Integrating Web Services into Active Mathematical Documents

Jana Giceva, Christoph Lange, and Florian Rabe

Computer Science, Jacobs University Bremen,
{j.giceva,ch.lange,f.rabe}@jacobs-university.de

Abstract. Active mathematical documents are distinguished from traditional paper-oriented ones by their ability to interactively adapt to a reader's inputs. This includes changes in the presentation of the content of the document as well as changes of that content itself.
We have developed the JOBAD architecture, a client/server infrastructure for active mathematical documents. A server-side module generates active documents, which a client-side JavaScript library makes accessible for user interaction. Further server-side modules – in the same backend, or distributed web services – dynamically respond to callbacks invoked when the user interacts with the client. These three components are tied together by the JOBAD active document format, which backwards-compatibly enhances MathML by information about interactivity.
JOBAD is designed to be modular in the specific web services offered. As examples, we present folding and elision in mathematical expressions, type and definition lookup of symbols, as well as conversion of physical units. We evaluate our framework with a case study where a large collection of lecture notes is served as an active document.

1 Introduction and Related Work

Documents are an important interface for distributing mathematical knowledge. Recently, the technological development has shifted attention more and more towards digital documents and the added-value they can offer. This has led to a number of research efforts on interactive mathematical documents involving features such as adapting mathematical documents, interactive exercises, and connecting to mathematical web services.

The ActiveMath project investigates how to *aggregate documents* from a knowledge base such that the resulting document contains exactly the topics that the reader wants to learn and their prerequisites [Act08]. *Interactive Exercises* have been realized in ActiveMath and MathDox [GM08,CCK+08,CCJS05]. Here, the user enters the result into a form and then gets feedback from a solution checker in the server backend. ActiveMath comes with its own web services [MGH+06], and MathDox has originally been designed for talking to computer algebra systems but can also connect to other services via MONET (see below). Gerdes et al. have developed a reusable exercise feedback service for

exercises that has also been integrated with MathDox [GHJS08]. Besides supporting MathDox's own communication protocol, Gerdes's service also complies to the XML-RPC and JSON data exchange standards [GHJS08]. The services developed within the MathDox and ActiveMath projects, such as the ActiveMath course generator, are potentially open to any client, but have not been used with any frontend other than their primary one so far [Ull08,MGH+06].

There are also elaborate *web service architectures* for mathematical web services that have been designed for integration with many systems, such as the ones developed in the SCIEnce [SCI09] and MONET projects [Mon05]. SCIEnce explicitly targets symbolic computation and grid computing and does not consider documents as user interfaces. MONET is an architecture that, in principle, allows for any kind of mathematical web service. Still, mainly computational web services have been developed and evaluated within that framework. Web services can register with a central MONET broker that accepts requests, which do not directly call a web service but consist of a problem description (e. g., solve an equation, given as an OpenMath expression). The broker then forwards the request to the best-matching service. The above-mentioned MathDox allows for access to MONET web services via a document interface.

Asynchronous communication with a server backend (AJAX) allows for client/-server interaction without submitting forms. It is a prerequisite for responsive browser-based applications: A client-side script can exchange small data packets with a server backend and insert responses from the server into the current page. This technique is employed by MathDox [CCK+08] and Gerdes's frontend to their feedback service [GHJS08].

Despite the efforts mentioned above, there is still a lot of static mathematical content on the web. Where documents act as frontends to web services, as in the above-mentioned systems, they have usually been designed to give access to a small selection of web services performing very specific tasks (mostly giving feedback to exercises and symbolic computation) – as is the case with ActiveMath and MathDox.

Our goal is to facilitate the integration of diverse web services into mathematical documents – inspired by the Web 2.0 technology of *mashups* [O'R05,AKTV08]. Originally, mashups were handcrafted JavaScripts pulling together web services from different sites. Since then several mashup development kits have been developed, e.g., Yahoo! Pipes [Yah09] or Ubiquity [Moz09]. We aim at a similar development kit for mathematical applications. Our vision of an *interactive document* is a document that the user can not just read, but adapt according to his preferences and interests *while* reading it – not only by customizing the display of the rendered document in the browser, but also by changing notations (which requires re-rendering) or retrieving additional information from services on the web. Consider a student reading lecture notes: Whenever he is not familiar with a mathematical symbol occurring in some formula, he should be able to look up its definition without opening another document, but right in his current reading context. Or consider the problem of converting between physical units (e. g., imperial vs. SI). There are lots of unit converters on the web (see [Str08]

for a survey), but instead of manually opening one and copying numbers into its entry form, we want to enable an in-place conversion.

