Interactivity of Exercises in ActiveMath

Giorgi Goguadze [a], Alberto González Palomo [b], Erica Melis [c],
[a] *University of Saarland, Saarbrücken, Germany*
[b] *German Research Institute for Artificial Intelligence (DFKI) Saarbrücken, Germany*
[c] *German Research Institute for Artificial Intelligence (DFKI) Saarbrücken, Germany*
george@activemath.org

Abstract. Interactive exercising is one of the major ingredients of technology-enhanced learning. It reaches its full potential only, when appropriate feedback is given to the learner. This paper describes a principled approach for representing and processing interactive exercises in ActiveMath. This approach relies (1) on a modular architecture of the exercise player, (2) on a separation of different types of knowledge that have to be handled to generate feedback, and (3) on a generic representation of interactive exercises.

Keywords. architecture, intelligent scaffolding, exercise representation, web-based mathematics application

1. Introduction

Interactivity is one of the major advantages of technology-enhanced learning over traditional learning material. For an effective and interesting learning experience, a variety of types of interactive exercises is needed rather than multiple choice questions (MCQs) only. Moreover, it has been shown, that feedback is mandatory for fully exploiting the benefits of interactivity for learning [10].

Many e-learning systems do not provide feedback that goes beyond a correct/incorrect response referring to pre-defined results. Even in intelligent tutoring systems (ITS), this diagnosis is done for small domains only or feedback is authored.

Such a program is more powerful, if it uses reasoners such as a computer algebra system (CAS) to evaluate the input of the learner. An author can encode the exercise using the language of a CAS and connect to the CAS to "play" the exercise. This was our first approach in ActiveMath [1]. Previously, exercises in ActiveMath were programmed individually or as classes of exercises. Our experience is that such an approach is limited. On the one hand, it binds authors to the syntax of the computer algebra system. Moreover, CASs are not designed for representing human-like problem solving steps. Such exercises are not reusable

with other CASs, and therefore, the learning environment is bound to the particular CAS. On the other hand, it is not possible to apply different tutorial strategies to the same exercise and to adapt to the particular learner or learning situation.

In order to extend the types of range of interactive exercises to respond to author's wishes, to keep exercises modifiable and extensible, and to provide more elaborate feedback in ActiveMath, our goal was to allow for both, authored and generated feedback, for a variety of exercise types, and for a clear separation of functionalities necessary for interactive exercise. We report some main results on that way here.

This paper is organized as follows. After short preliminaries about the ActiveMath system and the OMDoc knowledge representation we introduce the extensions to the knowledge representation of exercises and show how this knowledge representation works in the ActiveMath exercise subsystem.

2. Preliminaries

ActiveMath is a web-based user-adaptive learning environment for mathematics [7]. It dynamically assembles courses of learning material for the individual learner according to her learning goals, preferences, and mastery of concepts. It makes suggestions on how to proceed with learning and navigation.

The information about the learner and his learning context is represented in a Learner Model. The dynamic assembling of courses is performed by the Tutorial Component which selects elements to be presented to the learner, according to the information from the Learner Model.

OpenMath *and* OMDoc *Representations for Mathematics* The content is annotated with metadata and stored in a knowledge base. The knowledge representation language in ActiveMath is OMDoc, a semantic markup language for mathematical documents [6]. OMDoc has evolved as an extension of OpenMath[1] which is an XML-representation standard for mathematical formulas. The main difference between OpenMath and other representation formats for mathematical formulas, such as Presentation MathML and LaTeX, is that OpenMath deals with the semantics of mathematical expressions rather than with their presentation only. OpenMath defines so-called Content Dictionaries in which mathematical symbols are declared and their semantics is defined. OpenMath formulas are tree like XML-structures of OpenMath symbols. The semantic representation of formulas allows for automatic translation of these formulas to (and from) the languages of different mathematical systems This provides the basis for interoperability of different computer algebra and other reasoning systems.

OMDoc defines learning objects (LOs), such as exercises, definitions, and some relations between them. Such LOs can consist of text mixed with formulas in OpenMath format.

We extended the OMDoc representation of the exercises to meet frequent usage requirements for interactive exercises. The new format is briefly described in the following section. A full account can be found in [4].

[1] see http://www.openmath.org for the standard specification

3. Knowledge Representation of Interactive Exercises

An exercise can be seen as a finite state machine of `interaction`s, either predefined or generated. An `interaction` is a state of an exercise in which an interaction of the learner with the system takes place.

The `interaction` contains `feedback` for the event that triggered it, an `interactivity assignment` that describes how to substitute certain parts of the feedback by interactive elements (e.g. replace a term by a blank), and one or more `transition map`s describing which `interaction` comes next depending on the learner's input.

