# OMDoc: Towards an Internet Standard for the Administration, Distribution and Teaching of mathematical Knowledge

Michael Kohlhase

FB Informatik, Universität des Saarlandes, Saarbrücken
http://www.ags.uni-sb.de/~kohlhase

**Abstract.** In this paper we present an extension OMDoc to the OpenMath standard that allows to represent the semantics and structure of various kinds of mathematical documents, including articles, textbooks, interactive books, courses. It can serve as the content language for agent communication of mathematical services on a mathematical software bus.

## 1 Introduction

It is plausible to expect that the way we do (conceive, develop, communicate about, and publish) mathematics will change considerably in the next ten years. The Internet plays an ever-increasing role in our everyday life, and most of the mathematical activities will be supported by mathematical software systems (we will call them *mathematical services*) connected by a commonly accepted distribution architecture, which we will call the *mathematical software bus*. We have argued for the need of such an architecture in [SHS98,FHJ+99], and we have in the meantime gained experiences with the MathWeb system that provides a general distribution architecture (see [FK99b]); other groups have conducted similar experiments [DCN+00,AZ00] based on other implementation technologies, but with the same vision of creating a world wide web of cooperating mathematical services. In order to avoid fragmentation, double inventions and to foster ease of access it is necessary to define interface standards for MathWeb[1]. In [FHJ+99], we have already proposed a protocol based on the agent communication language Kqml [FF94] and the emerging Internet standard OpenMath [AvLS96,CC98] as a content language (see Fig. 1). This layered architecture which refines the unspecific "application layer" of the OSI protocol stack is inspired by the results from agent-oriented programming [Sho90], and is based on the intuition, that all agents (not only mathematical services) should understand the agent communication language, even if they do not understand the content language, which is used to transport the actual mathematical content. The agent communication language is used to establish agent identity,

[1] We will for the purposes of this paper subsume all of the implementations by the term MathWeb, since the communication protocols presented in this paper will make the constructions of bridges between the particular implementation simple, so that that the combined systems appear to the outside as one homogenous web.

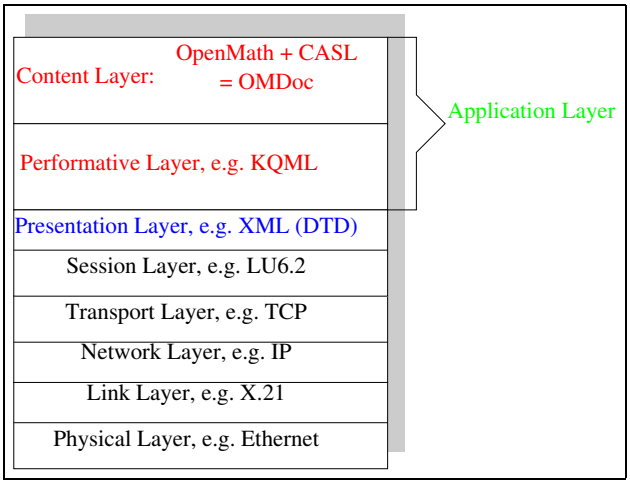


**Fig. 1.** Artificial Communication: KQML and the OSI Reference Model

reference and – in general – model the communication protocols (see [AK00] for details in the case of mathematical services). Thus we can concentrate on the content language in this paper.

The experience with MATHWEB in general, and with the ΩMEGA system – a mathematical assistant system based on several MATHWEB services (see [BCF+97]) – in particular have shown that it is not sufficient to be able to communicate *mathematical objects*object, but also *mathematical knowledge*knowledge!mathematical in general. Support for the communication of mathematical objects is already provided by OPENMATH, which is

> [...] a standard for representing mathematical objects, allowing them to be exchanged between computer programs, stored in databases, or published on the worldwide web. [...] [CC98]

This is sufficient for symbolic computation services like computer algebra systems, which manipulate (simplify) or compute objects like equations or groups. Even though the logical formulae constructed or manipulated by reasoning systems like the ΩMEGA system can be expressed as OPENMATH objects, mathematical services like reasoners or presentation systems need more information e.g.:

1. is this formula an axiom, a definition, or a theorem to be proven?
2. what is a good strategy to proceed with the proof in this domain?
3. is this constant basic, or defined (so that it can be expanded to a formula involving simpler concepts)?
4. what is the common name of this concept (and its grammatical category)?