In [KMR08,KMM07], we investigated how OMDoc documents containing content markup can be rendered as XHTML with embedded presentation MathML. The rendering was relative to context dimensions such as the native language of the reader or the field of knowledge. This approach used *notation definitions* to translate content markup into presentation markup. It focused on generating documents *before* the user gets to read them. In the present paper, we continue this line of research and introduce JOBAD as the client-side counterpart. JOBAD is a JavaScript API for OMDoc-based Active Documents. While its primary intended application is to be part of the active documents served by our server, it is independent of the server and can be flexibly reused to enable any mathematical document to interact with the reader or web services. Our contribution includes the JOBAD interactive document format, an XHTML+MathML-based interface language between server- and client-side computation.

In Section 2, we present the main component of JOBAD, a collection of small JavaScript modules that add interactive services to a mathematical document. Here we assume a broad notion of "service" including local interactive functionality as well as any service with an HTTP interface, regardless of whether it complies with a "heavyweight" web service standard like XML-RPC or MONET. In Section 3, we present several web services that we have implemented and describe how to integrate third-party services. In Section 4, we briefly describe a first JOBAD case study, and we conclude in Section 5.

2 An Architecture for Active Documents

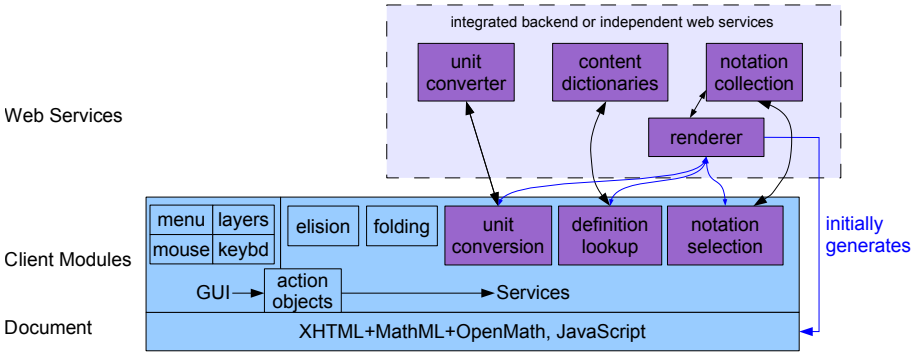

Fig. 1. JOBAD Architecture. Note the central role of the rendering service, which both generates JOBAD-compliant documents and is needed for many other services.

The JOBAD architecture is divided into the actual mathematical services, the user interface elements, and generic communication and document manipulation functions (see figure 1). On the client side, JOBAD consists of a JavaScript main module and one independent module for each service. The server controls the available functionality by loading a collection of service modules into the document. We distinguish three kinds of interactive services by the amount of data they exchange with the web service backend: 1. services that merely draw on data embedded into the document, 2. services that send a symbol identifier and an action verb to the backend in order to retrieve additional information, 3. and services that send complex mathematical content expressions to the backend.The decision which kind of service should be used for a particular functionality depends considerably on the format used for interactive documents: Some meta-information about the current appearance may be embedded into the document to permit local interaction, for example, alternative presentations or parallel content markup. If this is not feasible, further information must be embedded that instructs the JOBAD client how to retrieve the external document fragments.

2.1 A Document Format that Enables Services

The MathML specification leaves some details about the structure of a document to application developers [W3C]. Therefore, for JOBAD, we pose some additional requirements:

1. Alternative displays, among which the user can switch, SHOULD[1] be realized by `maction` elements and an `@actiontype` attribute that indicates the type of service [W3C, section 3.6.1]. Particularly, services that rewrite a formula SHOULD retain the previous state of the formula as an alternative to which the user can switch back.
2. Unless subterms are already part of a grouping operator (e. g. radicals, or super-/subscripts, or fractions; see [W3C, table 3.1.3.2] for a complete list), they MUST be grouped using the invisible `mrow` element (which is optional in MathML).
3. For services that need access to the semantics of mathematical expressions, the latter MUST be provided as parallel content markup [W3C, chapter 5.4]. There MUST be cross-links from the subterm-grouping elements of Item 2 to the corresponding content elements. Content elements MAY be annotated with additional attributions.
4. If services that directly operate on the presentation markup to customize its display require annotations that will be interpreted in a service-specific way, it may be considered impractical to add them to the parallel content markup and look them up there. In such a case, annotations MAY be added directly to presentation markup elements as attributes from another namespace (see [W3C, section 2.4]). It is RECOMMENDED that such annotations require considerably less additional space than parallel markup; otherwise parallel markup SHOULD be preferred.

