Teaching Strategies Ontology Using SWRL Rules

Eric WANG, Leila KASHANI, and Yong Se KIM
Creative Design and Intelligent Tutoring Systems Research Center
Sungkyunkwan University, Suwon, Korea
ewang@skku.edu, kashani@skku.edu, yskim@skku.edu

Abstract. We present an ontology to represent generic teaching strategies in an intelligent tutoring system framework. The learning contents, learner's state, and system actions are modeled in just enough detail to support the definition of teaching strategies. Teaching strategies are organized into a hierarchy of teaching goals, and are represented using SWRL rules.
Keywords: Ontology Modeling, Teaching Strategy

1. Introduction

We investigate a domain-independent framework for intelligent tutoring systems. Within this framework, we are studying the representation of generic teaching strategies, including an explicit representation of each strategy's computational procedures. The eventual goal of this work is the development of an ontology of reusable teaching strategies that could be automatically converted into various rule engine formats, and executed directly. That is, we propose to extract much of the specification of each strategy's executable code from the esoteric realm of "programming", to the more accessible realm of a public, editable, and machine-understandable ontology.

Ikeda and Mizoguchi [5] have previously considered a domain-independent approach to ITS, including a hierarchical representation of teaching goals and teaching strategies. While they informally describe rules that choose among competing strategies, they do not describe the explicit representation of individual strategies using rules.

2. Teaching Strategies Ontology

Recent advances in Semantic Web technology, particularly the development of public tools for editing rules in Semantic Web Rule Language (SWRL) [4], have raised the possibility that ontology editors can now be used to model rules in a convenient manner, and thereby encode procedural knowledge. In this paper, we present an ontology for teaching strategies that uses SWRL rules to encode strategies, give a concrete example of a SWRL rule for a teaching strategy, and discuss some implementation issues for the supporting framework to enable these rules to be executed. We use Protégé with OWL plugin for ontology modelling, and Protégé's SWRLTab plugin for editing SWRL rules.

One of this paper's objectives is to evaluate the suitability of SWRL, and the maturity of its tools, for this kind of application. We therefore restrict the detail of our ontology to be just sufficient for representing some teaching strategies in a concrete manner, rather than as a comprehensive ontology of all possible teaching strategies. Mizoguchi *et al.* [7] presents a more detailed list of ontology concepts for ITS.

2.1 Aspects of ITS Modeling

The ITS framework is usually decomposed into a few major subsystems, including learning contents (domain model), learner model, pedagogical module, and user interface. Such a perspective has been found to be practical and effective for implementation and run-time execution.

However, when designing an ITS, we find that some design considerations cannot be neatly localized within any one subsystem. Instead, they cut across all subsystems. For example, if we commit to a particular learning theory during the design stage, which may be characterized by a set of data fields, then this entails customizing every subsystem, in parallel: the learner model must store these data fields, domain model elements must provide them, some pedagogical knowledge must use them, and so on. Following the research area of aspect-oriented programming, we call such cross-cutting concerns *aspects*. Aspects tend to be disjoint, and multiple aspects can be combined within a single ITS.

From past ITS and pre-ITS research, we identify 4 aspects in ITS design, roughly in increasing order of sophistication in modelling.

(a) The **domain coverage aspect** records only whether elements of the domain knowledge have been presented or visited, or not. It makes no attempt to model the learner's understanding or cognitive processes. This aspect characterizes some pre-ITS systems, such as online textbooks, computer-aided instruction (CAI) systems, and hyperlinked multimedia applications.

(b) The **history aspect** records detailed learner and system actions, e.g. to generate a log file that can support further analysis.

(c) The **learner assessment aspect** deduces the learner's cognitive model of the domain, and maintains an internal representation of the learner's expertise. This aspect is used by model tracing tutors, and has been used to support identification of specific bugs and their remediation.

(d) The **affect aspect** models the learner's affective state. For this paper, we consider the learner's levels of confidence and effort [6].

