Jacobs University Bremen

Guided Research Thesis Computer Science

Universal OpenMath Machine

Author: Vladimir Zamdzhiev
First Advisor: Prof. Dr. Michael Kohlhase
Second Advisor: Dr. Florian Rabe
Date: May 8, 2011

Abstract

Mathematical Knowledge Management aims to promoting better access to repositories of mathematical knowledge using software systems. OMDoc and OpenMath are mathematical content standards developed with this goal in mind. However, integrating computation into OMDoc is a challenge not originally addressed by these formats. In this work, we propose a software system that lays the foundations for this ambitious task. The Universal OpenMath Machine (UOM) is a software system that supports a client-server model that gives the client the ability to execute Scala implementations of OpenMath objects embedded in an OMDoc document and to simplify OpenMath expressions. We also describe in detail an application scenario with prospects in web-based education, where the user is viewing an OMDoc document using a web browser and then uses the UOM to interactively evaluate OpenMath expressions into simpler ones.

Contents

1 Introduction to Mathematical Knowledge Management

The amount of mathematical knowledge that we have today is enormous, certainly far more than any person could possibly understand in his lifetime. Moreover, mathematical knowledge is growing exponentially - it has been estimated that the total amount of published mathematics doubles every ten-fifteen years [Odl95]. Clearly, organizing this ever increasing both in size and in complexity knowledge is a very serious challenge and it is impossible to take advantage of it in its entirety without the help of computers. Computer software such as computer algebra systems, automated theorem provers, graphical simulation packages, etc. are already invaluable tools used by mathematicians. Naturally, mathematicians would also want to use computers in order to find relevant mathematical theorems not only from their own field, but from other fields of mathematics as well in a fast and efficient way. Mathematical Knowledge Management, lying at the intersection of mathematics and computer science, is a field concerned with the development of practices that would allow for better access to repositories of mathematical knowledge, visualising and understanding that knowledge and reusing and combining it to discover new knowledge.

2 Statement and Motivation of Research

OpenMath[BCC$^+$04] and OMDoc[Koh06] are mathematical content standards developed by researches in Mathematical Knowledge Management. They are designed to represent static mathematical data and as such there are inherent difficulties for them to address computation on data. However, computation has always been an important part of mathematics - one of the oldest algorithms, discovered more than two thousand years ago and still in use today is Euclid's algorithm for finding the greatest common divisor. With this work we try to alleviate this problem of integrating computation into OMDoc, by proposing a software system.

The software system we propose is called the Universal OpenMath Machine (UOM). The system is designed to execute algorithmic implementations which are embedded in an OMDoc document and are written for specific OpenMath objects that are part of the document. It allows readers of OMDoc documents to simplify or evaluate expressions, either part of the document or specified by the user, in which all objects have written implementations and thus provides for a more interactive experience when trying to understand mathematical theories. In principle, the system can be extended to support any computational language, but in this work we restrict ourselves to supporting only one. We decide to use an already existing programming language, instead of creating our own, as a first step towards making the system independent of the choice of a computational method.

Our programming language of choice is Scala[OSV07] as it supports the functional programming paradigm and as such it provides a convinient and intuitive way for mathematicians to write implementations for mathematical functions. Another consideration that led to this choice is that Scala is designed to express common programming patterns in a concise, elegant, and type-safe way. Moreover, Scala is a pure object-oriented language in the sense that every value is an object. Scala runs on the Java[GJJ96] Virtual Machine, can call existing Java libraries and is compatible with Java programs. Java programmers are likely to find the Scala programming language easy to use.

3 Related Work

The Kenzo system [DRSS99] is a computer program devoted to symbolic computation in Algebraic Topology. The system has been proven to be successful by discovering new results in Algebraic Topology that were previously unknown. [RER09] Researchers from the University of La Rioja have undertaken the task of popularising the program by making it more appealing to use. In [HPR08] and [HPR09], the authors present an architecture to dynamaically load new modules in a Graphical User Interface (GUI) for the Kenzo system by encoding information on the user interface specification and the code carrying the functionality of the GUI in the OMDoc and OpenMath formats. The authors are embedding pieces of program code into OMDoc documents by using the `<code>` tag and then dynamically create GUIs for specific documents using the code.

4 Preliminaries

In this section we present a very brief overview of the OpenMath and OMDoc standards.

