JACOBS UNIVERSITY BREMEN

Integrating Web Services into Active Mathematical Documents

by

Jana Giceva

Guided Research: Final Report
supervised by: Prof.Dr. Michael Kohlhase

in the
School of Engineering and Science
Computer Science

May 2009

JACOBS UNIVERSITY BREMEN

Executive Summary

School of Engineering and Science
Computer Science

Guided Research Thesis

by Jana Giceva

Interactive documents is an increasingly popular idea in todays world. The envision of an (inter)active document is that a reader should also be given the chance to adapt the document to his/her own preferences and needs apart from just reading it. This adaptation is not only in terms of customizing the display in the browser but also changing the content, by retrieving additional data from other related services offered on the web. The JOBAD architecture provides means for developing mashups of mathematical services within a document. While some of the interaction is to be based solely on the data already embedded in the document itself, most of the integrated web-services depend on the asynchronous client-server interaction and database information retrieval. The JOBAD server would provide some core services such as document rendering and serving, providing the available services to be uploaded in the active document, and a simple proxy mechanism that would allow other remote web-services to be used as services called from the JOBAD document. A variety of services can be supported with this architecture, some of which are already implemented such as the definition lookup and unit-conversion services, and will be presented in this paper. Eventually an evaluation will be provided and the results will be elaborated.

Acknowledgements

I would like to thank Christoph Lange and Florian Rabe for the great guidance and direct support throughout the whole guided research. Furthermore special thanks for Prof.Dr.Michael Kohlhase, Christine Müller and the whole KWARC group for helping me with their suggestions, and evaluations. I would also want to thank Jonathan Stratford for the help with his unit converter and Jan Willem Knopper for his hints on designing a modular JavaScript library.

Contents

List of Figures

1

Introduction

1.1 About Active Mathematical Documents

Throughout the years documents proved to be a vital interface for distributing not only mathematical but any type of knowledge. Only in recent years though, with the rapid development of technology did the format of the documents change. Digital documents became and still are at the focus of attention with the structure, format and other kinds of additional values that they add to the original idea of a document - to just statically present knowledge.

There has been quite some effort involved in creating and delivering documents that would adapt to the reader's needs. There is ongoing research for both adapting the documents to the reader's preferences prior to their serving, as well as for using the same as an interface for calling a few specific web-services.

Active mathematical documents belong to the class of web-documents. This is mainly due to the web-document formats like XHTML+MathML+JavaScript (see 1.3) that provide means for easily achieving the document interactivity. This interactivity would allow the reader to adapt the document in question to his/her own preferences while in the process of reading it, not only in terms of content presentation and display but also by retrieving additional data from the various services offered on the web and embedding it in the document.

These possibilities have led to numerous research engagements for interactive mathematical documents that offer features such as adapting mathematical documents, interactive learning and solving exercises. In the following section, an overview of the current work in this field is to be discussed in more detail.

1.2 Related Work

ActiveMath and MathDox have realized a project for Interactive Exercises [2–4], where the user can enter the result into a form, and then receive a feedback from a solution checker implemented in a server backend. ActiveMath introduces and uses its own web-services [5]. MathDox, on the other hand, has been originally designed for talking to computer algebra systems, but can currently 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 [6]. Apart from supporting the MathDox's communication protocol Gerdes is also compatible with the XML-RPC and JSON data exchange standards [6]. Although potentially open to clients, so far the services developed withing the ActiveMath and MathDox projects, have not been used with other frontends except their initial ones [5, 7].

Furthermore, there are also web-service architectures for mathematical web-services, designed for integration with many systems, such as the ones developed in the MONET [8] project. MONET is an architecture that allows for any kind of mathematical web-service. Yet, for now, only computational web-services have been developed and evaluated within it. The framework consists of a MONET broker, where the interested web-services are registered. The broker then accepts requests, that mainly consist of a problem description (e. g., solve an equation) given as an OpenMath expression, and then forwards the request to the best-matching of the registered services. MathDox, that was mentioned above, provides a mechanism for accessing the MONET web-services via a document interface.

The asynchronous communication with a server backend (provided by AJAX - see 1.3), allows for a client/server interaction with no form submission. This provides base for developing responsive browser-based applications, where a client side JavaScript can exchange data packages with a server backend, and insert the server responses into the current page. This technique is employed by MathDox [3] and Gerdess frontend to their feedback service [6].

Although quite some effort, as mentioned above, is engaged in this area the web is still dominated by static mathematical documents. Even the active documents, that act as frontends to web-services, are somewhat limited, as they have been designed to give access to a small selection of web-services, performing very specific tasks. These documents are mostly designed and used to give feedback to user-solved exercises and for symbolic computation, as in the case with MathDox and ActiveMath.

A newly emerged technology of mash-ups in Web 2.0 [9, 10] provides means for even further advancement in the integration of diverse web-services into documents. Initially the original mashups consisted of handcrafted JavaScript gathering web-services from different sites. As of now, we have seen several mashup development kits emerging such as Yahoo! Pipes [11] and Ubiquity [12].

I would like to thank Christoph Lange for letting me use the Related Work section from the Web Services for Active Mathematical Documents [1] paper, when writing this section.

1.3 Wrap-up of Available Technologies

In this section I would briefly introduce the technical terminology that will be used later on in this paper.

The client side:

- **JavaScript**:scripting language used on the client side in the development of dynamic websites. It is the technology that along with XHTML enables the whole concept of responsive active documents, discussed in the previous section. [13]
- **jQuery**: jQuery is a lightweight JavaScript library that emphasizes the interaction between JavaScript and HTML, abundantly used in the development of JOBAD client side, as it provides features such as DOM element selection, traversal and modification, event handling, effects, animations, AJAX and CSS manipulation. [14]
- **XPath**: the XML Path Language, is a query language for selecting nodes from an XML document, enabling querying of XML documents, where jQuery had limited power [15]

The server side:

- **Apache Tomcat**: a servlet container where the Java classes responsible for document rendering and serving, client requests handling, and simple proxy data exchange methods are stored and invoked. It provides a "pure Java" HTTP server environment for Java code to run. [16]
- **Java Servlet**: A Java program that runs as part of an HTTP server and responds to requests from clients. It was chosen as a development environment for

the JOBAD server classes, because of the wide acceptance of the Java language, abundant online documentation and help pages. [17]

- **JOMDoc library**: a Java API for OMDoc documents, which facilitates the parsing of OMDoc XML documents into a Java data structure, to manipulate them conveniently. Its renderer transforms OMDoc documents into XHTML+MathML documents, and is the tool used by the JOBAD server when rendering the active mathematical documents prior to their serving. [18]

The document structure:

- **MathML**: the Mathematical Markup Language is an XML application for describing mathematical notation and capturing both its structure and content. This along with XHTML and JavaScript is the platform for creating active mathematical documents. [19]
- **Presentation and Content MathML Markup**: Presentation MathML focuses on the display of a mathematical expression [20], while the Content MathML focuses on the semantic meaning of it [21]. Both presentation and content MathML can be combined in a document in a concept known as parallel markup, where both and explicit presentation and content are provided in a pair of trees. [22]
- **OpenMath**: markup language for specifying the meaning of mathematical formulae, that can be used to complement MathML, and can in addition be also encoded in XML and binary format [23]
- **OMDoc**: or Open Mathematical Documents is a semantic markup format for mathematical documents. While MathML only covers mathematical formulae and the related OpenMath standard only supports formulae and "content dictionaries" containing definitions of the symbols used in formulae, OMDoc covers the whole range of written mathematics. [24]

The client/server communication protocol:

- **REST**: or REpresentational State Transfer is a model for web services based solely on HTTP. It takes the view that the Web already has everything necessary for web services, without having to add extra specifications like SOAP and UDDI. Any item can be represented at a URI and it can be manipulated using one of the simple operations defined within HTTP (GET to retrieve information, PUT and POST to modify it, DELETE to remove it). Using its pattern, mathematical resources such as for instance OpenMath symbols could be easily represented and accessed directly with URLs. [25]

- **AJAX**: or asynchronous JavaScript and XML, provides for asynchronous communication between the client and the server making the browser interactive applications as responsive as the desktop applications, thus more reliable and compelling for the reader. [26]

1.4 JOBAD project introduction

The aim of JOBAD is creating a mashup development kit, similar to the ones mentioned in section 1.2, this time for mathematical applications. With it, an interactive document would become a document that not only offers the user contents for reading but also the opportunity to adapt it according to his/her own preferences and needs. This adaptation would also include retrieving additional information from other services offered on the web, which will in most-cases require a document re-rendering.