Unfortunately, OPENMATH fulfills this goal only partially, since it exclusively deals with the representation of the mathematical objects proper. Of course it would be possible to characterize an axiom by applying a predicate “axiom” to a formula or using a special variant of the equality relation for definitions, but this would only solve item 1 above.

This paper is concerned with the question about a communication standard for *mathematical knowledge*. We propose an extension OMDoc of the OpenMath standard to alleviate this perceived limitation. We will use mathematical documents as a guiding intuition for mathematical knowledge, since almost all of mathematics is currently communicated in this form (publications, letters, e-mails, talks,...). To ensure widespread applicability, we will use the term document in an inclusive, rather than exclusive way (including papers, letters, interactive books, e-mails, talks, communication between mathematical services (see for instance [FK99b,FHJ+99]) on the Internet,...), claiming that all of these can be fitted into a common representation. Since such documents normallly have a complex structure of their own, the specific task to be solved in the extension to OpenMath is to provide a standardized infrastructure for this as well. As we will use the Internet standard Xml [BPSM97] (see section 2) as a basis for this, we can consider the syntax problem for communication in MathWeb as solved by the imminent wider acceptance of Xml (OpenMath is based on Xml and we have defined an Xml representation for Kqml in [FK99a]).

Another piece of infrastructure which will play a role for understanding OMDoc is the MBase system [FK00,KF00], a MathWeb service that acts as a distributed mathematical knowledge baseknowledge base system that can answer questions as the ones shown above. OMDoc serves as a *input output language* for MBase, so that MBase can be used as a and as document preparation language. Thus the system offers a service that allows to store and (flexibly) reproduce (parts of) OMDoc documents. As OMDoc can directly be transformed to e.g. LaTeX, external input to MBase can directly be published.

To evaluate the scope of OMDoc, let us look at a few possible applications. OMDoc can serve as

- a *communication standard* between mechanized reasoning systems, e.g. the Clam-Hol interaction [BSBG98], or the ΩMEGA-TPS [BBS99] integration.
- a data format that supports the *controlled refinement* from informal presentation to formal specification of mathematical objects and theories. Basically, an informal textual presentation can first be marked up, by making its discourse structure[2] explicit, and then formalizing the textually given mathematical knowledge in logical formulae (by adding `FMP` elements; see sections 5 and 2).
- a basis for *individualized (interactive) books*book!interactiveindividualized book. OMDoc documents can be generated from MBase making use of the *discourse structurestructure!discourse information encoded in MBase.
- an interface for *proof presentation* [HF97,Fie99]: since the proof part of OMDoc allows small-grained interleaving of formal (`FMP`) and textual (`CMP`) presentations.

These and similar applications are pursued in the ΩMEGA project at the Saarland University, Saarbrücken (see `http://www.ags.uni-sb.de/~omega`) in cooperation with the RIACA project at Eindhoven.

[2] classifying text fragments as definitions, theorems, proofs, linking text, and their relations; we follow the terminology from computational linguistics here.

In the next section we will review the Internet standards and their architecture that are the basis before we come to the definition of OMDoc proper.

## 2 Markup, XML, OpenMath, MathML, and OMDoc

Mathematical (and other) texts are often written on text processors (which are often WYSIWYG type). Many authors consistently confuse information and document structure with presentation by associating formatting characteristics with various textual document components. Even in LaTeX, one can mix structural markup like `\chapter{Title}` or

```
\begin{Definition}[Title]...\end{Definition}
```

with presentation markupmarkup!presentation, such as font size information, or using

```
{\bf proof}:...\hfill\Box
```

to indicate the extent of a proof.

The problem with presentation markup is that it is specified for human consumption, and although it is *machine-readable*, the data presented in the document is not *machine-understandable*. Generally, it is very hard to automate anything for documents, where the structure is specified by presentation markup.