4.1 OpenMath

OpenMath[BCC$^+$04] is a standard for representing mathematical data in as unambiguous a way as possible. It can be used to exchange mathematical objects between software packages or via email, or as a persistent data format in a database. It is tightly focussed on representing semantic information. Formally, an OpenMath object is a labelled tree whose leaves are the basic OpenMath objects integers, IEEE double precision floats, unicode strings, byte arrays, variables or symbols. OpenMath objects can be built up recursively in a number of ways. One of them is function application, another is attribution, which can be used to add additional information to an object without altering its foundational meaning. Another example is binding objects which are used to represent an expression containing bound variables.

4.2 OMDoc

Open Mathematical Documents (OMDoc)[Koh06] is an open markup language for mathematical documents. It is designed to provide a semantics-oriented infrastructure for the communication and storage of mathematical knowledge. The representation in OMDoc makes the document content unambiguous and its context transparent. OMDoc achieves this by employing document markup, specifically by embedding control codes into mathematical documents that identify the structure of the document, the meaning of text fragments, and their relation to other mathematical knowledge. The main design principles behind the OMDoc format are for it to be ontologically uncomitted, to combine formal and informal views of the mathematical knowledge in its documents, to be based on sound logic and representational principles and to be based on structural content markup. In addition, OMDoc documents have the capability to provide for semantic markup and therefore allow for documents that are benefitting from this to be mechanically processed by proof verification software, thereby certifying the corectness of the proofs. Thus, the OMDoc format provides representations that can easily be read by humans and representations that allow for an advanced level of machine support.

5 Description of the System

In this section we describe the Universal OpenMath Machine by first introducing the reader with a general overview and then providing a more detailed explanation of the inner workings of the system.

5.1 General Overview of the system

The Universal OpenMath Machine is designed to evaluate or simplify OpenMath expressions which are composed of OpenMath objects that are part of OMDoc documents and for which there exist written Scala algorithmic implementations that define how an OpenMath expression can be simplified into another OpenMath object. The UOM is deployed as a web service which users can easily access via a web browser or any other program that can send an HTTP[FGM+99] request. Users interact with the system in a straightforward manner by just passing an OpenMath object to it. Also, users can upload documents or even precompiled implementations to the system. Authors of documents or implementations have to follow a small set of simple rules to make their work exploitable by the UOM. The UOM consists of several components that interact together. The major steps that need to be performed by the UOM in order to respond to a request by a client are to recursively simplify an expression by searching for OpenMath objects with implementations, parsing relevant OMDoc documents and generating Scala source files from them, compiling the scala code and maintaining an implementation database cache and finally executing implementations when needed in order to simplify or evaluate OpenMath objects. The inner workings of the UOM are described in greater detail in subsection 5.2.

5.1.1 The UOM web service

The UOM provides a web service that can be accessed by any user who is running a web browser on his computer. This makes the service available to a very wide spectrum of devices and especially to unpowerful computers - from low-end PCs to smartphones and tablets that are becoming increasingly more popular. Users of the service can also benefit from the client-server model in several ways.

One of them is that users are not required to have a Scala compiler or interpreter installed, which makes the service accessible also to users that cannot program in Scala or any language at all, and it also reduces the computational load on the user's computer. The UOM service can be deployed on powerful servers which can in turn reduce the time needed to produce a response to the request compared to the time it would take should the service be running locally on the user's computer. Moreover, the UOM server can dynamically keep an implementation database cache, a database where implementations of OpenMath objects are precompiled and ready to use, which would produce faster response times and makes the client-server model even more beneficial.

Another advantage of the client-server model is that users will not be required to deploy any additional software on their machines and will not be bothered with any support, maintenance or system administrative tasks involved with running the software. In addition, users will be safe from any security or privacy vulnerabilities that might arise from executing malicious third-party code embedded in an OMDoc document. Clearly, optimising performance and coping with potential security issues requires a degree of knowledge or effort which some users do not have or are unwilling to make, which is another reason to employ the client-server model.

5.1.2 User interaction

All that is required from the user in order to utilize the UOM is to send an HTTP request which encodes an OpenMath object. Every atomic subterm of the object refers to an OMDoc constant or other element that has a unique URI [BLFM05], associated with it. This is all of the information required by the UOM in order to produce a response. In the current version of the system, the user would have to upload documents or implementations to the system if there are OpenMath terms whose URIs refer to documents that are not on the server, however in future versions of the service, support for automatically retrieving documents based on the URIs of the OpenMath objects can be implemented.