Many use-cases can be given as examples of how these documents can prove useful. Simply 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 [27]) that can be invoked from within the document that would replace the units with the familiar ones.

I would like to thank Christoph Lange for letting me use the use-case examples from the Web Services for Active Mathematical Documents [1] paper.

2

Work Documentation

2.1 JOBAD project architecture

The JOBAD project architecture is divided into three main parts:

- actual mathematical services
- graphical user interface elements
- document manipulation functions

Figure 2.1 illustrates the architecture design and the interdependencies between the building blocks. The design is such that more emphasis was put on the client side, making the server side as lightweight as possible. This leaves space and freedom for the JOBAD users to be able to write their own service plug-ins using the existing client/server infrastructure.

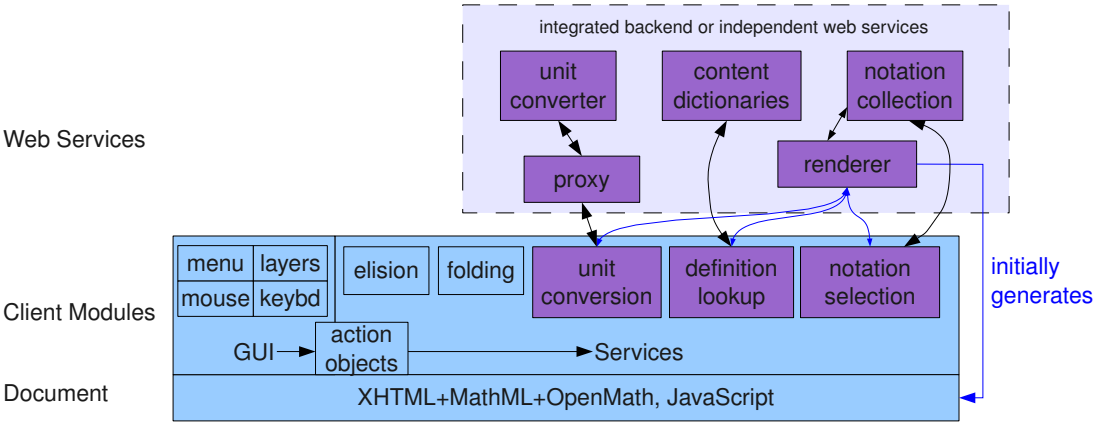

FIGURE 2.1: JOBAD Architecture. Note the central role of the rendering service, which both generates JOBAD-compliant documents and is needed for many other services. Courtesy of the authors of [1]

On the client side, JOBAD is actually a modular JavaScript library, consisting of one main module, responsible for loading the collection of independent modules for each of the provided services, as chosen by the user. In addition to the service specific modules there are also helping modules that provide the basic support for document structure and content manipulation, as well as information extraction and server communication.

The server on the other hand is in charge of providing the collection of service modules to be loaded in the document, and informing the main client module of their availability and location. It is also responsible for document rendering and processing the client service requests, contacting the corresponding web-service or backend and sending the received response back to the client.

The interactive services supported by this architecture can be classified by the amount of data exchanged with the web service backend in the following way:

- services that use the data embedded in the document
- services that send an action verb and a symbol identifier in order to retrieve additional information
- services that send complex mathematical content expressions and an action verb telling the server what it requires from it

These services however depend heavily on the format used for the interactive documents. Apart from presentational markup, some additional meta-information about the current appearance can be embedded in the document in order to enable document-user interaction, such as alternative presentations, or parallel content markup. If this is not available then additional information needs to be embedded so that the JOBAD client side can retrieve the necessary data to perform the offered services. Further discussion on this topic follows in the next section.

2.2 Document format needed for the services

As extension to the MathML specification the following additional requirements for the structure of a document are needed by JOBAD, so that it operates its services:

- Alternative displays, from which the user can choose from, should be realized by `maction` elements, where the `@actiontype` attribute is set according to the type of service [28, section 3.6.1]. For instance, a service that replaces the original presentation of the formula, keeps the previous state of it as an alternative to which the user can switch back.

- Parallel content markup [28, chapter 5.4] has to be provided for services that acquire access to the semantic part of the selected mathematical expressions. There must be cross-links from the subterm-grouping elements to the corresponding content elements.

Alternative Display

The first requirement of alternative displays is needed by several services currently supported, such as the folding and the unit conversion services. The folding service operates solely on data embedded in the document, where the user can choose from two or more alternative displays of a certain mathematical expression. An example would be to offer the user an abbreviation of the mathematical expression as its alternative. (For more detailed description please refer to [1]). The unit conversion service, substitutes the original presentation markup of the mathematical unit expression with its conversion, keeping the original display hidden in the document; thus offering the user to switch back to the initial display without having to contact the remote web service again. In order to achieve this switching among presentations the `maction` element is used. The alternative displays are placed as children to the `maction` element, and are switched according to the `@selection` value. For instance in the following example listing 2.2 the browser displays the $c^2 = a^2 + b^2$ element, likewise if `@selection` was set to 2, the $c = \sqrt{a^2 + b^2}$ element will be shown.

```
<maction actiontype="show_alternative" selection="1">
  <mrow> c^2 = a^2 + b^2 </mrow>
  <mrow> c = √(a^2 + b^2)</mrow>
</maction>
```

LISTING 2.1: maction object format and usage of @selection

Parallel Markup

The second requirement - cross-linked parallel markup [22] is needed to obtain the corresponding content elements of the mathematical expressions whose presentation markup was selected by the user. Although the direction of this cross-linking between content and presentation markup is not specified in MathML, in JOBAD it is fixed to "presentation→content". It is chosen as such, because several presentation symbols can be linked to one content symbol which is not the case vice-versa. For instance $a+b+c \rightarrow plus(a, b, c)$. Thus in JOBAD, the `@xref` is needed in symbols, numbers, identifiers, and

FIGURE 2.2: 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. Figure taken from paper [1], with courtesy of Christoph Lange

subterm-grouping presentation elements in order to link them with their corresponding content elements. Whenever a user selects a whole mathematical expression, then the closest common ancestor of all selected XML elements that has an `@xref` filled is determined and then cross-linked to obtain the corresponding content expression. An example is given in figure 2.2

For further information about additional MathML requirements, that are needed by JOBAD but not related to this guided research thesis, please refer to the Web Services for Active Mathematical Documents paper [1, chapter 2.1]

2.3 User Interface

The idea on the client side is to provide a variety of user interface elements for input and output. Currently a right click on a math expression, triggers the appearance of a context menu meant for the element that is under the cursor or for the selected range of mathematical elements if such was made. The selection is done in the usual way of dragging the mouse, but only the whole logical expression is evaluated i.e. the nearest selection encapsulating container gets selected with its contents, and the corresponding available services is displayed in the dynamically populated context menu.

Calls to the services are implemented by generic action objects, which allows for diverse access to them, thus providing framework for extensible user interface. For instance, the same elision action element can be triggered either by the local context menu, or

by future formula local toolbar. This can be further extended to recognizing global keyboard shortcuts and invoking the corresponding action object accordingly.

Apart from directly changing the display of a mathematical expression, which was described in the alternative displays part 2.2 JOBAD also offers tool-tip-like popups, used for displaying results from some information-retrieving services. This information can be annotation that is embedded but hidden in the documents itself, but mainly refers to a response from the supported web-services, such as symbol definition look-up.

2.4 Implemented Services

The JOBAD project is designed to be flexible with the web-services offered. It is done in such a way that it leaves the users space and freedom apart from using the already implemented services to also define their own, relying on the infrastructure and functionalities available. The JOBAD document server would advertise all the available web-services, and for every client-chosen service the corresponding JavaScript module will be served, and loaded by the main client side JOBAD module.

As most of the existing web-services are built upon and function over the HTTP, and the objects they interact with can be conceived as resources with URLs, it is clear why the REST pattern was chosen as means for communication between the client and the server. The URLs can represent mathematical resources such as symbols, so they can be directly used to retrieve the definition or type of a symbol. This type of supported services, with only a small amount of data-package exchange between the client and the server, are referred to as symbol-based services. The other type of services are the ones that require a transmission of a more complex mathematical expression, that was selected by the user, that would be included in the body of the HTTP request, and they are referred to as expression-based services. The descriptions of both symbol-based and expression-based services must consist of the content element name, cross-referenced from the user-selected element and a URL that invokes the service. The URL may also contain placeholders for the content dictionary of the document sending the request.

The services developed until now show to some extent the power of the design, and by going through them and their implementation one will understand the impact of such a document interactivity and the range of further extension possibilities and project scalability.

