

Interactivity of Exercises in ACTIVEMATH Full paper

Giorgi Goguadze ^a, Alberto González Palomo ^b, Erica Melis ^c,

^a *University of Saarland, Saarbruecken, Germany*

66123 Saarbrücken, Germany

Tel: +49681 302 64822 george@activemath.org

^b *German Research Institute for Artificial Intelligence (DFKI)*

66123 Saarbrücken, Germany

alberto@activemath.org

^c *German Research Institute for Artificial Intelligence (DFKI)*

66123 Saarbruecken, Germany

Tel: +49 681 302 4629, melis@dfki.de

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, interesting and challenging learning experience, a variety of types of interactive exercises is needed rather than multiple choice questions (MCQs) only. It has been shown, however, that feedback is mandatory for fully exploiting the benefits of interactivity for learning [8].

Many e-learning systems do not provide feedback that goes beyond a correct/incorrect response referring to pre-defined choices in MCQs or menus to choose from. Usually, more helpful, elaborate feedback is authored or programmed for each exercise (class of exercises) and a general diagnosis of the learner's actions is missing. Even in intelligent tutoring systems (ITS), this diagnosis is done for small domains only and feedback is mostly authored and represented together with other types of information in the exercises.

Such a program is powerful if it uses a computer algebra system (CAS) to evaluate the input of the user. An author could just encode the exercise using the language of the computer algebra system he is using and connect to the CAS to "play" the exercise.

Previously, exercises in ACTIVEMATH were programmed individually or as classes of exercises. Our experience is that such an approach is quite limited. On one hand, it binds authors to the syntax of the computer algebra system and the authors are mostly struggling with the CAS language trying to program their pedagogical strategies into the syntax. Moreover, CASs are not designed for representing such tutoring strategies and human-line 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, adapting to the particular learner or learning situation.

In case a CAS interface is used as the front end not all types of interactivity are supported. For example, depending on the CAS, it could be impossible to present multiple choice questions to the learner, or puzzle exercises where there learner has to move graphical objects on the screen.

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. The latter responds to the needs of modifiability, extensibility and interoperability.

This paper is organized as follows. After short preliminaries about ACTIVE-MATH system and 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 architecture. Finally, we summarize how the problems mentioned above are solved in ACTIVEMATH.

2. Preliminaries

ACTIVEMATH is a web-based user-adaptive learning environment for mathematics [6]. It dynamically assembles courses of learning material for the individual

learner according to her learning goals, preferences, and mastery of concepts as well as to the competencies to be achieved. It makes learning suggestions, presents interactive exercises and scaffolds the exercising.

The information about the learner and his learning situation is represented in a Learner Model. The dynamic assembling of courses is performed by the Tutorial Component which selects materials to be presented to the learner, according to the information from Learner Model. The content is annotated with metadata and stored in a knowledge base.

OPENMATH and OMDOC Representations for Mathematics The knowledge representation language in ACTIVEMATH is OMDOC, a semantic markup language for mathematical documents [5]. OMDOC has evolved as an extension of OPENMATH¹ 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. 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 (via so-called phrasebooks). This provides the basis for interoperability in different computer algebra and other reasoning systems.

OMDOC defines learning objects (LOs), such as exercises, definitions, and 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 described in the following section.

3. Knowledge Representation of Interactive Exercises

An exercise consists of a finite state machine of interactions. An interaction contains feedback for the event that triggered it (either the start of the exercise or an answer from the user), 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 maps” describing which interaction comes next depending on the user’s input. Different kinds of exercises arise from different modes of input, and consequently the input mode can be replaced without altering the semantics of the exercise.

We develop a representation that matches the essence of the interaction. This makes the authoring without authoring tools or other support more difficult.

¹see <http://www.openmath.org> for the standard specification

3.1. Representation of the Interaction

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

An **interaction** consists of one or more **feedback** elements, followed by **interactivity assignment** and **transition map** elements, and can contain nested **interaction** elements at any point.

As an alternative to a concrete interaction, a reference (xref) can represent the interaction element.

The element **feedback** contains general textual content presented to the user in the step². 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 it is removed.

The **interactivity assignment** assigns 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, marking.

The **transition map** contains the conditions and next interaction pointer given a condition is satisfied. It consists of one or more **condition** elements and a **default** for a default condition.

Each interaction can be annotated with metadata, describing competencies, relations to concepts trained in the current step, difficulty, mastery assessment to be sent to the learner model.

In order to evaluate the **condition** element we define three predicates: **syn_eq**, **num_eq** and **sem_eq**. **syn_eq** means 'syntactic equality' and calls a procedure that checks whether the input of the user is syntactically equal to a given expression. **num_eq** means 'numeric equality' and calls a procedure that checks that the input of the user should be a number and is numerically equal to a given number. If input differs from prescribed result less than the 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 user is the same mathematical object. For instance, the user's input can be sent to a computer algebra system, normalized by the CAS and compared to a normalized pre-defined result.