We extended the knowledge representation of interactive exercises in [3] by introducing three layers of markup for each `interaction` of the exercise. The first layer consists of the content, i.e. text possibly mixed with formulas (`feedback`). The second layer attaches interactive elements of different types to the places in the content marked for interactivity (`interactivity assignment`). The third layer defines conditions to be satisfied in order to proceed to further interactions (`transition map`).

The element `feedback` contains textual content presented to the learner in the step. [2]

The optional attribute *`from`* allows to import the `feedback` elements from another interaction, and the attribute *`keep`* specifies whether the previous input plus feedback is shown after the step is performed or whether it is not.

The `interactivity assignment` attaches interaction types to the content of the `feedback` element. It defines which parts of the content become interactive and what types of interactivity is to be used, e.g. selection (presented as choice questions or drop-down menus), fill-in-blank, mapping, or marking.

The `transition map` contains the `condition` elements and a `default` for a default condition. Default condition is used if none of the other conditions are satisfied. Each condition has a pointer to an interaction to be executed, if this condition is satisfied.

Currently, the `condition` element is a statement that may contain predicates `syn_eq`, `num_eq` and `sem_eq`. `syn_eq` means 'syntactic equality' and calls a procedure that checks whether the input of the learner is syntactically equal to a given expression. `num_eq` means 'numeric equality' and calls a procedure that checks whether the input of the learner is a number and is numerically equal to a given number. If the user input differs from prescribed result less then a given epsilon, then the input is considered numerically equal. `sem_eq` means 'semantic equality' and calls a procedure that checks whether the input of the learner is the same mathematical object. For this evaluation a computer algebra system is used.

Each `interaction` can be annotated with metadata describing competencies, relations to concepts trained in the step, as well as difficulty, mastery assessment to be sent to the Learner Model.

Figure 1 shows a representation of an exercise step the rendering of which is shown in the Figure 2 (OpenMath formulas are abbreviated to linear notation for the lack of space).

[2] This is analogous to QTI's `material` element

```
<interaction id="int_1_hint">
<feedback from="int_1" keep="yes"/>
<feedback><CMP xml:lang="en">Try again:
diff((x*sin(x+1))+2,x)=diff("blank1"),x)+diff("blank2",x)</feedback>
<interactivity_assignment>
<blank for="blank1"/><blank for="blank2"/>
</interactivity_assignment>
<transistion_map>
<condition xref="int1_correct">
<composite><sem_eq>x*sin(x+1)</sem_eq><sem_eq>x</sem_eq></composite>
</condition>
<default xref="int1_wrong"/>
</transition_map>
</interaction>
```

Figure 1. Exercise step representation

In the Figure 1 `interaction` consists of two feedbacks, the first of which is imported from the previous interaction and kept in the next step. The second feedback is a hint statement and will dissappear in the next step. The `interactivity assignment` defines 2 blanks for those two parts of the formula marked with IDs. In the `transition map` one condition is defined, containing the semantic comparisons with the correct results and one default condition.

For all possible steps an exercise can contain fully authored `interaction` elements as well as an `interaction_generator` element that is used to generate `interaction`s automatically.

4. Authored vs Generated Answers

The architecture of the exercise system has the potential to process fully authored as well as (partially) generated exercise representations. Whether the interactions are generated depends on the availability of intelligent components that can help to diagnose errors of the learner in a particular learning domain and a component for generating feedback from the diagnosis.

Part of the feedback generation is the application of tutorial strategies. In order to allow for different tutorial strategies for the same exercise, these strategies have to be represented separately and applied to the exercise representation.

The exercise subsystem renders interactive steps, evaluates the learner's input, diagnoses and generates feedback, automatically generates interactions applies tutorial strategies for adapting to the learner and his learning situation, reports the achievements of the learner to the rest of the ActiveMath system.

Consider the two versions of an exercise in Figure 2 and 3, whose interactions were completely generated by a reasoning component and whose feedback was inserted automatically by two different tutorial strategies. Figure 2 shows the result of the application of the first strategy. In this strategy, when the learner asks for a hint, the system provides him with a simpler task by inserting the structure of the derivation rule to be applied here. In the second strategy, shown in the Figure 3, the needed derivation rule is formulated, and the learner is invited to try the same task again.

