# Authoring Web Content in ActiveMath: From Developer Tools and Further

Paul Libbrecht `paul@activemath.org` [*]

DFKI and Universität des Saarlandes, Saarbrücken, Germany

**Abstract.** In the ActiveMath learning environment, authoring is currently based on source editing following a *build*-practice similar to software development cycles.
In this article we explain the characteristics of a build-cycle that are important to the creation of content for an adaptive educational hypermedia such as ActiveMath. The trust of an author into the behaviour of the system is one of the important goals that is aimed at. We then describe what, we expect, would further help the author's trust, namely, usable tools to input and play simulations of the learning process.

## 1 ActiveMath as an adaptive educational hypermedia

ActiveMath is an intelligent web-based learning environment. It presents content and lets learners perform interactive exercises.

Content, in ActiveMath is organized in knowledge units called *items* which roughly correspond to paragraphs in a text-book. The content of items is made of text and mathematical formulæ encoded semantically using OpenMath [2]. Items are given a type (for example, a *definition* or *exercise*), and are complemented with metadata, the latter is made of slot-values (e.g., the *difficulty* or *learning-context*) as well as relations (e.g., the statement that a proof *proves* a given assertion, or for a concept to *depends-on* another concept). A subset of items, named *concepts*, along with their metadata make the *domain-model*. Please see [6] for a more detailed coverage of ActiveMath knowledge handling.

The adaptivity in ActiveMath is based on an overlay user-model based on this domain-model. The user-model is permanently updated using information such as reading of items and exercise results. Based on this user-model the following adaptivity features are provided in ActiveMath:

– Using preferences, the presentation is given an appearance or *theme*
– In ActiveMath the learner mostly accesses content through the familiar metaphor of books which are presented as a sequence of pages with a table-of-contents. The latter is presented with visual hints (little red, yellow, or green bullets) indicating which page refers to concepts with low, medium, or high user-model values. This can be seen as an elementary link-annotation adaptivity.

[*] This work has been partially supported by projects funded by the German Ministry for Education and Research (BMBF)

– Books can be dynamically generated in ActiveMath: they are generated according to the goals of a learner and to a pedagogical scenario. Course generation makes use of the items types, slot metadata, and relations as well as the user-model values so as to select the concepts that should be read as well as the supporting texts and exercises. This course generation can be viewed as a form of adaptive navigation support.
  While staying with the metaphor of a book, this course generation will be enhanced into a reactive component where the learner will be presented, in the usage of the book, with modifications to her book considered to be appropriate for her learning (see [7]).
– Interactive exercises are currently mostly non-adaptive in ActiveMath: they are, for example, multiple-choice-questions. Exercises, within the LeActiveMath European project are planned to become adaptive using the same source of information as the course generation: it should make it possible for an interactive exercise to propose actions and provide feedback depending on the knowledge (and understanding or applications capabilities) of the learner for the given concept.

## 2 Authoring in ActiveMath: Tools Available

This section reviews briefly the authoring tools available in order to write content for ActiveMath.

### 2.1 Writing by Hand

Content in ActiveMath is encoded as XML documents in the OMDoc syntax [4]. As such, it can, in principle, be written by hand. This was done by some developers in our group while building the system. Experience has shown that doing so was a very confusing task especially with respect to OpenMath formulæ which are encoded in very deep sub-trees.

An alternative was to write QMath documents, offering a compact syntax for both formulæ and content structure. The QMath processor then converts them to OMDoc. Results were disappointing as QMath was not complete enough for the needs of ActiveMath's OMDocs. The treatment of mathematical formulæ in QMath was however kept: a wrapper was written, called OQMath, letting QMath process the formulæ. This allows the rest of the document be a valid XML document.

Experience has proven that the encoding for OQMath documents offers a reasonable readability while still allowing support of XML-based editing tools. An editor was chosen as reference tool to edit documents, the open-source editor jEdit. It offers, among others, templates, suggestion of possible children at insertion point and automated XML-validation (with a feedback similar to spell-checking).

The package is distributed under the name jEditOQMath and comes along with build-scripts to let authors reload content easily into a running ActiveMath, thus allowing them to cycle between content-preview and content-editing. The tool has been used successfully by non-developers, even without prior knowledge of HTML or XML.