A universal way to access the UOM from any device would be to use an online web interface that provides the necessary features. Such a feature could be browsing through a catalogue of OMDoc documents and having the ability to select theories or constants within it. Another would be to have a tool that allows constructing user objects from already existing OpenMath objects in other documents.

Once the request has been made on behalf of the user the response time depends on several factors. Excluding external factors to the software (e.g. server load, network latency, etc.), there are two main performance bottlenecks of the system. The first one is the compilation step which can be avoided if the required implementations are already registered in the database. That would be the case for recently accessed documents or for documents that are popular among the users. The second performance hurdle could potentially come from the execution of the implementation. Poorly written implementations or implementations of functions that are inherently complex might take considerable time to execute. However, there would be many use cases where the response time would be comparable to that of a website. One such use case would be responding to a request for which all of the implementations are in the database (for instance the document is stored on the same machine as the UOM) and the complexity of the implementations is negligable.

5.1.3 Document Requirements

Special requirements when creating an OMDoc document apply only to constants that are to have Scala implementations written for them. Implementations are allowed only for elements at the object level that are represented as OpenMath objects. The Scala implementations should be enclosed in `<OMFOREIGN>` tags. The implementation for a single OpenMath object does not have to be self-contained in the sense that it can be compiled by the Scala compiler on its own. The implementation can implicitly rely on other OpenMath objects which are referenced in the same or different OMDoc documents. Each implementation should use the MMT API[Rab08] data structure which represents OpenMath objects - `Term` - in the sense that all of the input parameters are `Term` and also the return value of the implementation is `Term` as well.

Authors should know that OMDoc theories are translated to Scala classes. The full binary name of the class is formed from the URI of the theory which removes the possibility for naming conflicts. Constants from theories are translated to member fields of the corresponding class. This provides a convenient way for users to refer to any OMDoc constant when writing implementations. In addition, authors should use theory inclusions when their implementations are referencing OpenMath objects from different documents or theories.

5.2 Implementation Details

In this section we describe the important computational steps that the UOM performs in order to evaluate the request of the user and also describe the different components of the system. The system has three main components - the extractor, the implementation database and the main controller.