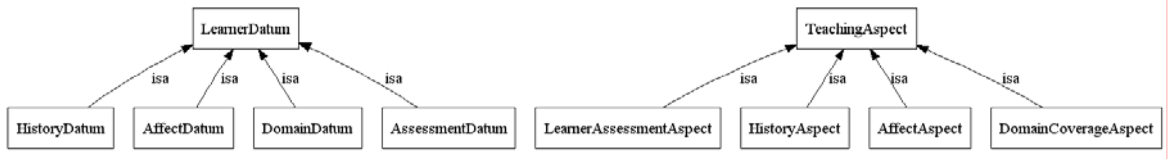

Figure 1. Aspects in ITS modeling

From the execution-oriented perspective of the ITS framework, we define an ontology base class LearnerDatum for learner's state data, and a TeachingAspect base class for teaching goals. These base classes are then partitioned by each of the 4 aspects into parallel sets of subclasses, as shown in Figure 1.

2.2 Learning Objects and Learner State

An ITS's domain knowledge consists of learning objects, which are enumerated by the domain experts. Murray [8] observes that most ITS authoring tools provide a hierarchical organization among learning objects; he advocates a 5-level organization. We model a more generic representation that supports any number of levels, and any graph-like structure, shown in Figure 2. Each LearningObjectGroup has 0 or more links to "other" groups. To support different types of links for different modelling purposes, we reify the links as a LearningGroupLink class. Each link references 1 or more "other" groups. We define link subclasses for *prerequisite*, *subgroup*, and *difficulty* relations. The EasierDifficultyLink and SimilarDifficultyLink subclasses allow a course designer to organize the learning contents into a lattice structure of difficulty levels. Note that learning object groups are not disjoint, i.e. a given learning object may appear in 2 or more groups.

The learner's state is represented as a stream (time-sorted set) of LearnerDatum individuals, shown in Figure 2. Each LearnerDatum associates some dynamic data with 1 or more specified learning content objects. In particular, the HistoryDatum subclass records a numeric score, while the AffectDatum records quantitative measures of confidence and effort.

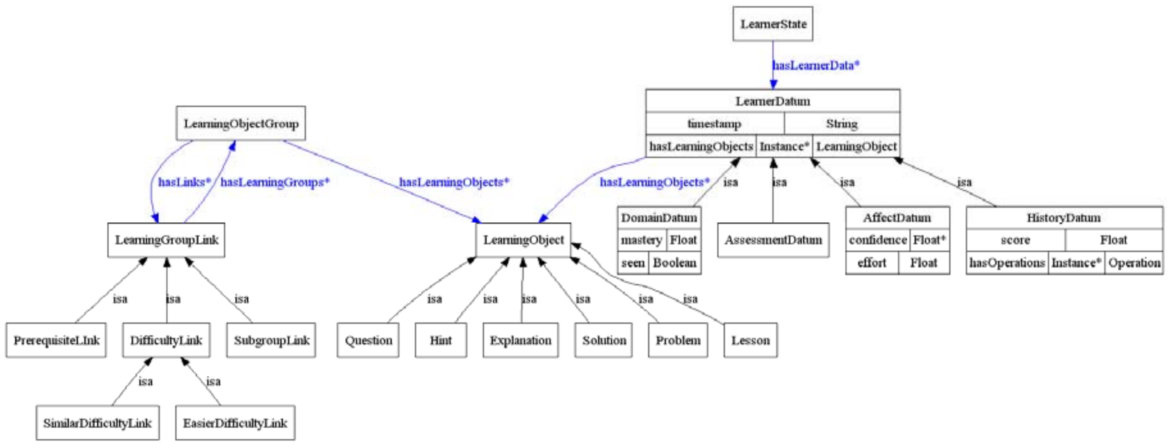

Figure 2. Learning objects and learner state

2.3 Teaching Goals and Strategies

Teaching strategies are derived from teaching goals. Teaching goals are typically organized into a hierarchy of subgoals. Other ITSs have derived their hierarchy of teaching goals from various learning theories. The authors of the IRIS shell [1] developed the Cognitive Learning from Automatic Instruction (CLAI) theory, which organizes goals based on Gagné's cognitive processes and instructional events and their own instructional actions. Ikeda and Mizoguchi [5] use a 4-level organization of goals and strategies, but without specifying how each level is obtained.