An exercise can include fully authored **interaction** elements describing steps in the exercise and an **interaction_generator** elements that is used to instantiate the Interaction Manager as described in section 4.

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 on whether a component exists for generating feedback from the diagnosis.

²This is analogous to QTI's **material** element.

Bottleneck is to find and encode all relevant potential erroneous learner actions. In [4] 1000 buggy rules for the fractions domain were empirically found and narrowed down statistically to 300. Both, offline and online generation of such erroneous rules is possible.

Consider the two versions of a small exercise in Figure 1 and 2, whose interactions were completely generated by a domain reasoner component and whose feedback was inserted automatically by two different tutorial strategies. Figure 1 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 the simpler task by pre-inserting the structure of the derivation rule to be applied here. In the second strategy, shown in the Figure 2, the needed derivation rule is formulated, and the learner is invited to try the same task again.

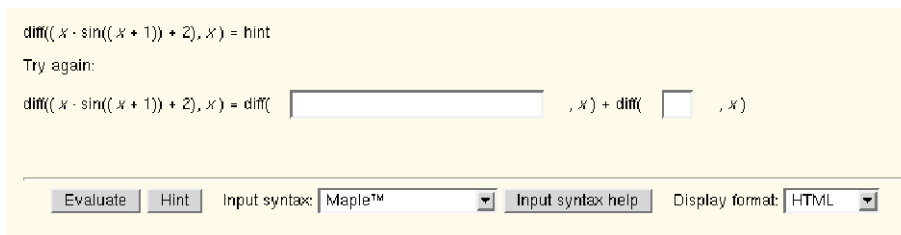


Figure 1. Partial insert - hint strategy

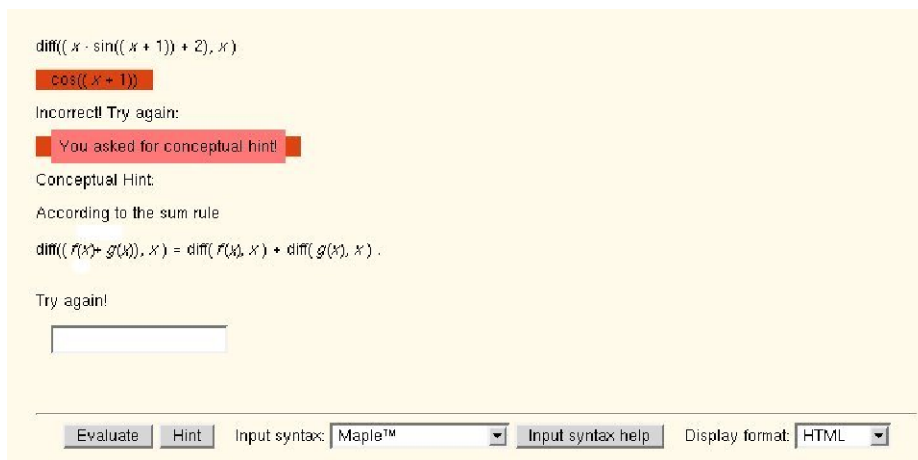


Figure 2. conceptual hint strategy inserting the rule statement

The hint in the first strategy is generated via the domain 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 domain reasoner each time. The domain reasoner

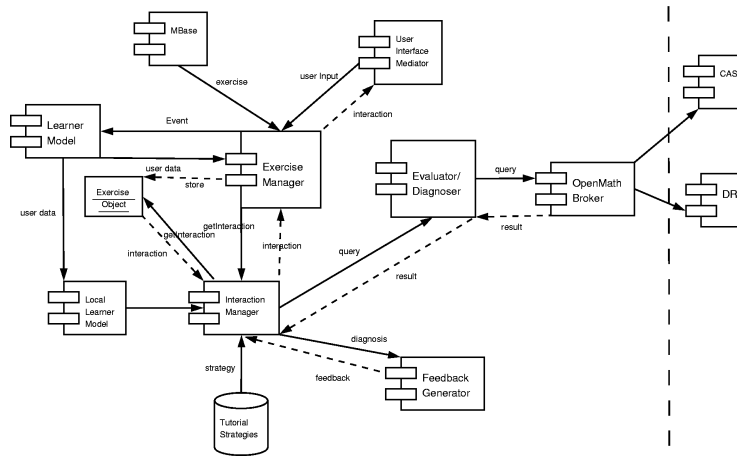


Figure 3. Exercise Subsystem architecture

only determines the rule and provides the link to it in the knowledge base to the tutorial strategy.

4. Separation of Functionalities for Scaffolded Exercising

Among the functionalities of the exercise subsystem are rendering interactive steps, evaluation of the user's input, diagnosis and feedback generation, automatic exercise generation, application of tutorial strategies for adapting to the learner and his learning situation, reporting the achievements of the learner to the rest of the ACTIVEMATH system.

It is a well-known fact that modifications and extensions as well as adaptation 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 4 displays a simplified architecture and information flow.

We briefly describe the components shown in the figure.