2.4.1 Service Description

The currently available web-services supported by JOBAD are:

- rendering service
- definition look-up service
- unit conversion service

In this section I will present each of these services separately, and show and describe some of their features.

Rendering

The rendering service plays the grand role of converting the output obtained from other services to XHTML+MathML format, that would be then delivered and presented to the reader. Whether that would be rendering of the initially presented document or transforming a simple fragment of OpenMath to Presentational + Content MathML, the rendering service is and will be the most abundantly used service present on the server side.

Definition Lookup

In the definition lookup service, the client side JavaScript sends the content identification of the symbol, previously selected by the reader, to the server and receives as a response an expression or a formula that represents the definition of that symbol. In a similar way, one can also retrieve the type of the symbol, as type lookup service implementation is analogous to this one. The communication between the client and the server, as explained before, uses the RESTful URI format, where the type of information one is interested in retrieving is coded as the value of the action parameter. The value it currenlty takes is `definitionLookup`, but later also the `typeLookup` value will be put into practice.

The server side then performs a query in the OMDoc-based content dictionaries, assuming the document is written in OMDoc and retrieves the requested information. It then sends back the JOMDoc rendered query result to the client side. Currently, the client side of JOBAD presents this response to the user by dynamically populating a tool-tip, with the human readable presentation of the definition of the selected symbol as show in figure 2.3. Note that the response from the server needs not be a rendered formula in

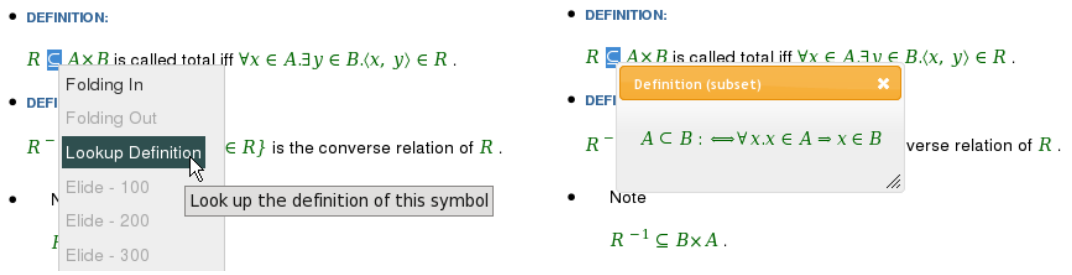

FIGURE 2.3: Looking up a definition (left: selecting the action, right: the result) courtesy of Christoph Lange from [1]

XHTML with Presentational MathML but can also be only the content markup retrieved from the content dictionary. This can all be settled via the options offered by the HTTP request header with the so-called content negotiation [29] where the client can specify the desired MIME type of the response. This can be set to application/openmath+xml when accepting a response of content MathML, or application/xhtml+xml for accepting a response of XHTML with presentational MathML i.e. JOMDoc rendered result.

Unit Conversion

The unit conversion service allows a user to select part of a mathematical expression that is related to the displayed unit and then choose to convert it to another unit, belonging to the same dimension [27], from the context menu. The formula is then re-rendered, changing the unit and its value to the desired one, keeping the original value embedded in the document in case the user wants to convert back to it later on.

This unit conversion service is set to work with the OpenMath encoding for units as specified in [30], where base units are symbols in special CDs, while derived units can be formed by multiplication or division of base units with numeric factors or other base units. For instance:

```
<OMOBJ>
  <OMA>
    <OMS cd="arith1" name="times"/>
    <OMI> 1 </OMI>
    <OMA>
      <OMS cd="units_ops1" name="prefix"/>
      <OMS cd="units_siprefix1" name="kilo"/>
      <OMS cd="units_metric1" name="metre">
    </OMA>
  </OMA>
</OMOBJ>
```

LISTING 2.2: OpenMath format for representing units

Ideally, the unit conversion service would accept one such OpenMath expression o, as well as the target unit u, the client has chosen. If conversion from o to u was possible, which is the case when both units belong to the same dimension [27], then the result would as well be returned as an expression in the OpenMath format, which can be referred to as $uc(o, u)$.

On the client side, this result has to be integrated within the current formula, however one does not want to lose the previous unit-expression but keep it hidden in the document, in case the reader wants to switch back to the original unit. Let's denote with p the presentation markup of the user selected element o, then we can add the new expression render$(uc(o, c))$ as an `maction`-alternative for p to the document. This time the `@selection` is set to the latter one, so that the expression with the desired unit is presented to the user.

For the purpose of this service, the unit converter developed by Stratford and Davenport [27, 31] was used. This converter performs conversions according to the OpenMath Formal Mathematical Properties (FMPs) of the unit symbols involved, see listing 2.4.1 for how the mili unit is defined.

```
<OMOBJ>
 <OMA>
  <OMS name="eq" cd="relation1"/>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMI> 1 </OMI>
    <OMA>
      <OMS name="prefix" cd="units_ops1"/>
      <OMS name="mili" cd="units_siprefix1"/>
      <OMS name="unit"/>
    </OMA>
  </OMA>
  <OMA>
    <OMS name="times" cd="arith1"/>
    <OMA>
      <OMS name="power" cd="arith1"/>
      <OMI> 10 </OMI>
      <OMI> -3 </OMI>
    </OMA>
    <OMV name="unit"/>
  </OMA>
</OMA>
</OMOBJ>
```

Listing 2.3: OpenMath FPM definition of semantics for the mili unit

In its current version however, the converter does not talk OpenMath but uses string input/output, so the current implementation of the unit-conversion service, as you will be presented with in the following section, actually performs the communication with exchanging string expressions such as "1 metre".

2.4.2 Service Development and Implementation

In the following section, a short description is provided of the development process and implementation of the services described above (see 2.4.1).

Rendering Service implementation

The client sends a request to the JOBAD server with an `action` parameter value set to *render_document* and the document name as in the following HTTP GET request:

```
JOBAD/documents?action=render_document&document=mathtestcase
```

The JOBAD server side, that was developed using the Java Servlets, parses the request received from the user, and acts according to the `action` parameter value. For the document rendering service the main JOBAD server function retrieves also the name of the document requested before calling the corresponding **serveJOMDocRendered()** function. It will then retrieve the document from the document database uploaded on the server, and serve the rendered result back to the user. The actual rendering service is performed using the JOMDoc library, which implements the rendering algorithm described in [32, 33]. It has access to a collection of notation definitions, which map content markup patterns to presentation markup templates [33]. It will then have to go through another transformation, again performed by the JOMDoc library, this time to enable the required JOBAD add-ons. The JOMDoc is called using its **transform()** function and with the **JOBAD-postproccessing.xsl** XSL style sheet that populates the document with the neccessary JOBAD utility elements, such as the UI elements like the context-menu and the tool-tip and other css/script elements that set the JOBAD working environment like the main **jobad.js** and the **jobad.css** files.

The above-mentioned implementation exists, however it does not behave as expected. The problem is that whenever the documents are served by the JOBAD Servlets, not all JavaScript client modules are working, although loaded in the document and validated by JSLint [34]. There is also a problem occurring when loading the jQuery script from the local server. It was discovered that the problem is in the long time response it requires, and thus it is currently called from the official jQuery web-page [14]. These

technical limitations were not solved in the times-pan of the guided research, thus a temporary solution was provided that allows for further testing of the other offered and supported services. The testcases are pre-JOMDoc rendered and pre-JOBAD enabled before uploaded on the Apache Tomcat server, and are called by the user directly from there without being served by the JOBAD Servlet.

The rendering service, however, does not only consist of document rendering but also of rendering other mathematical expressions from OpenMath to presentational MathML or parallel markup. This is the widely used feature by the other services, where the response from the other local or remote web-services is not human readable and the client side expects a response of MIME type: `application/xhtml+xml`.

Definition Lookup Service implementation

The definition look service already requires more involved client-side modules activity. Once the client selected a symbol in the document, and asked for its definition, the definitionLookup module A.1.2 is activated. It retrieves the corresponding content element to the user-selected one, using the parallel-markup support module functions. When the definitionLookup module is in possession of the content element, it creates a new message that is then sent to the JOBAD server. This new message consists of the action parameter set to expandDefinition, a string value of the identification of the symbol content element, and the corresponding values of the cd and cd base as in the following example:

```
JOBAD/mathtest?action=expandDefinition&cd=arith1&cd_base=bla&symbol=div
```

This message is then forwarded to the AJAXsupport module functions, where the connection to the server is established and the message is passed on to the server side. Currently, there are AJAX support functions available for both GET and POST HTTP methods, but for the sake of debugging and getting-a-clear-overview of the messages passed back-and-forth between the client and the server, the GET method was used both in the development of this and the other supported web-services.