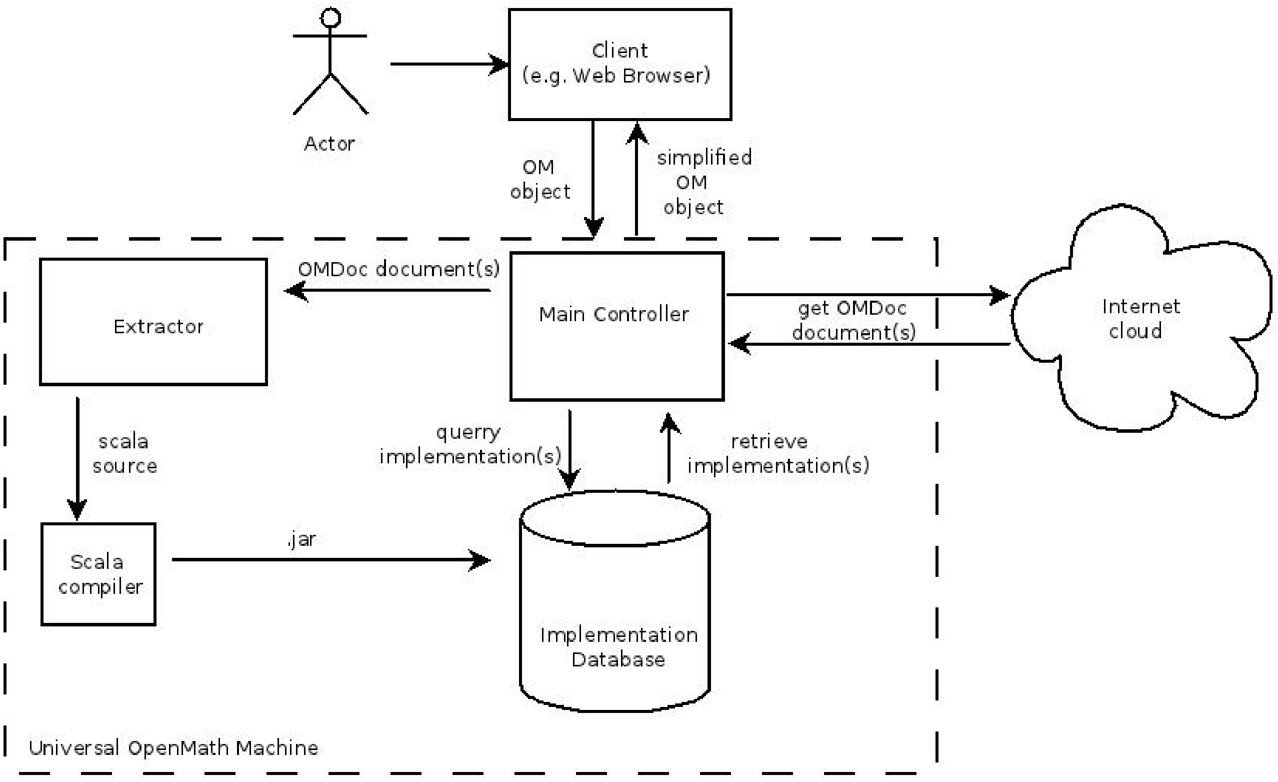

Figure 1: System framework

5.2.1 Extractor

The extractor is the component of the system whose task is to generate Scala files from OMDoc documents. Given an OMDoc document, the extractor parses it and translates the different theories found in the document to Scala classes. In order to avoid naming conflicts, the extractor creates unique binary names for every class that corresponds to an OMDoc theory. Each class is written as part of a package, whose name is derived from the URI of the document in the traditional way. Therefore, all OMDoc theories in a single document will be translated to classes with the same package name. The name of the class inside the package is then just the name of the corresponding OMDoc theory.

Given an OMDoc theory, the extractor will translate the OMDoc constants that do not have an implementation to member value fields of the class with the same name as the constants. Constants with implementation are written to the Scala source file as methods. The implementation is taken verbatim from the document and the author should write it exactly as it would appear in a Scala class, including the **def** keyword, signature, return type, etc. This allows for implementations that take variable number of OpenMath terms as input parameters. In addition, the extractor generates an auxiliary method and member field value in order to provide a universal way of accessing implementations from the main controller using reflection.

Lastly, the extractor also handles theory inclusions by first checking if the theory to be included has been translated. If it is not, then the extractor retrieves it and translates it.

5.2.2 Implementation Database

The implementation database is used by the UOM to store precompiled implementations that are associated with an OpenMath object. The benefit that the implementation database brings to the UOM is faster response times for implementations that are stored by avoiding the compilation, which is otherwise necessary and computationally expensive. Currently, the database is stored entirely in main memory and is simply a Scala HashMap that associates the unique names of OpenMath objects to implementations. In future versions of the system, the database can be improved by making it persistent and storing it on the file system, but for our current purposes, it is enough to store it in memory as a proof-of-concept. Updating the database is done by providing a jar file which contains a collection of class files. The UOM uses Java reflection on the class files and stores the implementations into the HashMap making them ready to use on demand.

5.2.3 Main Controller

The main controller performs several functions. It interacts with the operating system in order to provide some of the needed functionality (e.g. call the scala compiler, the jar utility etc.). It also controls the general flow of the program by invoking methods in the other components. The main controller also receives the request of the client and identifies subterms, within the OpenMath term to be evaluated, for which implementations need to be obtained. The main controller first consults the database to check if there are any implementations that are already available. For the rest, if any, the controller will, in future versions of the system, retrieve the documents that are required and pass them to the extractor. In the current version, the user is required to upload his document or precompiled implementation to the server if the implementation or document are not stored there. Once the extractor has completed its work, the main controller compiles the scala Source files using the Scala compiler, and creates a jar file from the collection. The jar file is then passed to the implementation database and all the extracted implementations are registered with it. Finally, a response can be produced to the client, by recursively simplifying the term in the request using the implementations in the database. The interactions are illustrated in figure 1.

6 Application Scenarios

We describe two application scenarios of the Universal OpenMath Machine. In section 6.1 we present an example of how the UOM can be used for web-based education and we also illustrate the different computational steps by using an example OMDoc document. In 6.2 we present a use case for the UOM in unit conversion.

6.1 Web-based education

One of the most promising application areas of the Universal OpenMath Machine is in web-based education. The system would allow a user to view OMDoc documents and evaluate expressions of interest using a web browser on a low-end computer that has very limited software installed, in particular lacking the capability of executing Scala code. The system is designed to perform the computationally intensive tasks on the server, so the client is not required to run on a powerful computer. Students and other readers learning about mathematical theories can benefit from the system as it will provide a more interactive learning experience by allowing students to evaluate and simplify relevant mathematical objects from those theories. The reader will also be able to evaluate examples that were created by himself.