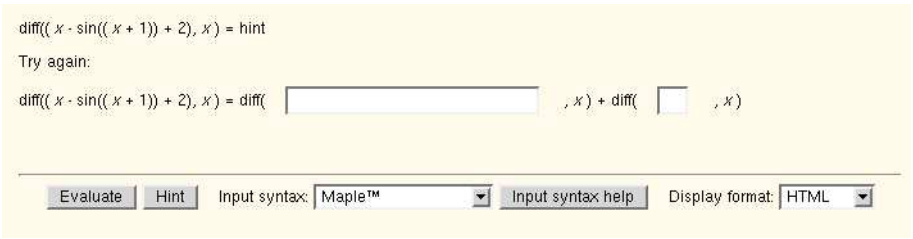

Figure 2. Partial insert - hint strategy

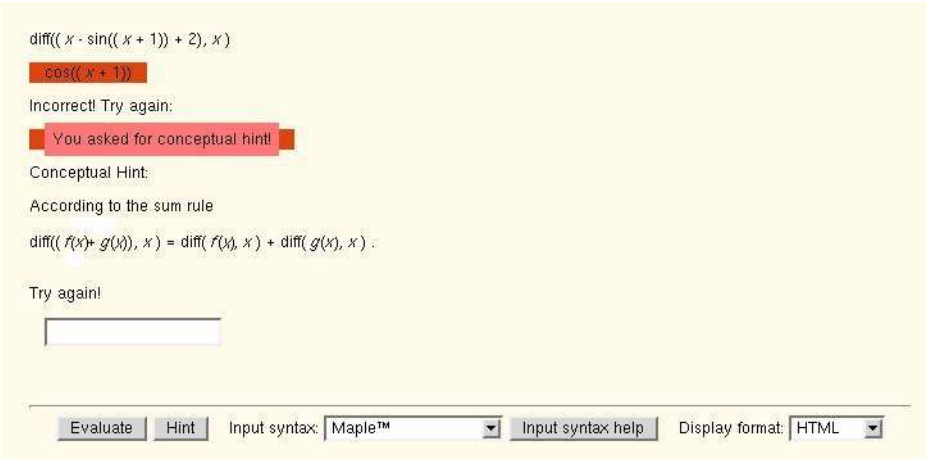

Figure 3. Conceptual hint strategy inserting the rule statement

The hint in the first strategy is generated via the reasoner that computes concrete functions which are inserted into the rule application. The hint in the second strategy does not depend on the concrete function and does not have to be generated by the reasoner each time. The reasoner determines the appropriate rule and provides its link to the tutorial strategy.

5. Separation of Functionalities for Scaffolded Exercising

It is a well-known fact that modifications and extensions are much easier to realize, if different types of functionalities and knowledge are separated by design and implementation. Therefore, the ActiveMath exercise player has a modular architecture.

The diagram in Figure 5 displays a simplified architecture and information flow. The `Exercise Manager` is the main component of the exercise subsystem responsible for managing the collection of the exercise steps and for executing the exercise steps. In the initialization process, the `Exercise Manager` receives

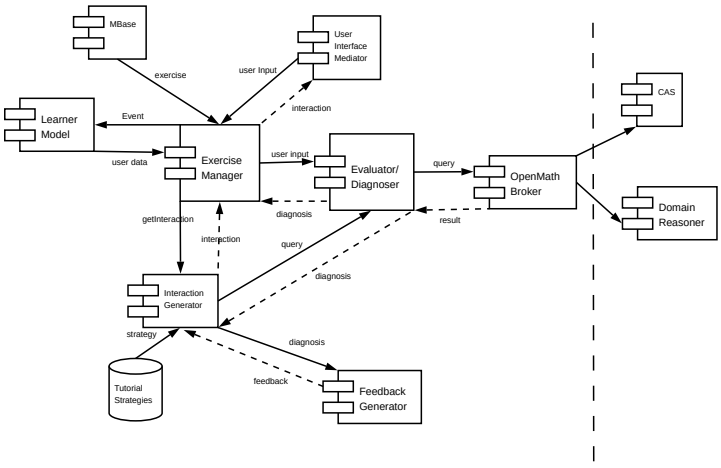

Figure 4. Exercise Subsystem architecture

an exercise from the database and the learner data from the `Learner Model`. In each step it connects to an `Interaction Manager` in order to receive the next `interaction` to be played.

In case of a fully authored interaction, the `Interaction Manager` is substituted by the default static `Interaction Manager` that is delivering the existing manually authored `interaction` from the exercise. After receiving a fully authored `interaction`, `Exercise Manager` sends it to `User Interface Mediator` which presents it to the learner.

In case of a (partially) generated interaction, another generator is employed. This generator might need to connect to the `Evaluator/Diagnoser` which can use `Domain Reasoner` or `CAS` components to provide the diagnosis on the action of the learner. The `Evaluator/Diagnoser` can query external services such as `Domain Reasoner` or `CAS` for diagnosis on the input. Using this diagnosis the feedback and the next interaction are generated.

After the learner has submitted his answers or requests [3], the `User Interface Mediator` is translating this answer/requests into OpenMath and passes it to the `Exercise Manager`. The `Exercise Manager` connects to the `Evaluator/Diagnoser` component which evaluates the learner's input by comparing it to each of the conditions in the `transition map` of the current `interaction`.