## 2.2 On the Importance of Validation and Error-Reporting

As we see, the developer-tools approach is still quite present. But developer-tools are not only present by the fact that a source is being edited, but also by the fact that a *build-script* is being used along with error-reporting presented in the source. We describe here the validation tasks and error-reporting performed in the current jEditOQMath and highlight why they are important:

- XML-validation: in jEditOQMath, it happens simply at each save and makes sure the content elements are appropriately nested. The lack of such validation has been observed several times to create awkward presentation problems which are hard to detect.
- reference checking: reference between items are both inside the items and in the metadata relations. For a long time, tools to report these errors were not present and the database storing the content was very tolerant. Reference errors were frequent and had such misbehaviours as the lack of content in course-generation, the crash of the latter, or wrong user-model updating.

Actually, the needs for error-reporting are independent of the fact that a source is being edited. A visual tool should, as well, provide such feedback, including an error-list taking the user to the place where the error is to be corrected.

## 2.3 Verifying the Content

The need to verify (or *test*) an installed content can also be deduced from authors observed thus far: they spend almost more time to proof-read the content presentation than actually writing content. They do so as a *demo-user* (sometimes several) using their local web-browser.

One reason for this fact is certainly that the current presentation is from an XML source with extensible presentation of mathematical symbols for which it is easy to loose the overview. Moreover, this presentation provides access to many possible interactions (e.g. the navigation along a path of relations in the dictionary). It has been observed that such aspects of the presentations are, indeed, manually checked by authors.

All the adaptive behaviours of ActiveMath are also verified manually: the appearance under different *themes*, the results of course-generation, and the user-model updates. Such tests are, however, very long to perform: currently, course-generation takes several minutes and relies on a user-model state which the author should be clear with. Moreover, verifying the user-model update can only be done following a navigation path which takes a long sequence of clicks and may need much resources to be computed.

As a result, such verifications, although wished, are very rarely performed.

## 3 Scope of the Authoring Task

Before going further with the verifications task, allow us to try to answer the following question: *how much information should be affected by an authoring tool?*

The candid answer to this question would be that authoring tools should edit the *content*. Authors tend, however, to wish more than just providing content items, along with their metadata. Here are a few examples of possible modifications of the default ActiveMath behaviour that authors have wished thus far:

– the presentation of ActiveMath functions may be changed in order to make the system more accessible to target users (for example removing some links-generation for school pupils)
– adapting the pedagogical rules used in the course-generation scenarios to particular usages or particular content
– performing elementary changes in the appearance such as the introduction of the institution's logo or the change of some colour choices
– the base user-model that new registrants will be endowed with should be adapted to the expected newcomers (or their groups)

In order to be able to provide the freedom to perform and distribute such information, the scope of authoring should encompass the whole data of ActiveMath, including configuration, rules, and menu-templates. A notion of *project* should thus to be defined, extending the notion of content-collection. We shall try to satisfy these requirements: projects will be defined which will have *deployment* routines to install on a fresh ActiveMath. As much as possible, these projects will be encoded using ontology-based tools so as to allow a declarative knowledge representation of such modifications.

Offering a freedom as important as impacting the whole ActiveMath is, however, dangerous: it is easy to break a system by changing its configuration or changing a set of rules and a presentation can be made unusable because of wrong colour choices. The need to verify the *installation* of a project is thus made even more important. In the next section, we sketch how such verification could be helped by simulations which we propose to implement.

## 4 Simulations to Check the Learning Path

In [3], Hayashi, Ikeda, Seta, Kakusho, and Mizoguchi present an approach to author learning content using ontological engineering. They propose to model *conceptual simulations* which are high-level specifications of the expected behaviour of the content-playing-in-the-software. This approach seems to be the right path to take in the long term, where the systems' behaviour is sufficiently transparent, it seems not to provide an answer to nowadays verification wishes on existing adaptive systems and requires a very abstract representation for the simulations to be entered.

We claim that such a simulation can be made much more visual and concrete, in fact, close to the current practice of authors checking the content with his browser but providing capability to input quickly such a simulation (including the time spent to read items), store it and replay it, with a summarized view (e.g. thumbnail) and, most probably, with automated tests on such values as the learner-model entries or the existence of a link.Making these simulations of the size of a thumbnail allows an author to have several simulations under the eyes, thereby being to envision several target users. Being able to replay these simulations often with the content evolving allows the author to use these as a form of integrated tests: a quick view on the sequence of screens obtained in such a simulation provides a glimpse at the expected views the target user will experience, the measure of success of the tests is a measure of achievement of the content, a practice similar to the practice of unit-testing in software development.