We present an example that illustrates the general workings of the system. For this scenario, we have selected a short document which nonetheless captures the essential problems that the system will have to overcome, namely implementations using other implementations of OpenMath objects defined in different theories. The user is viewing the document from a web browser and would like to evaluate an expression that he has created. The document is displayed below.

```
<omdoc xmlns="http://omdoc.org/ns" base="http://cds.omdoc.org/unsorted/uom.omdoc">
   <theory name="lists">
     <constant name="elem"/>
     <constant name="list"/>
     <constant name="nil"/>
     <constant name="cons"/>
     <constant name="append">
       <definition>
         <OMOBJ>
           <OMFOREIGN>
             def append(l: Term, m: Term) : Term = {
               l match {
                 case this.nil => m;
                 case OMA(this.cons, List(this.elem, rest)) =>
                    OMA(this.cons, List(this.elem,  append(rest, m)) );
                 case _ => throw new Exception("malformed term");
               }
             }
           </OMFOREIGN>
         </OMOBJ>
```

```
      </definition>
    </constant>
  </theory>

  <theory name="lists_ext">
    <include from="?lists"/>
    <constant name="append_many">
      <definition>
        <OMOBJ>
          <OMFOREIGN>
            def append_many(l: Term*) : Term = {
              val lists = new org.omdoc.cds.unsorted.uom.omdoc.lists;
              l.toList match {
                case Nil => lists.nil
                case hd :: tl => lists.append(hd, append_many(tl : _*))
              }
            }
          </OMFOREIGN>
        </OMOBJ>
      </definition>
    </constant>
  </theory>
</omdoc>
```

The theory *lists* introduces the basic components needed to represent the list data structure which can be found in LISP and many functional programming languages. In addition, using the basic objects *elem, list, nil, cons*, we can define the function *append* which given two lists returns the list formed by appending the second one to the first list. The second theory - *lists_ ext* - introduces a new fucntion called *append_ many*. Given any number of lists as parameters, the function should return the list formed by appending all of the lists consecutively given the order. The implementation written for *append_ many* relies on *append* and uses the objects from the first theory - *lists*. Suppose the user would like to evaluate the expression from the above theory which represents the list

***append_ many*([1,2,3], [4,5], [6,7])**

The browser sends a request to the UOM server specifying the expression of interest. The UOM first consults the implementation database and then, it identifies the OMDoc document which is reffered by the OpenMath objects and the extractor generates the following Scala source file :

```
// Source file generated by the Universal OpenMath Machine

import info.kwarc.mmt.api._
import info.kwarc.mmt.api.objects._
import info.kwarc.mmt.uom.Implementation

package org.omdoc.cds.unsorted.uom.omdoc
{
class lists {
  val base = DPath(new utils.xml.URI("http://cds.omdoc.org/unsorted/uom.omdoc"))

  val elem = OMID(base ? "lists" ? "elem")
  val list = OMID(base ? "lists" ? "list")
  val nil = OMID(base ? "lists" ? "nil")
  val cons = OMID(base ? "lists" ? "cons")

  // UOM start http://cds.omdoc.org/unsorted/uom.omdoc?lists?append
  def append(l: Term, m: Term) : Term = {
    l match {
      case this.nil => m;
      case OMA(this.cons, List(this.elem, rest)) =>
        OMA(this.cons, List(this.elem, append(rest, m)) );
      case _ => throw new Exception("malformed term");
    }
  }
  // UOM end

  def append_*(l : Term*) : Term  = {
    return append(l(0), l(1))
  }

  val append_** = new Implementation(
    base ? "lists" ? "append"
    ,
    append_*
    )

}
}

package org.omdoc.cds.unsorted.uom.omdoc
{
class lists_ext {
  val base = DPath(new utils.xml.URI("http://cds.omdoc.org/unsorted/uom.omdoc"))

  // UOM start http://cds.omdoc.org/unsorted/uom.omdoc?lists_ext?append_many
  def append_many(l: Term*) : Term = {
    val lists = new org.omdoc.cds.unsorted.uom.omdoc.lists;
    l.toList match {
      case Nil => lists.nil
      case hd :: tl => lists.append(hd, append_many(tl : _*))
    }
  }
  // UOM end

  def append_many_*(l : Term*) : Term  = {
    return append_many(l : _*)
  }

  val append_many_** = new Implementation(
    base ? "lists_ext" ? "append_many"
    ,
    append_many_*
    )

}
}
```

Once the source file has been created, the UOM compiles it, produces a jar file which is used by the implementation database to register the implementations for the OpenMath objects that need to be evaluated. Then, the UOM simplifies the expression using the implementations and the result of the execution is the OpenMath expression which is equivalent to the list **[1,2,3,4,5,6,7]**. The result is then sent back to the client and the browser can render it on the screen.