We model a generic hierarchical representation of teaching goals, shown in Figure 3, which can represent subgoals to any depth. Each teaching goal has 0 or more subgoals, and 0 or more teaching strategies, which are leaf nodes in the graph. Each teaching strategy is represented by a SWRL rule, as described in the next section.

3. Teaching Strategy Definition using SWRL Rules

We explicitly encode teaching strategies as SWRL rules. Briefly, a SWRL rule has an IF-THEN format, and its head and body consists of atoms that evaluate elements in the knowledge base, using class and property names defined in the ontology, or perform basic computations, using SWRL's own library of built-in functions. Atoms in the body can be thought of as data queries into the knowledge base, while atoms in the head represent assertions that modify the knowledge base.

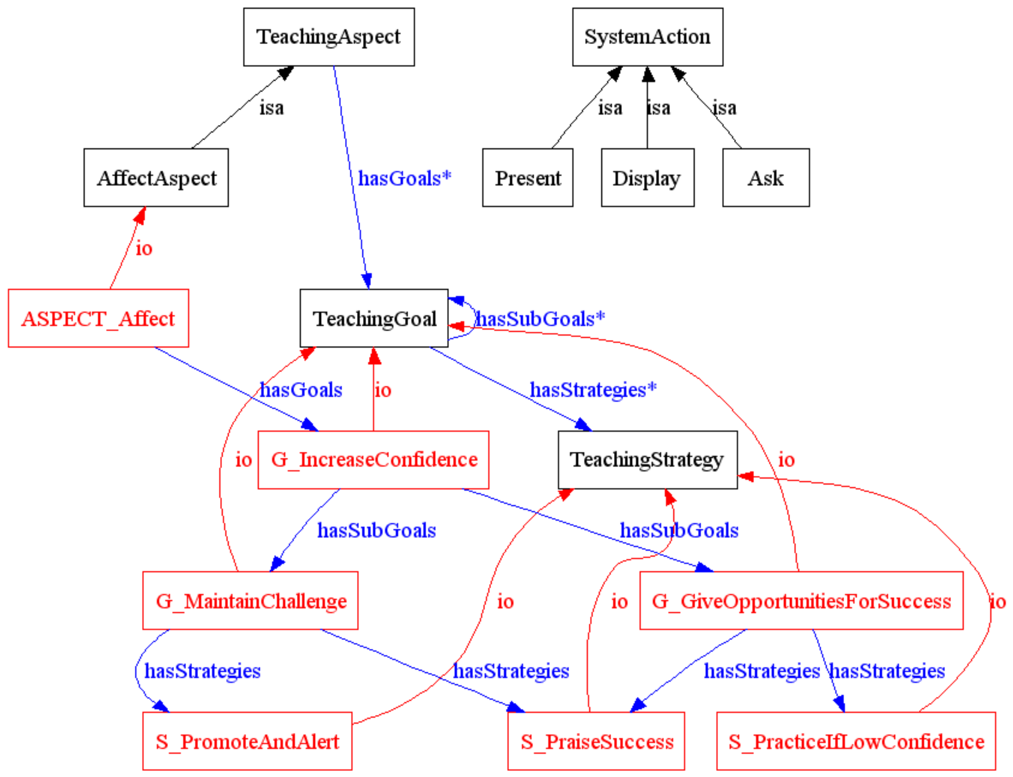

Figure 3. Teaching goals, teaching strategies, and system actions

Note that the SWRL standard specifies the language semantics only, without providing any inference engine. It is up each implementer to choose an engine, implement a conversion to and from SWRL, implement all of SWRL's built-in functions, and enforce the above semantics. We use Jess [3] as the inference engine, and we are exploring numerous strategies to convert from OWL + SWRL to Jess.