The server side then parses the received request and invokes the corresponding module for handling the definition lookup service - **serveDefinition()**. On the server, a symbol with type declaration and definition is represented as shown below (see listing 2.4.2), which allows for easy retrieval of the requested information with XPath.

```
<!-- ℂ → ℂ -->
<symbol name="sin">
  <type system="sts.omdoc#sts">
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMA>
        <OMS name="mapsto" cd="sts"/>
        <OMS name="NumericalValue" cd="sts"/>
        <OMS name="NumericalValue" cd="sts"/>
      </OMA>
    </OMOBJ>
  </type>
</symbol>
<!-- sin z = 1/2i (e^iz − e^−iz) -->
<assertion xml:id="sin-prop-1" type="lemma">
  <CMP> sin(x) = (exp(ix)-exp(-ix))/2i </CMP>
  <FMP>
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMA>
        <OMS cd="relation1" name="eq"/>
        <OMA>
          <OMS name="sin" cd="transc1"/>
          <OMV name="x"/>
        </OMA>
        <OMA>
          <OMS name="divide" cd="arith1"/>
          <OMA>
            <OMS name="minus" cd="arith1"/>
            <OMA>
              <OMS name="exp" cd="transc1"/>
              <OMA>
                <OMS name="times" cd="arith1"/>
                <OMS name="i" cd="nums1"/>
                <OMV name="x"/>
              </OMA>
            </OMA>
            <OMA>
              <OMS name="exp" cd="transc1"/>
              <OMA>
                <OMS name="times" cd="arith1"/>
                <OMA>
                  <OMS name="unary_minus" cd="arith1"/>
                  <OMS name="i" cd="nums1"/>
                </OMA>
                <OMV name="x"/>
              </OMA>
            </OMA>
          </OMA>
          <OMA>
            <OMS name="times" cd="arith1"/>
            <OMI>2</OMI>
            <OMS name="i" cd="nums1"/>
          </OMA>
        </OMA>
      </OMA>
    </OMOBJ>
  </FMP>
</assertion>
```

Listing 2.4: symbol definition and type declaration as kept on the server - taken from the transc1.omdoc cd

The definition lookup was actually implemented using the XOM Java XML library [35], that enables the servlet to transform the read document from a stream of bytes to an actual document, which can be later on queried easily. The query for the definition of a symbol returns as a result a content element, which is then sent to the rendering service to transform it to presentational MathML expression. The rendered result is eventually sent back to the client.

On the client side, the AJAX response handling function is created as a callback function and is received as a parameter from the function initiating the client/server communication. In this case the handling function belongs to the definition lookup module, and according to the current implementation it presents the rendered definition in a tooltip overlay, displayed at the cursor position. (see figure 2.3).

Alternatively, definition expansion is also possible, as the difference would only be in the response handling function. Instead of displaying the definition by dynamically populating the tooltip, the selected occurrences of the symbol would be replaced with its definition by re-rendering the formula. This also requires that the original formula is kept as an `maction` alternative using the `@actiontype` to "definition-expansion". Please refer to chapter 4 for a more detailed explanation for this service.

The type declaration lookup service would be implemented in exactly the same manner, only replacing query string on the server side to look for the type declaration of the symbol instead of its definition.

Unit Conversion Service implementation

The unit conversion service A.1.2 actually assembles all the other supported services and provides a good overview of what can be done with the JOBAD architecture. The original design was to show how remote web-services can be integrated within the JOBAD client side, without having to go through the JOBAD server. That would allow for a future developer to simply connect to a certain web-service via the available JOBAD utility module functions and handle the data exchange and response handling with not much difficulty. This however turned out to be not feasible despite what intuition says, due to some technical limitations. The problem is that because of security aspects Ajax does not support cross-domain references(see Same origin policy at [36]). As a result, the JOBAD client cannot simply connect to a remote web-service and exchange data. Therefore it was decided to extend the JOBAD server side with a simple and as lightweight as possible proxy that would enable the communication between the JOBAD client and the other web-services. This can be considered as a separate service offered,

which accepts a request from the client, with an `action` parameter set to "connectionService", and then the complete URL, that is contacted by the proxy along with the corresponding parameters set, included as a value to the `url` parameter. Note that this requires double encoding, and a usage of the POST HTTP method, to overcome the length-limitations of GET. Then the request sent by the client to the server has the following structure:

```
POST /backend?action="connectionService"&url="http://remote-url.org" HTTP/1.1
Host: jobad.mathweb.org
Accept: application/xhtml+xml
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
```

LISTING 2.5: sample of a POST request body format

The communication technical foundations being explained, I would now proceed to the actual implementation of the unit conversion service. As a visual guidance for the following explanation section one can use the sequence diagram presented in the next page (see Figure 2.4.2).

The user's selection of a mathematical expression is taken, and if the selected presentation element references a content element of type as shown in listing 2.4.1 the unit conversion will be shown as an option in the context menu. If chosen by the user, then the unit conversion module function checks the context within which the chosen presentation element is in. If it already has the unit-value pair, the user is interested in, as an `maction` alternative then only a switch in the `@selection` value is performed otherwise the client JOBAD side needs to connect to the online converter and retrieve the neccessary value from there.[1] For that the `convertTo()` function contacts first the parallelmarkupsupport module function `getContentElement()` and obtains the content element to which the user selected presentation one refers to. Having the unit and its value, the convertTo() function creates the parameter list that would be sent to the unit converter, and passes it to the AJAXsupport module functions. Unlike the definition lookup service, this time the HTTPconnection() function is used but implementing the POST HTTP method, that connects with the JOBAD server, asking it to contact the remote service at the URL passed as a parameter and forward back the response.

The server side, then receives the request and activates the simple proxy, calling its `URLconnection()` function and passing it the contents of the request `url` parameter as the URL to be contacted. The proxy establishes the connection to the remote converter and returns back the converter's response to the main `doGet()` method of the JOBAD servlet, which forwards it back to the client.

[1] the maction-check is not currently implemented, but is mentioned for completeness