6.2 Unit Conversion

In [Cîr11], the author proposes a framework for unit conversion in technical documents. The framework encompasses many different systems, technologies and formats in order to enable interactive unit conversion to the reader. As part of the framework, the author uses the UOM web service for the arithmetic computations that are required in order to present correct unit conversions to the reader. In the stage where the UOM is used, the Universal OpenMath Machine receives an OMDoc document containing unit quantities that need to be simplified or evaluated arithmetically. For that purpose, the UOM is preloaded with the implementations of the OpenMath symbols used to denote the common arithmetic operations of addition, subtraction, division and multiplication which are used throughout the framework. After performing the arithmetic evaluations, the UOM web service returns the simplified OMDoc document, which is used by the framework to eventually produce Presentation MathMl[ABC+03] + OpenMath.

7 Conclusion and Future Work

We have described a software system that can act as a foundation towards the ambitious goal of integrating computation into the OMDoc format. The system allows users to simplify or evaluate OpenMath expressions that refer to object level elements in OMDoc documents. In addition, we presented an application scenario in web-based education that highlighted how the system can bring a more interactive experience to readers. The system has also been proven to be useful for unit conversion.

In its current state, the UOM supports only Scala, but it can be further improved by allowing support for additional programming languages. This will bring more versatility to the system and will make it more attractive to a wider group of authors. Finally, the system can be improved by addressing security and privacy considerations that come with running third-party code.

References

[ABC$^+$03] R. Ausbrooks, S. Buswell, D. Carlisle, S. Dalmas, S. Devitt, A. Diaz, M. Froumentin, R. Hunter, P. Ion, M. Kohlhase, R. Miner, N. Poppelier, B. Smith, N. Soiffer, R. Sutor, and S. Watt. Mathematical Markup Language (MathML) Version 2.0 (second edition). Technical report, World Wide Web Consortium, 2003. See `http://www.w3.org/TR/MathML2`.

[BCC$^+$04] S. Buswell, O. Caprotti, D. Carlisle, M. Dewar, M. Gaetano, and M. Kohlhase. The Open Math Standard, Version 2.0. Technical report, The Open Math Society, 2004. See `http://www.openmath.org/standard/om20`.

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force, 2005.

[Cîr11] Mihai Cîrlănaru. Authoring, publishing and interacting with units and quantities in technical documents, 2011.

[DRSS99] X. Dousson, J. Ruibo, F. Sergeraert, and Y. Siret. The Kenzo Program. Technical report, Institute Fourier, Grenoble, 1999. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo`.

[FGM$^+$99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. *RFC 2616, Internet Engineering Task Force (IETF)*, 1999.

[GJJ96] J. Gosling, W. Joy, and G. Steele Jr. *The Java Language Specification.* Addison-Wesley, 1996.

[HPR08] J. Heras, V. Pascual, and J. Rubio. Mediated Access to Symbolic Computation Systems. In *Mathematical Knowldege Management*, Lecture Notes in Artificial Intelligence, pages 446–461, 2008.

[HPR09] J. Heras, V. Pascual, and J. Rubio. Using Open Mathematical Documents to interface Computer Algebra and Proof Assistant systems. In *Mathematical Knowldege Management*, Lecture Notes in Artificial Intelligence, 2009.

[Koh06] M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Number 4180 in Lecture Notes in Artificial Intelligence. Springer, 2006.

[Odl95] A. M. Odlyzko. Tragic loss or good riddance? the impending demise of traditional scholary journals. *International Journal of Human-Computer Studies*, 42:71–122, 1995.

[OSV07] M. Odersky, L. Spoon, and B. Venners. *Programming in Scala*. artima, 2007.

[Rab08] F. Rabe. The MMT System, 2008. See `https://trac.kwarc.info/MMT/`.

[RER09] A. Romero, G. Ellis, and J. Rubio. Interoperating between Computer Algebra systems : computing homology of groups with Kenzo and GAP. In *International Symposium on Algorithms and Computation*, 2009.