3.1 Predicates for Rule Body Atoms

Atoms in the body of a rule (left-hand side, or IF portion) can represent queries into the knowledge base. One of the fundamental ontology queries is a class membership test, which SWRL can represent directly. However, teaching strategies often involve simple tests that evaluate a datum and return true or false, such as a numeric comparison. We must convert from a true-false perspective to a class membership test. We define a Predicate class that has one SWRL rule, which encodes a query for the desired type of datum. Simple tests such as numerical, logical, or string comparisons can be easily written using SWRL's library of built-in functions. Conceptually, whenever a new datum *d* of the proper type is generated (by any means), this Predicate's rule automatically fires, and either adds *d* as an instance-of itself (since a Predicate subclass represents a set of individuals), or not.

For example, the **low-confidence-p** subclass of Predicate, shown in Table 1, encodes the pedagogical policy that a confidence value less than 0.50 is considered "low". This is implemented as a SWRL rule that looks up each AffectDatum's confidence field. The advantage of modelling these policies as SWRL rules is that each policy's computation is explicitly represented in the ontology, and can be viewed and edited, as well as reasoned about by other applications.

Predicate subclass	Pedagogical policy	SWRL Rule
low-confidence-p	confidence less than 0.50 is low	AffectDatum(?A) ∧ confidence(?A, ?c) ∧ swrlb:lessThan(?c, ?V50) → **low-confidence-p**(?A)
passing-score-p	a score of 75% or higher is passing	HistoryDatum(?H) ∧ score(?H, ?s) ∧ swrlb:greaterThanOrEqual(?s, ?V75) → **passing-score-p**(?H)

Table 1. Predicates using SWRL rules

Note that the Protégé SWRLTab plugin can't edit SWRL literal values yet. This is a limitation of the current version of the SWRLTab tool. We work around this by the syntactic convention that a SWRL "variable" of the form ?V*xx* actually means the literal (constant) value 0.*xx*. For example, ?V75 denotes the constant value 0.75.

3.2 Result Sets for Rule Head Atoms

Atoms in the head of a rule (right-hand side, or THEN portion) can represent modifications to the knowledge base, such as assertions that add new facts to the knowledge base. To reify this interpretation in the ontology, we define a ResultSet base class that represents any set of new assertions. From this, we derive an EligibleSet subclass that represents the set of learning objects that are eligible to be selected as the learner's "next" object. Teaching strategy rules that handle curriculum planning will explicitly refer to the EligibleSet class to determine what content to show next. Rules that refer to EligibleSet in their heads will populate the EligibleSet by granting (or denying) membership to certain learning objects; an example of such a rule is presented in Section 4. Subsequently, other rules can refer to EligibleSet in their bodies to further process the set of eligible objects, e.g. to select one or more for display.

Note that some additional infrastructure is needed to fully implement the ResultSet mechanism, e.g. precise temporal control to clear this set before each recalculation. We assume that this infrastructure will be provided at the implementation level. This can be justified because the broader mechanism of treating atoms in rule heads as assertions must itself be implemented outside of SWRL. The supporting infrastructure issues are discussed further in Section 5.

3.3 System Actions

The SystemAction class, shown in Figure 3, represents atomic system actions. **Display** causes some media content to appear on the screen in a non-interruptive mode, e.g. by streaming into an existing window. **Present** shows some content in an interruptive mode, e.g. as a pop-up window. **Ask** represents a system-initiated dialogue, in which some content is shown in an interruptive mode, and the selected response is returned to the system.

We assume that these system actions are straightforward to implement using standard user interface components. In future work, we will extend the ontology to explicitly model the semantics of each action in terms of lower-level characteristics such as intrusiveness and expected response types. This would support higher-level strategy rules that choose between alternative system actions.