[1] We use these capitalized keywords in accordance with RFC 2119 [Bra97].

Requirements 2 and 4 lead to a linear overhead in the size of the formulas (with a small factor), requirement 3 as well but with a factor of 2 to 4, as content markup, element IDs, and references to those IDs are added. Requirement 1 can lead to an exponential overhead when alternative displays are nested. This could be avoided if Presentation MathML allowed for structure sharing between expressions (which only Content MathML allows so far).

As these requirements are hard to satisfy by manual authoring, this markup is generated by our rendering service that generates presentation from content markup using pattern matching-based notation definitions for all symbols (cf. section 3.1). In the following, we will describe these requirements in more detail by discussing how they are used in a variety of services we implemented.

Switching between Alternative Displays An `maction` element can have multiple children, one of which is displayed at a time, controlled by the `@selection` integer attribute – whose value can be changed at runtime. The MathML specification suggests possible `@actiontype` values but does not prescribe a fixed semantics for them. We have introduced several values for that attribute and clearly specified the desired behavior of the services using them.

In particular, we use `maction`, with the action type `abbrev`, for author-defined abbreviations of complex expressions. Consider a physics document, where the author wants to provide $W_{\mathrm{pot}}(R)$ (potential energy) as an instructive abbreviation of the complex term $\frac{-e^2}{4\pi\epsilon_0 R/2}$. We introduce the OpenMath symbol `folding#abbrev` that serves as an attribution key. Our rendering algorithm uses the value of such an attribution as an alternative display.

For example,

```
<OMATTR>
  <OMATP>
    <OMS cd="folding" name="abbrev"/>
    W_pot(R)
  </OMATP>
  -e^2 / 4πε_0 R/2
</OMATTR>
```

is rendered as

```
<maction actiontype="abbrev" selection="1">
  -e^2 / 4πε_0 R/2
  W_pot(R)
</maction>
```

Other JOBAD services create `mactions` on the fly (using the name of the service as the value of the `@actiontype` attribute) whenever they rewrite a term t into t' (e. g. by converting units; see Sect. 3.3). This allows for making interactions *undoable*: The previous state t of a term is preserved in the second, hidden child of the `maction`, and the user can switch back to it. Not only do interactions become *un*doable locally, but they also become *re*doable: When

information from a remote web service has been used to rewrite $t \rightsquigarrow t'$, as is the case with unit conversion, e. g., and the user switches back to t, he can recover t' without causing the information to be retrieved from the web once more, as it is still cached in the `maction`.

Grouping Subterms for Interactive Folding Besides author-defined abbreviations, we have implemented interactive folding of *arbitrary* subterms, so that a reader can hide them if he feels distracted. Any subterm that is properly grouped (see requirement 2 above) is eligible for folding. When the user requests folding of a subexpression for the first time, we put both the original subterm and its folded version into an `maction` element with `actiontype folding` for making the action undoable (see above).

As an example, consider the expression $[1 + [2 \cdot x]]$, where square brackets denote `mrow`s. Suppose the user selects [part of] the subterm $2 \cdot x$ or right-clicks somewhere in that term and requests it to be folded. Then, the formula will display as $[1 + \dots]$. Clicking on the dots and selecting the "unfold" action from the user interface (e. g. the context menu) will restore the original appearance.

Cross-Linked Parallel Markup MathML provides the ability to attach semantic annotations to presentation markup as so-called *parallel markup* [W3C, chapter 5.4]. We use this to obtain the content counterpart of presentation expressions selected by the user. The direction of such cross-links is generally unspecified in MathML, but we fix it to "presentation→content" as that is the most natural direction for looking up a formal representation of the user's selection. Moreover, it is the only direction in which associative infix operators can be cross-linked, as multiple presentation operators have to link to one content symbol (cf. Fig. 2).

Therefore, we require links from symbols, numbers, identifiers, and subterm-grouping presentation elements to corresponding content elements, given by `@xref` attributes. If a content expression is to be looked up for a given selection of presentation elements, we locate the closest common ancestor of all selected XML elements that carries an `@xref` attribute and dereference it to obtain the corresponding content expression. An example is given in Figure 2.