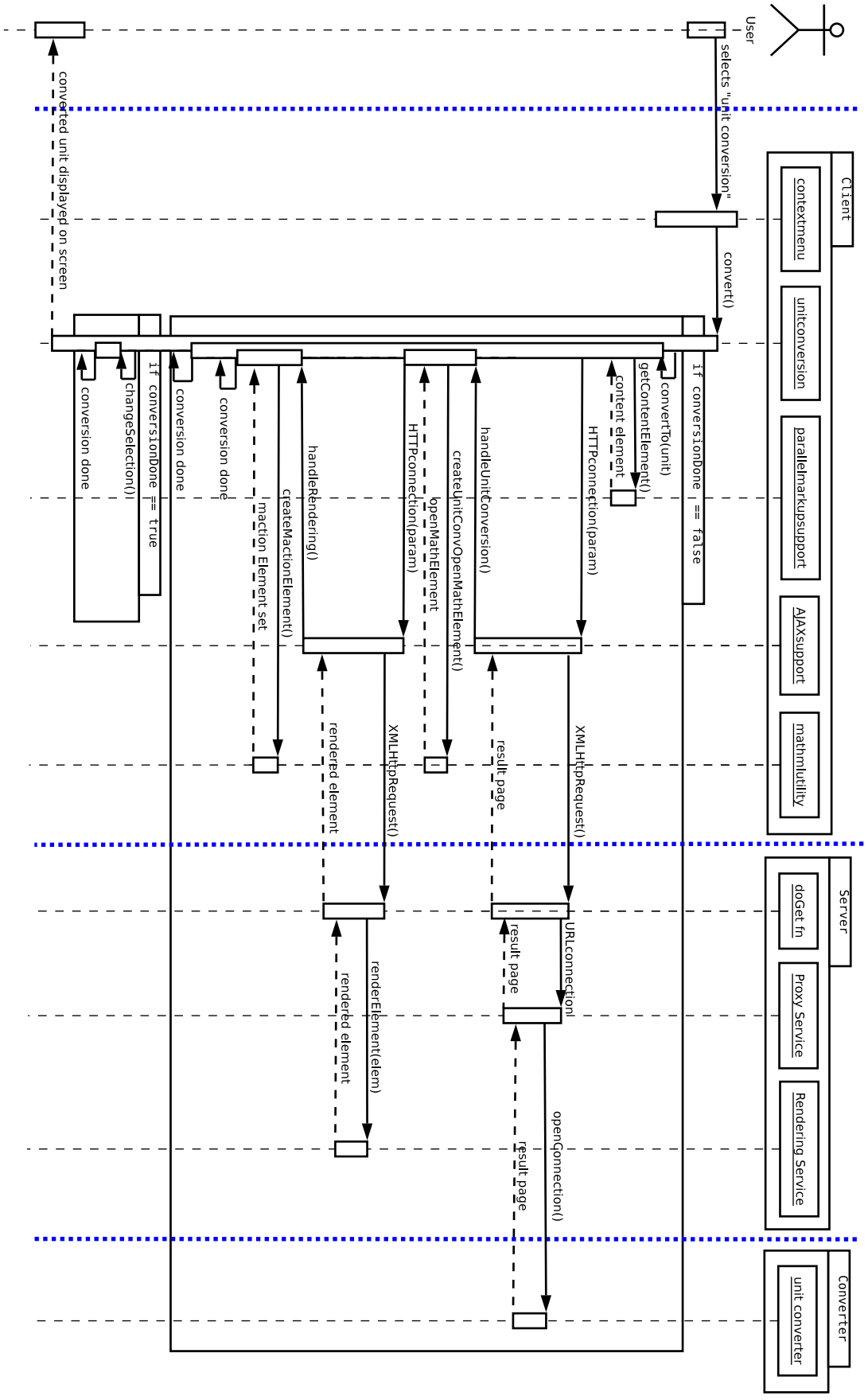

FIGURE 2.4: unit conversion sequence diagram

This simple proxy design was chosen over its alternative due to its simplicity, and the possibility for further expansion to other web-services that would depend on the services provided by some remove web-services, without making the future developer to have to deal with the JOBAD server code in addition to the client side request preparation and the response handling. The alternative, of making the proxy a bit smarter, would require additional methods on the server side that will actually parse the response of the converter and send only the relevant piece of data back to the client, in its original state or JOMDoc rendered depending on the expected response. This would lower both the traffic load and the amount of messages exchanged between the JOBAD client and server, but only in cases when the web-service response is not of the needed format (and thus needs rendering) and its size is bigger than usual, as it was the case with the unit-converter (see below for details). When considering both of these alternatives, apart from the extensibility simplicity that was preferred when choosing the first design, also the nature of real web-services was considered. The web-service that we are currently testing the JOBAD design with, the unit conversion, uses an online converter that is not really built to be a web-service and thus the received response is much larger than the one one would expect to obtain from a regular web-service. This is because the unit converter has an interactive user interface, which is served together with the result of the conversion. When dealing with a response from a real web-service, where the result is relatively small, the re-transmission of the same message one more time over the web should not cause much overhead. In addition to this, a notable consideration is also the type of the response received from the web-service. The converter, we are using, accepts queries and returns results in the form of a string, which requires additional parsing and conversion, however this should not be the case with other web-services. In some cases it is expected that the neccessary format would be returned, as with the services designed in the MONET and SCIEnce [37] projects that return OpenMath, and with it, no additional parsing or rendering would be needed.

On the client side, the AJAXsupport function HTTPconnection() that established the connection with the server, passes the response to the corresponding response handling function, that was yet again passed as a callback function by the convertTo() function. The handling function, parses the result page document and extracts the relevant values using XPath. It then uses the mathmlutility module functions to create an OpenMath object that contains the new content elements for the unit and value pair received. The content object is then sent to the JOBAD server again using the AJAXsupport functions, that ask the server for the rendering service by setting the corresponding value to the `action` parameter.

The server side, passes the received OpenMath object element to its rendering service, which as explained before (see section 2.4.2), transforms the content element to a

presentational MathML expression and sends it back as a response to the JOBAD client.

On the client side, the response handling function overtakes the control, by creating an `maction` element at the place where the initially selected presentation element is, and adding the server response as an `maction` alternative to the original presentation element, setting the `@selection` attribute to the newly added alternative and by it displaying the user the mathematical expression in the units desired.

3

Evaluation

3.1 Evaluation of Work

In order to test and evaluate the JOBAD design and the services implemented so far, two evaluation techniques were used:

- analyzing the architecture and its extensibility with the implementation of a "new and different" web-service - unit conversion service
- getting user's feedback

Both of them are discussed briefly, followed by a short discussion of the results.

3.1.1 Analyzing the Architecture

The main goal for the implementation of the unit conversion service was to test and analyze the feasibility and scalability of the JOBAD framework, as well as to get an understanding of the investment needed to integrate other new services.

As explained in more detail in section 2, with the implementation of the unit conversion service the JOBAD design was enhanced by introducing the simple proxy that allows for future services to connect to other remote servers and use their services. This being added, the most complicated step during the implementation was the OpenMath to string and vice-versa conversion that was necessary in order to adapt to the string oriented interface of Stratford's unit conversion service from the OpenMath based interface in JOBAD. The rest of the required functionality turned out to be already available and just needed small modification and final composition of the offered helping module

functions. The context-menu was chosen as an interface, that was already implemented, and was just extended to offer the new unit conversion service, offering the target units. The parallelmarkupsupport functions helped when checking if the selected term was a quantity and obtaining its unit to be converted. The AJAXsupport module functions and the newly introduced server proxy service provided the support for sending a string to the web-service and retrieving the response, while the rendering of the result of the conversion (after converting it back to OpenMath) is done by the rendering service. Eventually, replacing two XML subtrees in a mathematical expression (both in the presentation and the content markup) and hiding the previous presentation tree in an maction, is a utility function provided by JOBAD, used by other services.

3.1.2 Getting User's Feedback

The other evaluation criteria that was considered, concentrates on another relevant aspect - usability, user-friendliness and general audience acceptance.

For this purpose audience from different knowledge backgrounds was asked to play around with the offered documents at https://trac.omdoc.org/jomdoc/wiki/JOBAD and answer several questions that ranged from how they find the active documents idea, to questions about intuitive interface and helpful services. Finally they were asked to contribute with further wishes and ideas for improvement.

In general, maths and computer science people, thought that the active mathematical documents idea has a great potential, and that it will be widely accepted by the audience, mainly because they will no longer have to adapt themselves to the document they are reading but rather adapt the document instead. They said that it would simplify the user-interaction on the net and definitely help the less experienced people get the information they want so that they understand the document they are reading. The best accepted service by far was the unit conversion service followed by the definition lookup service, which they found especially useful in the field of logics. Suggested services included the following: simplification and evaluation of a mathematical algebraic expression, conversion of a formula into another notation, or simply connecting to another remote service with a selected expression where other more advanced options will be offered. A use-case would be understanding the Gaussian distribution formula, found in a document, by plotting it and analyzing the result.

Physicists and chemists also liked the idea and were quite visionary with respect to the usage of interactive mathematical documents in their fields. Physicists especially found the unit conversion service compelling, and asked further questions of whether the unit conversion performed on the chosen specific element can be also extended until the end

of the mathematical expression, eventually getting the final result in the desired unit. They also preferred more detailed theoretical explanation on the selected symbol, rather than a sentence , which they said might only confuse them more. Another request was in terms of the definition expansion, where they asked if a symbol, that was previously derived in the document in terms of other more familiar symbols, can be replaced using that definition.

Students from the School of Humanities and Social Sciences agreed that the active documents are really nice and that they make reading and learning much easier and faster. Especially for the mathematical documents, where they run into unfamiliar symbols much more frequently than other users. The problem as they put it is that: when coming across on an unfamiliar word its definition can be easily found on any web dictionary, but this is not the case when encountering a complex mathematical symbol that is not easily typed, and thus they found the definition lookup service most useful.

In general almost all people asked whether they can save their own configuration of the document, once they are done adjusting it, and if they can retrieve it the next time they ask for that particular document.

3.2 Discussion of Results

The JOBAD design and its extensibility as well as the implemented services were evaluated in the previous sections. The results show that the framework architecture is suitable for further integration of new web-services, proving the future developer with most of the necessary functionalities as it was the case when implementing the unit conversion service. The feedback received from users shows that the audience is accepting the whole concept quite easy and visionary. Almost all grasped the power of the interactive documents, and found the offered services quite useful and intuitive.

4

Future Work

4.1 Implementation Improvements

In the following section, a small list of tasks is given, that are either suggesting ideas or stating unsolved problems, both of which would further improve the current implementation of the services.

1. As mentioned previously, there is still an unsolved problem with the serving of active document via the JOBAD server. This would is one of the most important tasks left for the future development of the project.
2. Modularization of the JOBAD server - task that would eventually offer the existing and new functions grouped by the service they provide. Is expected to allow a clearer view, easier upgrade and maintenance of the code.
3. Optimization of the proxy communication between the server and the remote service. This was not taken into consideration when implementing the proxy service, and was left as a future enhancement task.

