting Student Answers, In Proc. Fifth International Conference on Intelligent Tutoring Systems, Montreal, 2000, 383-392.

[Martin, 02a] Martin, B., Mitrovic, A.: Authoring web-based tutoring systems with WETAS, In Proc. International conference on computers in education, Auckland, 2002, 183-187.

[Martin, 02b] Martin, B. Mitrovic, A.: Automatic Problem Generation in Constraint-Based Tutors, In Proc. Sixth International Conference on Intelligent Tutoring Systems, Biarritz, 2002, 388-398.

[Martin, 02c] Martin, B. Mitrovic, A.: WETAS: A Web-Based Authoring System for Constraint-Based ITS, In Proc. Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Malaga, 2002, 543-546.

[Martin, 03] Martin, B., Mitrovic, A.: ITS Domain Modelling: Art or Science?, In Proc. International Conference on Artificial Intelligence in Education, AIED2003, Sydney, Australia, 2003, 183-190.

[Mitrovic, 99] Mitrovic, A., Ohlsson, S.: Evaluation of a Constraint-Based Tutor for a Database Language. International Journal of Artificial Intelligence in Education 10: 238-256.

[Mitrovic, 03] Mitrovic, A., Koedinger, K. R., Martin, B.: A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modelling, In Proc. Ninth International Conference on User Modeling UM 2003, 2003, 313-322.

[Mitrovic, 06a] Mitrovic, A.: Large-Scale Deployment of three intelligent web-based database tutors, In Proc. Proceedings of ITI, Cavtat, Croatia, 2006, 135-140.

[Mitrovic, 06b] Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., Holland, J.: Authoring constraint-based tutors in ASPIRE, In Proc. ITS 2006, Taiwan, 2006, 41-50.

[Mizoguchi, 00] Mizoguchi, R., Bourdeau, J.: Using Ontological Engineering to Overcome Common AI-ED Problems. International Journal of Artificial Intelligence in Education 11: 107-121.

[Murray, 97] Murray, T.: Expanding the Knowledge Acquisition Bottleneck for Intelligent Tutoring Systems. International Journal of Artificial Intelligence in Education 8: 222-232.

[Ohlsson, 94] Ohlsson, S.: Constraint-Based Student Modeling. Student Modeling: The Key to Individualized Knowledge-Based Instruction. J. Greer and G. McCalla. New York, Springer-Verlag: 167-189.

[Ohlsson, 96] Ohlsson, S.: Learning from Performance Errors. Psychological Review 3(2): 241-262.

[Puerta, 92] Puerta, A. R., Musen, M.: A multiple-method knowledge acquisition shell for the automatic generation of knowledge-acquisition tools. Knowledge Acquisition 4: 171-196.

[Soller, 03] Soller, A., Lesgold, A.: A Computational Approach to Analyzing Online Knowledge Sharing Interaction, In Proc. 11th international conference on Artificial Intelligence in Education, Sydney Australia, 2003, 253-260.