Since our rendering algorithm supports pattern-matching-based and thus non-compositional translations from content to presentation markup, not every content subexpression corresponds to a presentation expression. For example, in the presentation element corresponding to $\sin^2 x$, there will be no subexpression pointing to the content expression $\sin x$.

Lightweight Annotations for Flexible Elisions The rendering algorithm that we introduced in [KMR08] enables a flexible elision of redundant brackets. We now describe how to utilize the output generated by that algorithm – and thus our rendering service, cf. Sect. 3.1 – for deferring the decision which brackets to elide until the document is read.

Fig. 2. Parallel markup: Presentation markup elements point to content markup elements. The light gray range is the user's selection, with the start and end node in bold face. We first look up their closest common ancestor that points to content markup, and then look up its corresponding content markup – here: *E.2*

When rendering a content expression $f(t_1, \ldots, t_m)$, brackets around the rendering of $t_i = g(s_1, \ldots, s_n)$ are redundant if the operator f binds weaker than g. Binding strength is determined by comparing the i-th input precedence of f with the output precedence of g. Redundant brackets are retained in the output document, but annotated with the difference between these precedences as the *elision level*. Besides brackets, our rendering service supports other *elision groups*, e. g., for type annotations, some of which are essential whereas others can be inferred. Then brackets are the special case of the elision group `fence`.

All visible Presentation MathML elements and all grouping elements can carry the attributes `@egroup` and `@elevel` from the OMDoc namespace. The value of the former can be any string with the value `fence` being reserved for brackets. The value of the latter can be any integer or the strings `infinity` and `-infinity`. In case an element is member of multiple elision groups, the `@egroup` and `@elevel` attributes can contain a space-separated list. For example, a left bracket, annotated with elision information, looks as follows:

```
<mo fence="true" omdoc:egroup="fence" omdoc:elevel="100">(</mo>
```

Our elision service allows the user to choose one *visibility threshold* for each elision group. If T_g is the threshold of elision group g, then all elements of group g whose elision level is above T_g are invisible. This is realized by using `maction` elements with action type `elision` that switch between an expression and an invisible `mspace` element. This also permits a document to provide initial visibility status for its elements.

2.2 User Interface

JOBAD offers various user interface elements for input and output. By right-clicking, a context menu can be requested for the object under the cursor or for the range of selected objects. A selection can be made in the usual way of dragging the mouse, or by repeated clicking on any part of a formula. In the latter case, the selection is extended step by step, always advancing to the parent grouping element. While performing actions on the current selection makes sense for services like folding or definition lookup (cf. Sect. 3.2), other services, such as elision, are also made available globally: If desired, bracket elision can be controlled document-wide by hotkeys from 0 (no redundant brackets displayed) to 9 (all redundant brackets displayed), and a collapsible toolbar placed next to each formula offers one slider per elision group for controlling the thresholds locally.

Calls to services are represented by generic action objects, which allows for providing diverse access to them. The same elision action can, e. g., be triggered via a local context menu, from a formula-local toolbar, and via a global keyboard shortcut.

Besides rewriting formulae, JOBAD offers tooltip-like popups for displaying information on demand. These can be annotations hidden in the document, but we mainly use it for displaying responses from web services, such as the definition of a symbol that the user wanted to look up (cf. Sect. 3.2).

3 Web Services and their Integration

The easiest way of realizing a mathematical web service is to expose functionality via an HTTP interface. When adopting the REST pattern [Fie00], URLs directly represent mathematical resources (e. g., OpenMath symbols). This can be used, for example, to retrieve the definition of a symbol. We call such services *symbol-based*. More complex services act on the selected expression. In those cases, we use parallel markup to obtain the corresponding content expression and include it in the body of the HTTP request. We call such services *expression-based*.

In the JOBAD framework, we do not commit to a fixed set of web services. Rather do we specify a way of how a JOBAD-compliant document server *advertises* available web services. For each service (client-side service or web service) the document server chooses to offer in the active documents it serves, it MUST serve the corresponding JOBAD JavaScript module to the client. In the head of an active document, each JavaScript module MUST be initialized. To modules that access web services, a description of a web service backend they can connect to MUST be provided. In the case of an integrated backend, these can be components of that backend, but it can also be remote web services that the document server is aware of.