4.2 Future Services and Evaluation

A specific design feature of JOBAD is its extensibility. Integrating new services for active documents in the JOBAD interactive document can be done with ease, and will not require much additional JavaScript code, as it was concluded in the evaluation (see 3). Apart from a list of the new services that will be implemented, also a short section of how future evaluation, testing and integration is to be conducted are provided in the two subsections that follow.

4.2.1 Services to be Implemented

A list of the services to be approached in the near future is provided below.

- **Symbol Type Lookup:** A service that would return the type of the selected symbol. Implementation is expected to be the same as the one used for definition lookup service.
- **Definition Expansion:** A service that would replace the selected presentation MathML of a symbol with its definition and re-render the mathematical expression, paying attention on the presentation context used, when multiple are present. The idea is to keep the original expression as an `maction` alternative. This service would be limited to simple, pattern-based and induction definitions (if applied step-by-step), i.e. will not be offered for implicit definitions.(See [38, chapter 15.2] for definition types supported by OMDoc.)
- **Notation Selection:** Every rendered symbol can be annotated with a reference to the notation definition in the backend that was used for its rendering. This can be used to ask the backend for alternative notations, offer the user the alternative notations and re-render the mathematical expression according to the notation selected by the user.
- **Guided Tour:** This service would be in a way extension of the definition lookup service. The goal is to generate a linear tutorial that would contain the explanation for each symbol that occurs in the explanations, until a foundational theory is reached.
- **Saving:** After a user has modified the interactive document, a service would be provided to allow him/her to upload the current configuration to the database.
- **Logging:** A web-service that would log all the interactions the user makes with the other services accessible from the document. Can be used for both debugging and user evaluation purposes.
- **Links to web resources:** The OpenMath Wiki [39] provides both definitions and discussions for symbols. Furthermore, its architecture allows for linking the symbol to further web-resources as for instance a Wikipedia article about mathematical concepts, which can be made available in a document.
- **Adaptive Display of statement-level structures:** On the level of definitions, theorems, and proofs a different kind of parallel markup from the OMDoc sources is generated, namely XHTML+RDFa [40]. This was already used for visualization

of rhetorical structures within a mathematical document [41] and the idea is to extend it to structured proofs.

- **Expansion of Algebraic Expressions:** This service would allow a user to select a formula from within the document and expand it according to his/her preferences. A simple example would be the expression: $(a+b)^3$ which can be expanded either as $a^3 + 3a^2b + 3ab^2 + b^3$ or using the binomial coefficients.

4.2.2 Future Steps in Evaluation and Integration

As a step further in evaluation a large OMDoc document is to be loaded on the server, and the implemented services will be enabled for testing. The idea is to use the complete lecture notes of a first-year undergraduate computer science course. The notes were initially maintained in LaTeXwith semantic annotations, which can be automatically converted to OMDoc [42].The annotations in the source documents include content-markup formulae, informal definitions of symbols, and notation definitions. The OMDoc representation can then be rendered into the JOBAD format, in order to be viewed using the JOBAD client and its offered services. Eventually this can be further tested during the fall semester lecture when one group of the computer science students will work with the static XHTML version of the lecture notes and a second group would use the JOBAD-enriched active document.

Furthermore, there is the plan of integrating the JOBAD architecture into the various integrated document management systems, such as the semantic wiki SWiM [43], and the panta rhei document browser and community tool [44].

5

Conclusion

The concept of an active document becomes more and more popular idea in today's world. Almost everyone that uses documents agrees that it is an idea with great potential, which is expected to be abundantly used in the future. In this paper, a vision of active mathematical documents was briefly explained and a powerful new architecture, that enables them, was introduced. The JOBAD architecture provides means for developing mashups of mathematical services within a document. While some of the services offered are based solely on the data already embedded in the document itself, most of the integrated web-services depend on the asynchronous client/server interaction and database information retrieval. The JOBAD server provides some core services such as document rendering and serving, providing the available services to be uploaded in the active document, and a simple proxy mechanism that allows other remote web-services to be used as services called from the JOBAD document. We have presented the definition lookup and unit conversion services with short explanation about their implementation and features. The unit conversion proved to be a successful test of the JOBAD architecture and its extensibility. It showed that the integration of future services within the current framework will not require much work from the developer's side, as most of the needed features are already provided by the already implemented services. The feedback we got from the users was more than positive, and offered us further insights of how far this concept of integrating web-services within active mathematical documents can reach. Furhtermore their feedback provided us with additional ideas for services that can be implemented in the future with a wide acceptance of the public.

Appendix A

JOBAD Code Documentation

This appendix contains the code documentation for the JOBAD source code, which can be found at https://svn.omdoc.org/repos/jomdoc/src/prj/jobad.

A.1 JOBAD Client Side Documentation

A.1.1 jobad.js

Jobad.js represents the master JavaScript user interface file. It contains the following global fields:

- **inContextMenu:** boolean variable that is set to true whenever the user clicks somewhere in the context menu, and to false in all the other cases. It is initially set to false.
- **selection:** boolean variable that is set to true whenever the user is registered to have made a selection. Its value is initially set to false.
- **focus:** object variable that points to the element, that is currently worked on. It is initially set to null.
- **selected:** object variable that points to the element that was originally selected by the user. It is initially set to null.
- **no-context:** boolean variable that is initially set to false. When set to true it disables the original browser context-menu.
- **xpos:** integer variable that keeps the value of the x-coordinate of the position of the cursor when an event is registered. It is initially set to 0.

- **ypos:** integer variable that keeps the value of the y-coordinate of the position of the cursor when an event is registered. It is initially set to 0.

In addition to the previously listed fields, jobad.js also contains the following functions:

- **mathmlClick(event):** function that handles the events that happen on MathML elements in the document. Its main feature is to disable the document's original context-menu, and replace it with the dynamically populated JOBAD contextmenu offering the available services.
- **otherClick(event):** function that handles the events that do not happen on MathML elements in the document. Its main feature is to make sure the context menu doesn't close when the event is registered to happen in it, so that the corresponding function is executed. In all other cases, the JOBAD contextmenu is closed and the document's original context menu is restored.
- **mathmlInfo(event):** function that displays some informative content, currently the title of the MathML element, whenever the cursor is registered to hover over a MathML element.
- **selectionParentTag():** function that handles the selection the user made, by finding the smallest element container that contains all the elements that belong to the selection.
- **mathmlSelection(event):** function that only registers that the user has made a selection of a MathML expression, by setting the value of the global `selection` field to true.
- **init():** function that loads all the modules in the document. It is also responsible of registering all the events that happen in the document including hovering over, clicking or making a selection on a MathML element or expression.
- **include(file):** function that is used to include a file in the document. Used by the **init()** function when loading the modules.

A.1.2 JOBAD modules

In this section each of the available JOBAD JavaScript modules is discussed in details, giving a short explanation for each of the fields and functions they contain.

Module: actions.js

A class definition for the `actionElement` element. An instance of the `actionElement` class has the following private fields:

- **label:** denoting the label that will be displayed on the user interface elements
- **info:** denoting the information title that is displayed when the cursor is hold above the link that would trigger this action element
- **action:** denoting the function that will be invoked if the action element is chosen by the user
- **id:** action instance identification

The constructor and the parameters it requires for creating an instance of the `action` class is the following:

```
var actionEl = new actionElement(newID, newLabel, newInfo, newAction);
```

As the class fields are private, the following getter and setter methods are necessary for setting and retrieving information from the `action` element:

- **getID():** returns the value of the `id` field of the current action element
- **getLabel():** returns the value of the `label` field of the current action element
- **getInfo():** returns the value of the `info` field of the current action element
- **getAction():** returns the value of the `action` field of the current action element
- **setID(newID):** sets the value of the `id` field to newID.
- **setLabel(newLable):** sets the value of the `label` field to newField.
- **setInfo(newInfo):** sets the value of the `info` field to newInfo.
- **setAction(newAction):** sets the value of the `action` field to newAction.

Module: contextmenu.js

A singleton class definition for the `contextmenu` element. The contextmenu has the following fields:

- **entries**: global list of instances of the `actionElement` class.