Such simulations are probably of more general applicability. In particular, they apply for both content that is re-used and content that is freshly authored.

## 5 Conclusion

We have been describing the authoring tools realized thus far for the ActiveMath learning environment. The experience gained has proven the importance of the validation tools and verification possibilities so that authors' trust in the presentation and adaptive features can be gained.

In order to raise the trust we propose to provide to authors efficient and usable tools to input and play simulations of the learning process using visual approaches but still allowing computable verifications.

It should be observed that the visual simulations we are proposing could be interpreted as an application of the WYSIWYG paradigm (*What You See Is What You Get*). This is misleading, however, as the simulations are intended to provide multiple views on the content whereas a WYSIWYG approach is based on a single view.

### 5.1 Related Work

The problem of putting the control of intelligent *agents*, such as adaptive systems, at work under the hands of a user is not new, see for example [5]. However, we have found little done towards offering control of an adaptive systems to a user. These simulations would like to provide ways to experiment with an adaptive systems more efficiently and in a comprehensive fashion.

Our investigation in the literature for finding related work has not been fruitful. The high-level simulation formulated in [3] seems to be the only such approach and is fundamentally different from ours. We anticipate, however, that the needs for such a concrete simulations will grow as the serving world evolves in adaptivity functions.

### 5.2 Future Work

The development of the source-editing facilities that we have presented will be stabilized and the authoring tools in ActiveMath will migrate to edition within visual tools. Following the *view to the future* of [1], we are currently evaluating considering graph-based input of the domain-model along with the items' content using the Protégé ontology editor. The wealth of the verification tools will be kept and be complemented with visual simulations and their associated tests.

### 5.3 Acknowledgements

The author would like to thank the members of the ActiveMath group under the direction of Erica Melis for the feedback provided and the patience for their realization. He would like to thank the authors of ActiveMath content having taken the time to dive into the proposed tools even with little help or documentation.

## References

1. Peter Brusilovsky. Developing adaptive educational hypermedia systems: From design models to authoring tools. In Tom Murray, Stephen Blessing, and Sharon Ainsworth, editors, *Authoring Tools for Advanced Technology Learning Environment.* Kluwer Academic Publishers, Dordrecht, 2003. See `http://www2.sis.pitt.edu/~peterb/papers/KluwerAuthBook.pdf`.
2. O. Caprotti and A. M. Cohen. Draft of the open math standard. Open Math Consortium, `http://www.nag.co.uk/projects/OpenMath/omstd/`, 1998.
3. Yusuke Hayashi, Mitsuru Ikeda, Kazuhisa Seta, Osamu Kakusho, and Riichiro Mizoguchi. Is what you write what you get?: An operational model of training scenario. In Gilles Gauthier, Claude Frasson, and Kurt VanLehn, editors, *Proceedings of the Fifth International Conference on Intelligent Tutoring Systems (ITS 2000)*, volume 1839 of *LNCS*, pages 192–201. Springer, June 19–23 2000. See also `http://www.ei.sanken.osaka-u.ac.jp/pub/hayashi/hayashi-ITS2000.pdf`.
4. Michael Kohlhase. OMDoc: Towards an internet standard for mathematical knowledge. In Eugenio Roanes Lozano, editor, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2000*, LNAI. Springer Verlag, 2001. See also `http://www.mathweb.org/omdoc`.
5. Pattie Maes. Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40, 1994.
6. E. Melis, J. Büdenbender E. Andrès, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Knowledge representation and management in ActiveMath. *Annals of Mathematics and Artificial Intelligence, Special Issue on Management of Mathematical Knowledge*, 38(1-3):47 – 64, 2003. Volume is accessible from `http://monet.nag.co.uk/mkm/amai/index.html`.
7. C. Ullrich. An instructional component for dynamic course generation and delivery. In R. Tolksdorf and R. Eckstein, editors, *Proceedings of Berliner XML Tage 2003*, pages 467–473, 2003.