The description of a symbol-based service MUST consist of a name that is displayed to the user and a URL that invokes the service. The URL MAY contain placeholders for cdbase, cd, and name of the symbol. Similarly, the description of an expression-based service MUST consist of a name and a URL.

Listing 1.1. Service initialization code in a document

```
<!-- utility functions (module loading, document manipulation,
     client/server communication) -->
<script src="../scripts/jobad.js"/>
<!-- our own initialization follows -->
<script type="text/javascript">
// GUI elements to be enabled
jobadInit("contextmenu"); // loads the context menu
// In-document services
jobadInit("elision");
// Web services
jobadInit("definition-lookup", "Look up definition",
  "http://jobad.mathweb.org/backend?action=definition-lookup
&cdbase=$cdbase&cd=$cd&name=$name");
</script>
```

3.1 Rendering

The rendering service is a prerequisite for making output from other services human-readable. In its simplest form, it accepts as input (in the body of an HTTP POST request) a fragment of OpenMath and returns the result of rendering it to JOBAD-enriched Presentation MathML. In the following, we will use render(c) for the result of rendering a content markup fragment c.

Our rendering service is implemented using the JOMDoc library, which implements the rendering algorithm described in [KLM+09,KMR08]. It has access to a collection of *notation definitions*, which map content markup patterns to presentation markup templates [KMR08].

3.2 Definition and Type Lookup

The definition lookup service sends the ID of a symbol σ to the server and expects as a response a content-markup formula containing a term that defines σ. The type of a symbol can be looked up analogously. Our implementation uses the RESTful URI format introduced at the beginning of this section. The information that we want to look up is encoded by the value of the *action* parameter, either `definition-lookup` or `type-lookup`. In the following, we will use def(σ) and type(σ) for the definition of a symbol, or the type, resp., as looked up by this service.

On the server side, the lookup is enabled by representing content dictionaries (CDs) in OMDoc [Koh06]. There, a symbol with type declaration and definition is represented as shown below, which allows for easy retrieval, e. g., using XPath. The situation in an OpenMath CD is similar, but as "definitional mathematical properties" (DefMPs) have not yet been specified, there is no way of identifying a definition of a symbol among its various mathematical properties.

```
<!-- Content markup omitted here to save space -->
<symbol name="sin">
```

```
  <type><OMOBJ>ℂ → ℂ</OMOBJ></type>
</symbol>
<definition for="#sin" type="simple">
  <OMOBJ>sin z = 1/2i (e^iz − e^−iz)</OMOBJ>
</definition>
```

Our current client-side implementation displays render(def(σ)) in a tooltip overlay at the cursor position. Alternatively, *definition expansion* is possible, which replaces the selected occurrence of a symbol with its definition and re-renders the formula. The original formula before definition expansion is kept as an `maction` alternative using the `actiontype definition-expansion`. In contrast to definition expansion, which can only be offered for simple ($\sigma :=$ expr) or pattern-based ($f(x) :=$ expr(x)) definitions, and for inductive definitions, if applied for a single step, definition lookup can even be offered for implicit definition. (See [Koh06, chapter 15.2] for definition types supported by OMDoc.)

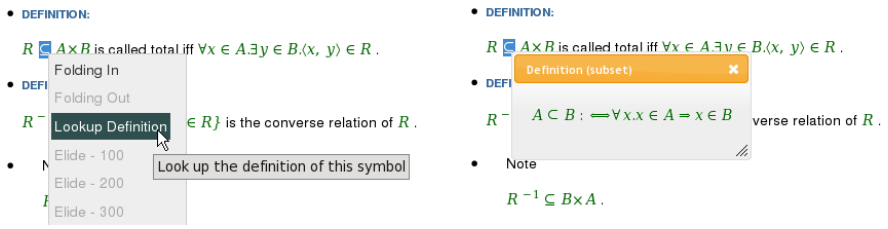

Fig. 3. Looking up a definition (left: selecting the action, right: the result); example taken from our lecture notes; cf. Sect. 4.

As the desired MIME type of the response can be given in the HTTP request header for so-called *content negotiation*, we can distinguish requests for content markup from requests for a rendered formula while still using the same URL:

```
GET /backend?action=lookup-definition
 &cdbase=...&cd=transc1&name=sin HTTP/1.1
Host: jobad.mathweb.org
Accept: application/openmath+xml
```

Analogously, the MIME type `application/xhtml+xml` would be used to obtain a response rendered in XHTML with Presentation MathML.