4. Example of Teaching Strategy Definition

The affect aspect models a learner's affective state, including confidence level [2][6]. A primary teaching goal within this aspect is to increase the learner's confidence. We model this teaching goal as shown in Figure 3.

- The top-level teaching goal is to **Increase Confidence** (Figure 3, center). This is decomposed into two subgoals.
  - The first subgoal is to **Give Opportunities For Success** (lower right). This is decomposed into two teaching strategies.
    - The **Praise Success** strategy presents a praise message associated with one or more of the learner's previous successful problems, especially on each successful completion of a problem.
    - The **Practice If Low Confidence** strategy applies whenever a learner's score is acceptable ("passing") but her confidence is "low", and results in assigning more problems of similar difficulty level. Note that this implies that the difficulty level will *not* be increased. The rationale is to avoid discouraging the learner unnecessarily.
  - The second subgoal is to **Maintain Challenge** (lower left). This is decomposed into two strategies, of which **Praise Success** is the same as above.
    - The **Promote and Alert** strategy applies when the learner's score and confidence are both high enough to earn a "promotion" to a higher difficulty level. This strategy presents a congratulatory message, and a warning that the difficulty level has, in fact, increased. The intent is to draw attention to the transition between difficulty levels.

The SWRL rule implementation of the **Practice If Low Confidence** strategy is shown in Table 2. Note that the rule has been rearranged, compared to the text-based description given above. This is an implementation technique in rule programming, arising from the fact that a SWRL rule must have a simple head.

Description	**SWRL Rule**
IF	
P is a Problem and …	Problem(?P) ∧
P's score is passing and …	HistoryDatum(?H) ∧ hasLearningObjects(?H, ?P) ∧ passing-score-p(?H) ∧
P's confidence is low and …	AffectDatum(?A) ∧ hasLearningObjects(?A, ?P) ∧ low-confidence-p(?A) ∧
P is in group G and …	LearningObjectGroup(?G) ∧ hasLearningObjects(?G, ?P) ∧
group H is of similar difficulty level to G and …	hasLinks(?G, ?L) ∧ SimilarDifficultyLink(?L) ∧ hasLearningGroups(?L, ?H) ∧
Q is any object in H and …	hasLearningObjects(?H, ?Q) ∧
Q is different from P	differentFrom(?P, ?Q)
THEN	
add Q to the eligible set	→ EligibleSet(?Q)

Table 2. SWRL rule implementation of the **Practice If Low Confidence** teaching strategy

5. Issues for Implementation

We assume that a conversion from OWL + SWRL to Jess is available (including a complete implementation of all SWRL built-in functions, or conversion to equivalent Jess language mechanisms). SweetRules [9] provides this conversion, but it involves a fairly complex installation of many third-party software components. The current Protégé + SWRLTab tool set does not provide a conversion to Jess, although this is currently being developed, and should be available soon.

In this section, we consider some issues in implementing a rule execution module for an ITS that encodes its rules using SWRL. As noted previously, additional infrastructure must be implemented at the Jess level to enforce SWRL's semantics, and to provide certain extensions to SWRL.

5.1 Effecting

A practical ITS must interact with a learner and evolve (change state) over time. Hence, its rules must have a capability to cause side-effects. Useful knowledge-related effects include modifying the knowledge base by adding or retracting facts, and changing data fields. Procedural effects include system actions such as user interaction events.

While SWRL does not define the semantics of rule atoms that represent effects, we can write such atoms in SWRL rules, using exactly the same syntax as atoms that represent class membership tests. Such atoms must be specially converted to Jess function calls, instead of to rule patterns that look up facts. We can adopt a simple convention that every rule head is always converted to an effect. Then we may implement a library of effects as functions in Jess. Knowledge-related effects map trivially to built-in Jess functions, while system action effects could be implemented as Jess functions that send an appropriate message to the user interface module.