The generator also applies a tutorial strategy from the `Tutorial Strategy` module to the step.

As soon as an `interaction` has been assembled, it is returned to the `Exercise Manager` which is passing it to the `User Interface Mediator`. In each step, the `Exercise Manager` is also sending an event containing the learner's input and information characterizing the step such as the competencies trained in the step,

[3] Requests allow asking for hints or other forms of help

achievement of the learner w.r.t. the task, identifiers of concepts and misconceptions in erroneous steps. This event can be used for updating the `Learner Model`.

The current implementation of the exercise subsystem is server-based and works with a tight integration of the components by means of procedure calls. The components of the exercise subsystem exchange fragments of XML documents (interactions or their parts) or only OpenMath formulas (e.g. for learner input).

6. Conclusion

ActiveMath offers a general system-independent semantic markup language for representing interactive exercises of different kinds. The exercise subsystem of ActiveMath is playing such exercises in a browser. The exercise subsystem can connect to CASs and to other external programs in order to diagnose the learner input and to generate feedback. There are no explicit calls to a particular external program in the exercise content.

An exercise can be manually authored, generated completely, or partially generated/authored.

In order to generate feedback and next interactions, the diagnosis of the learner's input has to be made. For this, the connection to a reasoning component is established.

Obviously, a bottleneck is to find and encode all relevant potential erroneous actions of the learner. For instance, in [5] 1000 buggy rules for the fractions domain were empirically found and narrowed down statistically to 300.

User-adaptive presentation can be achieved via automatic enrichment of the manually authored exercise by the tutorial strategies which are called and applied to the exercise representation. Therefore, it is possible to reuse the same tutorial strategies for different exercises.

The graphical user interface for presenting such an exercise to the learner is also interchangeable since the exercise representation does not determine the presentation/rendering. The presentation is handled separately by the ActiveMath presentation system [9].

Acknowledgment

We are indebted to Claus Zinn for his clarifying interventions and help.

This publication is partly a result of work in the context of the LeActiveMath project, funded under the 6th Framework Program of the European Community – (Contract IST-2003-507826) and RTD project iClass, funded under the 6th Framework Program of the European Community – (Contract IST-507922). The authors are solely responsible for the content.

References

[1] J. Büdenbender, E. Andres, A. Frischauf, G. Goguadze, P. Libbrecht, E. Melis, and C. Ullrich, Using Computer Algebra Systems as Cognitive Tools. *6th Inter-*

national Conference on Intelligent Tutor Systems (ITS-2002), Springer-Verlag, Series "Lecture Notes in Computer Science", editors S.A. Cerri, G. Gouarderes, and F. Paraguacu, number 2363, pages 802–810, 2002.

[2] Global Learning Consortium. IMS Question & Test Interoperability Specification: A Review <http://www.imsglobal.org/question/whitepaper.pdf>

[3] G. Goguadze, E. Melis, C. Ullrich and P. Cairns, Problems and Solutions for Markup for Mathematical Examples and Exercises. In Proceedings of the Second International Conference on Mathematical Knowledge Management, MKM03, Andrea Asperti (ed.). <http://www.ags.uni-sb.de/~ilo/articles/EncodingExoExa.pdf>

[4] Goguadze, G., González Palomo, A., "Knowledge Representation of Interactive Exercises in ActiveMath", SEKI-Report, Nr. SR-04-10, University of Saarland, 2004.

[5] M. Hennecke. *Online Diagnose in intelligenten mathematischen Lehr-Lern-Systemen.* PhD thesis, Fortschritt-Berichte VDI, Hildesheim, 1999.

[6] M. Kohlhase. OMDoc: Towards an OpenMath representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000.

[7] E. Melis, E. Andrès, J. Büdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. ActiveMath: A generic and adaptive web-based learning environment. *International Journal of Artificial Intelligence in Education*, 12(4):385–407, 2001.

[8] M. Mavrikis, A.G. Palomo, Mathematical, Interactive Exercise Generation from Static Documents. <http://www.maths.ed.ac.uk/wallis/format/papers/mm_agp_mkm.ps>

[9] P. Libbrecht, C. Ullrich, and S. Winterstein, An Efficient Presentation-Architecture for Personalized Content, in Proceedings of Berliner XML Tage 2003, editors R. Tolksdorf and R. Eckstein, Pages 379–388, 2003, ISBN 3885791161

[10] R. Schulmeister. Didaktisches Design aus hochschuldidaktischer Sicht - Ein Plädoyer für offene Lernsituationen. In U. Rinn and D.M Meister, editors, *Didaktik und Neue Medien. Konzepte und Anwendungen in der Hochschule*, number 21, pages 19–49. ., 2004.