3.3 Unit Conversion

The unit conversion service assumes the OpenMath encoding for units as specified in [DN03]: Base units are symbols in special CDs; derived units can be formed by multiplication or division of base units with numeric factors or other base units. For example:

```
<OMA>
  <OMS cd="arith1" name="times"/>
  <OMI>1</OMI>
  <OMS cd="units_metric1" name="metre"/>
</OMA>
```

The unit conversion service accepts one such expression o, plus a target unit u. If a conversion is possible, the result is returned as an OpenMath expression, which we denote by $\mathrm{uc}(o, u)$. On the client side, this result has to be integrated into the current formula. Let p with $o = c(p)$ be the presentation markup that the user selected; then we add $p' = \mathrm{render}(\mathrm{uc}(o, u))$ as an `maction`-alternative for p to the document.

We have not implemented our own unit converter but use the one developed by Stratford and Davenport [SD08,Str08], which performs conversions according to the OpenMath FMPs of the unit symbols involved. In its current version, their web service does not talk OpenMath but uses string input/output, so we have to convert values between their OpenMath and string representation (e. g. "1 metre").

3.4 Integrated Backends and Environments

For a clean conceptual model, we have treated our web services separately. From an efficiency point of view it does, however, make sense to arrange multiple services in an integrated backend. Consider unit conversion: Stratford's unit conversion web services internally relies on OpenMath CDs that declare one symbol per unit and define conversion rules for obtaining derived units [Str08]. Definitions of symbols are looked up from CDs as well. Last but not least, the rendering service needs notation definitions for the unit symbols, and CD authors often provide default notations for their symbols. Thus, offering those three services independently requires redundantly storing knowledge about symbols in three places. An integrated backend also saves time, as can be seen for definition lookup: With separate lookup and rendering services, the client-side active document has to connect to two web services in succession. An integrated backend could, however, offer readily rendered definitions by composing two of its internal functions and only minimally extending its external HTTP interface (cf. Sect. 3.2). We have implemented an integrated backend that performs rendering and definition lookup, and acts as a proxy for communicating with third-party web services on different domains to avoid security problems due to cross-site scripting.

4 Evaluation

Proof-of-concept demonstrations of individual JOBAD services can be tried at the JOBAD web site [JOB09]. Besides that, we conducted two evaluations to analyze the feasibility and scalability of our framework: 1. We loaded a large OMDoc document into our server and activated the elision, subterm folding, and definition lookup services. 2. We integrated an external unit conversion

service, which was added after the main phase of the development, to get an understanding of the investment needed to integrate further services.

The former involved the complete lecture notes of a first-year undergraduate computer science course. These lecture notes are originally maintained in LaTeX with semantic annotations, which can be automatically converted to OMDoc [Koh08]. The annotations in the source documents comprise content-markup formulae, informal definitions of symbols, and notation definitions. The OMDoc representation is then rendered into the JOBAD format, which is viewed using the JOBAD client, which offers flexible bracket elision, subterm folding, and definition lookup [JOB09]. So far, we have used this as a stress test, but for the Fall lecture we plan an evaluation where one group of our students will work with the static XHTML version of the lecture notes and a second group with the JOBAD-enriched active document.

The most complicated step in the latter evaluation was adapting the string-oriented interface of Stratford's unit conversion web service to our OpenMath interface. Most of the other required functionality turned out to be already available and just had to be composed. We chose the context menu interface and added a submenu containing the target units[2]. Checking whether the selected term was a quantity with a unit reduced to looking up its corresponding content markup (cf. Sect. 2.1) and performing a simple XPath node test on the latter. Sending a string to the web service and waiting for the response is a standard JavaScript function. Rendering the result of conversion (after converting it back to OpenMath) is done by another service. Finally, replacing two XML subtrees in a formula (both in the presentation and the content markup) and hiding the previous presentation tree in an `maction`, is a utility function provided by the JOBAD core and also used by other services.

5 Conclusion and Future Work

We presented JOBAD, our architecture for active mathematical documents. Our documents are generated dynamically from content-markup and viewed in a web browser, via which the reader can change interactively both content and form of the document. JOBAD constitutes the reader interaction component of our research group's framework for mathematical documents. As such, it is fully integrated into the authoring [Koh08,Lan08], notation management [KMR08], and storage [TNT09] work flows developed by our group.