5.2 Temporal control

Rules may depend on the results of other rules. In the example shown above, the **Practice If Low Confidence** rule uses the results of the **low-confidence-p** and **passing-score-p** rules. This defines a hierarchy of dependencies among rules. Hence, there is a need for a mechanism to specify the order in which subsets of rules may be executed. The dependencies could be computed automatically during the conversion to Jess, and automatically mapped to Jess language features such as salience and modules.

Rules may also depend on classes (sets of facts). The ResultSet mechanism evaluates a result by populating a class, which presumes that the class has been properly initialized (emptied). This can be handled by adding a class-initialization rule in Jess, and defining a dependency from the ResultSet rule to this rule.

5.3 Continuous execution

To support a lengthy session of interaction with a learner, the rule execution module must run continuously, and it must accept new facts (data, events) from other system modules. We use a simple Jess self-repeating rule to provide continuous execution, and to poll a socket for incoming message strings, which are directly asserted as Jess facts.

6. Summary and Future Work

We have developed an ontology for representing teaching strategies, and we are exploring the extent to which SWRL could be used as an abstract "programming language" for defining teaching strategies. A small number of the resulting SWRL rules have been manually tested (by rewriting them directly in Jess). We are currently developing the infrastructure for a rule execution module in Jess.

A future consideration is the overall temporal context in which strategy rules will execute. There is a need for a richer specification of temporal context to provide control over the sequencing of strategy rules. One established approach is to adopt an event-handling architecture, similar to that used by most modern operating systems' graphical user interfaces. This involves enumerating a comprehensive set of system events, and then defining explicit sequences of events (or more generally, a control flow). To each event, we associate a dynamic set of rules. Within the ontology, teaching strategy rules specify the events during which they are to be activated. As part of the conversion to Jess, strategy rules are "registered" with their indicated events. At run-time, as each event occurs, the set of rules associated with that event are tested. This could provide a familiar idiom to organize the development of teaching strategy rules.

Acknowledgements

This research was supported by the Korean Ministry of Science & Technology through the Creative Research Initiative Program.

References

[1] Arruarte, A., Fernández-Castro, I., Ferrero, B., and Greer, J., "The IRIS Shell: How to Build ITSs from Pedagogical and Design Requisites", *Int'l Journal of Artificial Intelligence in Education*, **8**, 341–381, 1997.

[2] du Boulay, B., and Luckin, R., "Modelling human teaching tactics and strategies for tutoring systems", *Int'l Journal of Artificial Intelligence in Education*, Special Issue on Modelling Teaching, **12**, 235–256, 2001.

[3] Friedmann-Hill, E., *Jess in Action*, Manning, Greenwich, 2003.

[4] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M., "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", *W3C Member Submission*, www.w3.org/ Submission/2004/SUBM-SWRL-20040521/, May 2004.

[5] Ikeda, M., and Mizoguchi, R., "FITS: A Framework for ITS – A Computational Model of Tutoring", *Int'l Journal of Artificial Intelligence in Education*, **5**(3), 319–348, 1994.

[6] Keller, J. M., "Motivational design of instruction", in Reigeluth, C. M. (Ed.), *Instructional design theories and models: An overview of their current status*, Erlbaum, Hillsdale, 1983.

[7] Mizoguchi, R., Sinitsa, K., and Ikeda, M., "Task Ontology Design for Intelligent Educational/Training Systems", *3rd Int'l. Conf. on Intelligent Tutoring Systems, Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs,* Montreal, June 1996.

[8] Murray, T., "Principles for Pedagogy-Oriented Knowledge Based Tutor Authoring Systems", in Murray, T., Blessing, S., and Ainsworth, S (Ed.s), Chapter 15, *Authoring Tools for Advanced Technology Learning Environments*, Kluwer, Boston, 2003.

[9] SweetRules, http://sweetrules.projects.semwebcentral.org/.