User Interface Mediator is translating the user's input (answers or requests ³) into OPENMATH and transforms the feedback received from the system to the presentation format needed.

Exercise Manager is the central component of the exercise subsystem. It receives the exercise from the database and runs the exercise. It passes data from one component to the other that has the required functionality. After receiving the user input from User Interface Mediator it sends it to the evaluator together

³Requests allow asking for hints or other forms of help

with the content of the **transition map** mapping the user input to potential next interactions, and receives the ID of the next interaction to be presented to the user. Then the Exercise Manager asks the generator for the next interaction element and replaces the ID by it.

At each step the Exercise Manager is sending events including the user's input and the metadata for the step to the learner model to keep it updated w.r.t. the changes in learners competencies.

Interaction Manager produces the next interaction to be delivered to the Exercise Manager. In the manually authored exercises, where the next interaction already exists, the default static generator is called. It is traversing the exercise graph and looking for the interaction with a given ID. For dynamically generated interactions, the generator creates the next interaction on demand. The name of the generator and its parameters are specified in the exercise content. For manually authored exercises, the additional generators might be employed that enrich the original exercise. This is the case, e.g. when a tutorial strategy is applied to it. The Interaction Manager can be supported by the learner model in order to perform the tutorial strategy according to user's action record, stored there. For example, the local user model counts, how many times the same step was executed and the tutorial strategy decides what to do in each case. The local learner model also receives information from the global learner model which is passed to the Interaction Manager in order to adapt the tutorial strategy to the (changes of) the learner's competencies. The Interaction Manager can also connect to external components, e.g. domain reasoner in order to generate feedback to the user's input.

Evaluator Chooses the adequate next step to the user's input among the alternatives provided by the transition map of the current step. It calls Domain OPENMATH broker to evaluate mathematical expressions of that input in CAS or a Domain Reasoner.

The Exercise Manager relies completely on the evaluator to provide the identifier for the next interaction, without checking whether it was among the ones provided in the pre-defined or authored transition map. The evaluator can choose to override the transition map and redirect the interaction to any other, such as a generic interaction element from an interaction library.

The evaluator could use a variety of parameters to decide which interaction is the next one. The most obvious is the user input expression as provided by the Exercise Manager. Another one could be the timestamp of the user's answers, because quick wrong solutions may indicate that the user misunderstood some concept or is bored and does not care, quick correct ones may indicate that the exercise is too easy so that larger steps can be expected from that user. Slow wrong ones may indicate that the user is disoriented and would need help to keep him motivated.

OPENMATH Broker Receives expressions to be evaluated by some OPENMATH service, which is chosen among a configurable list based on the symbols used in the expression. The current implementation calls the YACAS Computer Algebra

System⁴ as it handles both input and output in OPENMATH. We are considering to extend it to a more flexible architecture like the Logic Broker Architecture [1].

The current implementation of the exercise subsystem is server-based and works with tight integration of the components by means of procedure calls. During the communication components of the exercise subsystem exchange fragments of XML documents (interactions and their parts) and in some cases only OPENMATH formulas (user input).

5. Conclusion

ACTIVEMATH offers a general system-independent semantic markup language for representing interactive exercises of different kinds. The exercise subsystem of ACTIVEMATH is running such an exercise in a browser. The exercise subsystem can connect to different CAS and to other external programs in order to diagnose the user input and generate the feedback. The connection to the particular CAS is provided by a separate "broker" component that automatically decides which services are available for the current need of the exercise. This means, there are no explicit links to the external programs in the exercise content.

Such an exercise can be manually authored, or generated completely or partially by an external component.

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

The graphical 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.

Acknowledgment

Also 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 LEACTIVE-MATH project, funded under the 6th Framework Program of the European Community – (Contract IST-2003-507826). The author is solely responsible for its content.

References

- [1] C. Giromini. Logic Broker Architecture: Interconnessione di Sistemi Distribuiti per il Ragionamento Automatico. <http://www.ags.uni-sb.de/~corrado/papers/lba_master_thesis.ps.tgz>
- [2] Global Learning Consortium. IMS Question & Test Interoperability Specification: A Review <<http://www.imsglobal.org/question/whitepaper.pdf>>

⁴see <http://yacas.sourceforge.net> for description

- [3] G. Gogvadze, 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] M. Hennecke. *Online Diagnose in intelligenten mathematischen Lehr-Lern-Systemen*. PhD thesis, Fortschritt-Berichte VDI, Hildesheim, 1999.
- [5] M. Kohlhasse. OMDOC: Towards an OPENMATH representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000.
- [6] E. Melis, E. Andrès, J. Büdenbender, A. Frischauf, G. Gogvadze, 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.
- [7] 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>
- [8] 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.
- [9] C. Zinn, J.D. Moore, M.G. Core, S. Varges, and K. Porayska-Pomsta, The BE&E Tutorial Learning Environment (BEETLE), System Demonstration, Proceedings of the Seventh Workshop on the Semantics and Pragmatics of Dialogue (DiaBruck 2003), Saarbrücken, Germany, September 2003.