We gently extended the Presentation MathML format to create an interface language, in which the document server can embed into the served document information about interactivity or instructions on how to retrieve that information. This extension is backwards compatible in the sense that the markup is still valid MathML, and switching off JavaScript yields the same static documents as before.

[2] A static list at the moment; obtaining admissible target units for a given input is neither supported by our client-side implementation nor by the unit conversion web service at the moment.

We have implemented and evaluated an initial set of services that constitute a representative selection of the possibilities we envision. Folding and flexible elision work locally, type and definition lookup retrieve additional information based on a symbol URI, unit conversion sends a content markup object as an argument to a web service. The former service is based on presentation markup generated by the server a priori. For the latter two, the JOBAD modules are passed initialization parameters that instruct them about the server and its URL format. For the latter one, parallel markup is utilized to obtain the content representation of a presentation expression.

A specific design feature of JOBAD is its extensibility. Offering new services for documents in the JOBAD interactive document can be achieved by adding very little new JavaScript code. Adding new user interaction components and binding them to JOBAD services is possible with minimal effort. Finally, the JOBAD client code requires only very few properties of the specific server backends, so that the same client can be easily used with different web services even in the same document.

Future work will be based on this, and we intend to rapidly develop more services, but also to invite contributions from external developers. Due to the modularity of our framework, we expect that this work load can be divided into small and manageable units that can be handled efficiently by students. In particular, we intend to approach the following services:

Notation selection: Our rendering service can already annotate every rendered symbol with a reference to the notation definition in the backend that was used for rendering it [KLM+09]. This information can be used to ask the backend for alternative notations, to allow the user to select from them, and have the current formula re-rendered accordingly.

Guided tour (extension of lookup): This service generates a linear tutorial containing an explanation of every symbol in the current selection, and of every symbol occurring in these explanations, and so on, until some foundational theory is reached.

Flattening: Many documents consist of components that are combined by a module system (see [RK08]). A flattening service replaces import links with the (possibly translated) copy of the imported document.

Search: Our group has developed a semantic search engine for mathematical formulae [KAJ+08]. Therefore, a service that searches the web (or the server database) for the selected expression will be easy to realize.

Links to web resources: The OpenMath wiki [Lan09] not only provides symbol definitions, but also hosts discussions about them. Its architecture allows for linking symbols to further web resources, e. g. Wikipedia articles about mathematical concepts, which can then be made available in a document.

Adaptive display of statement-level structures: On the level of definitions, theorems, and proofs, we generate a different kind of parallel markup from OMDoc sources, namely XHTML+RDFa [ABMP08]. We have already used this for visualizing rhetorical structures in mathematical documents (cf. [Gic08]; demo available at [JOB09]) and plan to extend it to structured proofs.

Editing: Our group has developed the Sentido formula editor [LGP08]. An edit service will pass the selected term to a Sentido popup window and eventually replace it in the current document.

Saving: After a user has adapted a document, it is desirable to upload its configuration to the database.

Furthermore, we will integrate the JOBAD architecture into our various integrated document management systems, such as the semantic wiki SWiM [LGP08], and the panta rhei document browser and community tool [pan].

Acknowledgments : The authors would like Jonathan Stratford for help with his unit converter, Christine Müller for fruitful discussions on notation selection, Jan Willem Knopper for hints on designing a modular JavaScript library, as well as David Carlisle for clarifications on MathML. This work was supported by JEM-Thematic-Network ECP-038208.

References

[ABMP08] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and processing. Recommendation, W3C, 2008.

[ACR+08] S. Autexier, J. Campbell, J. Rubio, V. Sorge, M. Suzuki, and F. Wiedijk, editors. *Intelligent Computer Mathematics, AISC, Calculemus, MKM*, number 5144 in LNAI. Springer, 2008.

[Act08] ActiveMath, 2008. `http://www.activemath.org/`.

[AKTV08] A. Ankolekar, M. Krötzsch, T. Tran, and D. Vrandečić. The two cultures: Mashing up Web 2.0 and the Semantic Web. *Web Semantics*, 6(1), 2008.

[Bra97] S. Bradner. Key words for use in RFCs to indicate requirement levels. RFC 2119, Internet Engineering Task Force, 1997.

[CCJS05] A. M. Cohen, H. Cuypers, D. Jibetean, and M. Spanbroek. Interactive learning and mathematical calculus. In M. Kohlhase, editor, *Mathematical Knowledge Management (MKM)*, number 3863 in LNAI. Springer, 2005.