It also offers the following field manipulation methods:

- **add(actionElement):** pushes the `actionElement` received as a parameter, at the end of the `entries` list
- **doIt(index):** executes the action stored in the `actionElement` element with `index` value equal to the value of the index parameter.
- **removeActions(entry, index):** removes all `actionElement` elements from the content of the contextmenu
- **clear():** deletes all `actionElement` elements, and re-sets the `entries` list to an empty list.
- **empty():** sets the `entries` list to an empty list.
- **printActions():** prints the labels of the `actionElement` elements that are in the `entries` list
- **display():** displays the context menu

In addition to the class methods, the following contextmenu manipulation functions are available:

- **disableContextMenuItem(itemID):** function that disables contextmenu items on the fly. It makes the item with ID value equal to itemID parameter value, visible but unclickable.
- **enableContextMenuItem(itemID):** function that enables contextmenu items on the fly. It makes the item with ID value equal to itemID parameter value, clickable.
- **prepareContextMenu():** this is the main function in this module, that actually prepares the contextmenu once called by the user. This function first checks if a contextmenu already exists, and if yes it returns. Otherwise, it creates all the necessary `actionElement` elements, and dynamically populates and displays the contextmenu. It then checks if the conditions for the offered services are fulfilled and disables or enables the corresponding items accordingly.

Module: htmlsupport.js

This module offers the basic html utility functions that are needed by the functions from the other JOBAD modules.

- **addElement(elementID, newElement, idAttrVal, titleAttrVal, onMouseHover, htmlContent):** function that creates a new HTML element with the `@id` set to the value of the elementID parameter, tagname set to the value of the newElement parameter, the `@title` set to the value of the titleAttrVal parameter, the `@onmouseover` set to the value of the onMouseOver parameter, with contents equal to the value of the htmlContent parameter.
- **removeElement(elementID):** function that removes the element with `@id` equal to the value of the elementID parameter

Module: mathmlutility.js

The mathmlutility module has two constant fields referring to the namespaces used:

- **mathMLNamespace**
- **openMathNamespace**

It provides the following utility functions that are needed by the functions from the other JOBAD modules:

- **wrapMathMLElement(wrapperTagName):** function that takes the element in question, passed via the global reference object `focus` and wraps it with a MathML element whose tag name is passed as a function argument.
- **appendMathMLChild(childTagName, childContent):** function that appends a mathML child node to the element in question, accessed via the global reference object `focus`. The tag name and the contents of the child node are set according to the values of the function arguments childTagName and childContent.
- **getFirstMrowOrEquivalent():** function that returns the first ancestor of the element, that belongs to the list of valid MathML encapsulating elements: `mrow`, `msqrt`, `maction`, `mfrac`, `mfenced`, `mroot` and `math`.

- **createMactionElement(content, actionType):** function that dynamically creates an `maction` element, using the element pointed by the global reference object `focus` as one alternative, and the value of the **content** parameter as the second alternative. The `@action` attribute is determined by the function **actionType** argument and the `@selection` attribute is by default set to 2, i.e. points to the new alternative.

- **mactionOnDemand():** function that creates an `maction` element by demand of the user, by calling the **createMactionElement()** function with **content** argument set to "..." and the **actionType** argument set to "folding".

- **validMathMLTagNameCheck(objectTagName):** function that returns a boolean set to true if the value of the **objectTagName** is one of the following list of valid MathML encapsulating elements: `mrow`, `msqrt`, `maction`, `mfrac`, `mfenced`, `mroot` and `math`, and false otherwise.

- **checkMathMLEncapsulation(ancestorTagName):** function that returns a boolean set to true if the element pointed by the global `focus` object is encapsulated by an ancestor with tag name equal to the value of the **ancestorTagName** function argument, and false otherwise.

- **createUnitConvOpenMathElement(contents):** function that creates an OpenMath element needed for the unit conversion. It gets the content of the element from the value of its **contents** function argument. The content is then parsed in order to retrieve only the relevant data and the appropriate content elements are created.

Module: parallelmarkupsupport.js

The parallelmarkupsupport module offers the utility functions that retrieve the content elements given the corresponding presentation MathML elements.

- **getContentElement(element):** function that returns the corresponding content element given a specific presentation MathML element. The querying is currently done using the functions from jQuery, browsing the document looking for an element with `@id` attribute value equal to the value of the `@href` attribute value of the presentation MathML element. This however does not work with OpenMath elements and therefore this function will be re-written to give the same functionality but implemented using `XPath`.

- **getCommonAncestor(elem1, elem2):** function that takes as arguments two presentation MathML elements, selects the range of all elements between them and returns their common presentation MathML ancestor.
- **getContentRange(elem1, elem2):** function that takes as arguments two presentation MathML elements, gets their common ancestor using the **getCommonAncestor()** function and then retrieves its corresponding content element using the **getContentElement()** function.
- **getParallelName(element):** function that takes as an argument a presentation MathML element, gets its corresponding content element using the **getContentElement()** function and returns the value of the `@name` attribute.

Module: AJAXsupport.js

This module consists of functions that provide opportunity for asynchronous communication between the client side modules and JOBAD server side.

- **HTTPconnection(paramList):** a function that takes as an argument an array of parameters. The parameters order is important and should be as following: param[0] - xhtml element that would be used be dynamically populated by the response handling function; param[1] - the `@id` attribute of the xhtml element received as param[0]; param[2] - the JOBAD server URL address or any other address from the same domain that should be contacted in order to establish the connection; param[3] - list of (parameter, value) pairs; param[4] - callback function that handles the AJAX response. The function, establishes the connection to the URL address specified in param[2] and sends the parameters using the GET method. It eventually redirects the server response to the callback function received as param[4].
- **HTTP-POST-connection(paramList):** a function that takes as an argument an array of parameters. The parameters order is important and should be as following: param[0] - xhtml element that would be used be dynamically populated by the response handling function; param[1] - the `@id` attribute of the xhtml element received as param[0]; param[2] - the JOBAD server URL address or any other address from the same domain that should be contacted in order to establish the connection; param[3] - list of (parameter, value) pairs; param[4] - callback function that handles the AJAX response. The function, establishes the connection to the URL address specified in param[2] and sends the parameters using the

POST method. It eventually redirects the server response to the callback function received as param[4].

Module: foldingsupport.js

A module that handles the folding service actions. It consists of the following functions:

- **foldingSupportCheck():** function that checks whether one element is encapsulated with an `maction` element, by calling the **checkMathMLEncapsulation()** function with the parameter set to "maction".
- **mrowOrEquivalentCheck():** function that checks whether one element is encapsulated with any valid MathML encapsulating object, by calling the **checkMathMLEncapsulation()** function for each of the following elements: `mrow`, `msqrt`, `maction`, `mfrac`, `mfenced`, `mroot` and `math`.
- **foldIn():** function that folds the user's selected mathematical expression, by changing the value of the `@selection` attribute.
- **foldOut():** function that unfolds the user's selected mathematical expression, by changing the value of the `@selection` attribute.

Module: elisionsupport.js

A module that handles the elision service actions. It has two global fields:

- **visibleL**
- **visibleG**

both of which are initialized to zero with each loading of the module.

Furthermore the elisionsupport module contains the following functions:

- **elideB(visibilityLevel, visible):** function that performs bracket elision. It takes as parameters the visibility level chosen by the user - **visibilityLevel** and the current bracket visibility level of the mathematical expression - **visible**, and based on their values it either keeps the bracket at its place or hides it.

- **elideG(visibilityLevel, visible):** function that performs group elision. It takes as parameters the visibility level chosen by the user - **visibilityLevel** and the current group visibility level of the mathematical expression - **visible**, and based on their values it either keeps the group visible or hides it.
- **elideGroups()**: function that first finds all the group elements by invoking the **getAllGroupElements()** function, and then for each of them performs the elision by calling the **elideG()** function. Eventually it sets the value of the global field `visibleG` to the value of the visibility level chosen by the user.
- **elideAll()**: function that first finds all the bracket semantic elements by invoking the **getAllELevelSemanticsElements()** function, and then for each of them performs the elision by calling the **elideL()** function. Eventually is sets the value of the global field `visibleL` to the value of the visibility level chosen by the user.
- **getAllAttributeLessSemanticsElements():** function that returns a list of all the semantics elements in the document that do not have any attribute connected to them.
- **getAllELevelSemanticsElements():** function that returns a list of all semantics elements in the document that have the `@mcd:elevel` attribute.
- **getAllGroupElements():** function that returns a list of all semantics elements in the document that have the `@mcd:egroup` attribute.
- **swap():** function that just swaps the position of two sibling elements.

Note that this module will be redone, and the next implementation will make use of the `maction` elements, that would make the **swap()** function redundant and would require change in the code of the **elideL()** and **elideG()** functions.