[CCK+08] H. Cuypers, A. M. Cohen, J. W. Knopper, R. Verrijzer, and M. Spanbroek. MathDox, a system for interactive Mathematics. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*. AACE, 2008.

[DN03] J. H. Davenport and W. A. Naylor. Units and dimensions in OpenMath. `http://www.openmath.org/documents/Units.pdf`, 2003.

[Fie00] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[GHJS08] A. Gerdes, B. Heeren, J. Jeuring, and S. Stuurman. Feedback services for exercise assistants. Technical Report UU-CS-2008-018, Utrecht University, 2008.

[Gic08] J. Giceva. Capturing Rhetorical Aspects in Mathematical Documents using OMDoc and SALT. Technical report, Jacobs University, DERI Galway, 2008. `https://svn.kwarc.info/repos/supervision/intern/2008/giceva\_jana/project/internship\%20report.pdf`.

[GM08] G. Goguadze and E. Melis. Feedback in ActiveMath exercises. In *International Conference on Mathematics Education (ICME)*, 2008.

[JOB09] https://jomdoc.omdoc.org/wiki/JOBAD, 2009.
[KAJ+08] M. Kohlhase, Ş. Anca, C. Jucovschi, A. González Palomo, and I. A. Şucan. MathWebSearch 0.4, a semantic search engine for mathematics. manuscript, http://mathweb.org/projects/mws/pubs/mkm08.pdf, 2008.
[KLM+09] M. Kohlhase, C. Lange, C. Müller, N. Müller, and F. Rabe. Notations for active documents. KWARC Report 2009-1, Jacobs University, 2009. Available from: http://kwarc.info/publications/papers/KLMMR_NfAD.pdf.
[KMM07] M. Kohlhase, C. Müller, and N. Müller. Documents with flexible notation contexts as interfaces to mathematical knowledge. In P. Libbrecht, editor, *Mathematical User Interfaces Workshop*, 2007.
[KMR08] M. Kohlhase, C. Müller, and F. Rabe. Notations for Living Mathematical Documents. In Autexier et al. [ACR+08].
[Koh06] M. Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer, 2006.
[Koh08] M. Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2008.
[Lan08] C. Lange. SWiM – a semantic wiki for mathematical knowledge management. In S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, editors, *ESWC*, volume 5021 of *Lecture Notes in Computer Science*. Springer, 2008.
[Lan09] C. Lange. OpenMath wiki. http://wiki.openmath.org, 2009.
[LGP08] C. Lange and A. González Palomo. Easily editing and browsing complex OpenMath markup with SWiM. In P. Libbrecht, editor, *Mathematical User Interfaces Workshop*, 2008. Available from: http://www.activemath.org/~paul/MathUI08.
[MGH+06] E. Melis, G. Goguadze, M. Homik, P. Libbrecht, C. Ullrich, and S. Winterstein. Semantic-aware components and services of activemath. *British Journal of Educational Technology*, 37(3), 2006.
[Mon05] MONET. http://monet.nag.co.uk/mkm, seen March2005.
[Moz09] Mozilla Labs. Ubiquity. http://ubiquity.mozilla.com, 2009.
[O'R05] T. O'Reilly. What is Web 2.0. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, 2005.
[pan] The panta rhei Project. http://trac.kwarc.info/panta-rhei. seen March 2009.
[RK08] F. Rabe and M. Kohlhase. An exchange format for modular knowledge. In P. Rudnicki and G. Sutcliffe, editors, *Knowledge Exchange: Automated Provers and Proof Assistants (KEAPPA)*, November 2008.
[SCI09] The SCIEnce project – symbolic computation infrastructure for europe [online]. 2009.
[SD08] J. Stratford and J. H. Davenport. Unit knowledge management. In Autexier et al. [ACR+08].
[Str08] J. Stratford. Creating an extensible unit converter using openmath as the representation of the semantics of the units. Technical Report 2008-02, University of Bath, 2008. http://www.cs.bath.ac.uk/pubdb/download.php?resID=290.
[TNT09] TNTBase Project. https://trac.mathweb.org/tntbase/, 2009.
[Ull08] C. Ullrich. *Pedagogically Founded Courseware Generation for Web-Based Learning*. LNCS. Springer, 2008.
[W3C] W3C. Mathematical Markup Language (MathML) 3.0 (Third Edition).
[Yah09] Yahoo! Pipes [online]. 2009.