Module: definitionLookup.js

A module that provides the definition lookup service (see 2.4.2). It contains one global constant variable:

- **ServingOMDoc** : denotes the URL address of the JOBAD server to be contacted for the definition lookup service

In addition it also provides the following functions:

- **populateDialog(where, innerHTML):** function that initially makes sure the tooltip dialog is cleared from any previous content by calling the **clearDialog()** function, and then fills it with the content from the `innerHTML` parameter value.
- **clearDialog():** function that removes the content from the tooltip dialog object.
- **definitionLookup():** function that finds the content element of the user-selected presentation MathML element using the parallelmarkupsupport module functions, and prepares its `@id` and `@cd` attribute values as (parameter, value) pairs to be sent to the AJAXsupport module functions. It also determines the callback function that would later handle the response returned from the JOBAD server.
- **hangleResponseDialog():** function implemented as a variable so that it can be passed on as a callback function by the **definitionLookup()** function to the AJAXsupport module functions. It receives the AJAX response from the communication with the JOBAD server, and populates the tooltip dialog with this content.

Module: unitconversion.js

A module that provides the unit conversion service (see 2.4.2). It consists of the following functions:

- **isConvertable():** function that analyzes the content element that corresponds to the user-selected element, and checks if it is declared as a unit. Returns true if it is convertible and false otherwise.
- **convert()**: function that currently simply calls the **convertTo()** function with a parameter value set to "mile".
- **convertTo():** function that extracts the value and the unit name from the content element corresponding to the user-selected presentation MathML element, and creates the URL to be passed to the remote online converter. Then it should pass this URL to the **HTTP-POST-connection()** function from the AJAXsupport module, that would contact the JOBAD proxy service, without fear of the length of the URL that is currently passed as a value to one of the GET method parameters. This however is still not yet implemented, instead it just invokes the **HTTPconnection()** function from the AJAXsupport module and performs the same action.

- **handleUnitConversion():** function that is implemented as a variable so that it can be passed on as a callback function by the **convertTo()** function to the AJAXsupport module functions. It receives the AJAX response from the communication with the remote online converter via the JOBAD proxy, and parses it with XPath in order to retrieve the relevant data from it. It then invokes the **createUnitConvOpenMathElement()** function with the `content` argument set to the result data extracted from the converter response. After obtaining the OpenMath content element, it sends it for rendering via the AJAXsupport module function.
- **handleUnitRendering():** function that is implemented as a variable so that it can be passed on as a callback function by the **handleUnitConversion()** function to the AJAXsupport module functions. It receives the AJAX response from the communication with the JOBAD server, and places the parallel markup content as an `maction` alternative to the originally user-selected presentation MathML element for conversion.

Module: othersupport.js

Just for sake of completeness, the othersupport module has functions that would let the user know when some of the services are not currently supported or are under development/maintenance.

Currently consists of only one function:

- **underConstruction():** function that alerts that the currently selected item is under construction.

A.2 JOBAD Server Side Documentation

A.2.1 ServingOMDoc.java

- **ServingOMDoc()** constructor of the ServingOMDoc servlet class that just initializes the class fields necessary for document querying and rendering.
- **serveJOMDocRendered()** function that gets the requested OMDoc document from the database, renders it using JOMDoc command line interface and prints the result back to the client. This is the method currently used for rendering documents, as according to the JOMDoc developers the command line interface is the most reliable one at the moment. Once the Java style invocation is stable, this function would be replaced by a version of the **JOMDocJavaStyle()** function.

- **servePlainOMDoc()** function that simply prints out the requested OMDoc file from the database in OMDoc format.
- **JOMDocJavaStyle()** function that renders a document in a Java style using the JOMDoc Java rendering methods.
- **URLconnection()** function that establishes the connection to the requested URL, and prints out the received response to back to the client.
- **unitConversion()** function that provides the current working implementation of the unit conversion service on the server side (to become overridden soon, by simplifying the design and implementation of the JOBAD proxy). It establishes connection to the URL of the online unit converter, builds the request string using the GET method using the values received from the client side, and calls the **URLconnection()** function.
- **renderElement()** function that renders the content element using the JOMDoc renderer.
- **serveDefinition()** function that currently only loads one OMDoc document that holds the definitions of the symbols involved, and parses it using XOM so that it retrieves the definition. Once the content element is obtained, it is rendered using the **renderElement()** function and the result is sent back to the client side.
- **doGet()** Java Servlet method handling the HTTP GET requests. Gets the `action` parameter from the request and proceeds according to its value, for instance if the value of the `action` parameter is "expandDefinition" then it proceeds by retrieving the rest of the request parameters such as the element id, and the content dictionary cd and calls the **serveDefinition()** function. [add some footnote that currently only the id is retrieved, but will be fixed soon]
- **doPost()** Java Servlet method handling the HTTP POST requests that just transfers the handling of the received data and generating the response to the **doGet()** method described earlier.

Bibliography

[1] Jana Giceva, Christoph Lange, and Florian Rabe. Web Services for Active Mathematical Documents. 2009.

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

[3] 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.

[4] 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.

[5] 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.

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

[7] C. Ullrich. *Pedagogically Founded Courseware Generation for Web-Based Learning*. LNCS. Springer, 2008. ISBN 978-3-540-88213-8.

[8] Monet. MONET. http://monet.nag.co.uk/mkm, seen March2005.

[9] 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.

[10] 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.

[11] Yahoo! Pipes. http://pipes.yahoo.com, 2009.

[12] Mozilla Labs. Ubiquity. http://ubiquity.mozilla.com, 2009.

[13] Mozilla Foundation Netscape Communications Corporation. Javascript. http://www.javascript.com, 2008.

[14] jQuery Team. jquery. http://jquery.com, 2009.

[15] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, and Jerome Simeon. Xpath2.0. http://www.w3.org/TR/2007/REC-xpath20-20070123, 2007.

[16] Apache Software Foundation. Apache tomcat. http://tomcat.apache.org, 2008.

[17] Sun Microsystems. Java servlet. http://java.sun.com/products/servlet/, 2009.

[18] JOMDoc. JOMDoc Project — Java Library for OMDoc documents. URL http://jomdoc.omdoc.org.

[19] Robert Miner Nico Poppelie David Carlisle, Patrick Ion. Mathml2.0. http://www.w3.org/TR/2003/REC-MathML2-20031021/, 2003.

[20] Robert Miner Nico Poppelie David Carlisle, Patrick Ion. Mathml2.0 presentation markup. http://www.w3.org/TR/MathML2/chapter3.html, 2003.

[21] Robert Miner Nico Poppelie David Carlisle, Patrick Ion. Mathml2.0 content markup. http://www.w3.org/TR/MathML2/chapter4.html, 2003.

[22] Robert Miner Nico Poppelie David Carlisle, Patrick Ion. Mathml2.0 parallel markup. http://www.w3.org/TR/MathML2/chapter5.html, 2003.

[23] OpenMath Society. Openmath. http://www.openmath.org, 2007.

[24] Michael Kohlhase. OMDoc: An open markup format for mathematical documents (latest released version). Specification, http://www.omdoc.org/pubs/spec.pdf.

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

[26] Javaline Developers. Ajax. http://www.ajax.org/, 2009.

[27] J. Stratford and J. H. Davenport. Unit knowledge management. In Autexier et al. [45].

[28] W3C. Mathematical Markup Language (MathML) 3.0 (Third Edition).

[29] Apache Software Foundation. Content negotiation. http://httpd.apache.org/docs/1.3/content-negotiation.html, 2008.

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

[31] 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.

[32] M. Kohlhase, C. Lange, C. Müller, N. Müller, and F. Rabe. Notations for active documents. KWARC Report 2009-1, Jacobs University, 2009. URL http://kwarc.info/publications/papers/KLMMR_NfAD.pdf.

[33] M. Kohlhase, C. Müller, and F. Rabe. Notations for Living Mathematical Documents. In Autexier et al. [45].

[34] Douglas Crockford. Jslint. http://www.jslint.com, 2009.

[35] Elliotte Rusty Harold. Xom. http://www.xom.nu/, 2005.

[36] Mozilla Development Center. Same origin policy for javascript. https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript, 2005.

[37] SCIEnce. The SCIEnce project – symbolic computation infrastructure for europe. http://www.symbolic-computation.org/, 2009.

[38] M. Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer, 2006.

[39] C. Lange. OpenMath wiki. http://wiki.openmath.org, 2009.

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

[41] 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.

[42] M. Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2008.

[43] C. Lange and A. González Palomo. Easily editing and browsing complex OpenMath markup with SWiM. In P. Libbrecht, editor, *Mathematical User Interfaces Workshop*, 2008. URL http://www.activemath.org/~paul/MathUI08.

[44] panta-rhei. The panta rhei Project. http://trac.kwarc.info/panta-rhei. seen March 2009.

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