With the advent of the Internet, which is quickly becoming the world's fastest growing repository of mathematical documents, it is not possible to manage all the available knowledge manually, because of the volume of information distributed over the Web.

The generally accepted solution is to use *logical*logical markupmarkup!logical or *generic*generic markupmarkup!generic markup, i.e. to describe the structure of the data contained in the documents. In this markup scheme, the logical function of all document elements – title, section, paragraphs, figures, tables, bibliographic references, or mathematical equations or definitions – must be clearly defined in a machine-understandable way.

This motivation has led to the development of the "Simple Generalized Markup Language" SGML, and more recently to the "eXtensible Markup Language" XML [BPSM97] family of markup languages. XML was designed as a simplified subset of SGML that can serve as a rational reconstruction of the "Hypertext Markup Language" HTML [RHJ98], which carries most of the markup on the Internet today. From SGML, XML inherits the concept of a "document type definition" (DTD), i.e. a grammar that defines the set of well-formed documents in a given XML language and in particular, allows documents to be validated by generic tools (parsers). Moreover, presentation markup for the data specified in an XML document can be flexibly generated by using the XSL style sheet mechanism [Dea99]. In particular, it is possible to use more than one XSL style-sheet for a given document to generate specialized presentations (e.g. personalized to the tastes of a specific reader) of contained data using the content markup in the document.

Thus the "content markup" paradigm gives improved presentation (for human consumption) and improved machine readability at the same time. This

has led to considerable activity in developing specialized markup schemes for specific application areas (This paper is an instance of this activity).

OpenMath is a content markup language for communicating mathematical objects realized as an Xml language. Its syntax (given by a DTD) and semantics are specified in the evolving OpenMath standard [CC98]. The central construct of OpenMath is that of an OpenMath object (`OMOBJ`), which has a tree-like representation made up of applications (`OMA`), binding structures (`OMBIND` using `OMBVAR` to tag the bound variables), variables (`OMV`) and symbols (`OMS`).

Fig. 2 shows an OpenMath representation of the law of commutativity for addition on the reals (the logical formula $\forall a, b.a \in \mathbf{R} \wedge b \in \mathbf{R} \rightarrow a + b = b + a$). The mathematical meaning of a symbols (that of applications and bindings is

```
<OMOBJ id="commutativity-formula">
  <OMBIND>
    <OMS cd="quant1" name="forall"/>
    <OMBVAR>
      <OMV name="a"/>
      <OMV name="b"/>
    </OMBVAR>
    <OMA><OMS cd="logic1" name="implies"/>
      <OMA><OMS cd="logic1" name="and"/>
        <OMA><OMS cd="set1" name="in"/><OMV name="a"/><OMS cd="barshe" name="real"/></OMA>
        <OMA><OMS cd="set1" name="in"/><OMV name="b"/><OMS cd="barshe" name="real"/></OMA>
      </OMA>
      <OMA><OMS cd="relation" name="eq"/>
        <OMA><OMS cd="barshe" name="plus-real"/><OMV name="a"/><OMV name="b"/></OMA>
        <OMA><OMS cd="barshe" name="plus-real"/><OMV name="b"/><OMV name="a"/></OMA>
      </OMA>
    </OMA>
  </OMBIND>
</OMOBJ>
```

**Fig. 2.** An OpenMath representation of $\forall a, b.a + b = b + a$.

known from the folklore) is specified in a so-called content dictionary, which contain formal (`FMP` "formal mathematical property") or informal (`CMP` "commented mathematical property") specifications of the mathematical properties of the symbols. For instance, the specification

```
<CDDefinition>
  <Name>plus</Name>
  <Description>Addition on real numbers</Description>
  <CMP>Addition is commutative</CMP>
  <FMP><OMOBJ xref="commutativity-formula"/></FMP>
</CDDefinition>
```

could be part of the content dictionary[3] `barshe.cd` for elementary properties of real numbers (cf. section 4.3 for the relation of content dictionaries with OMDoc documents).

[3] In fact the reference `<OMOBJ xref="commutativity-formula"/>` pointing to the `OMOBJ` with the `id` attribute `commutativity-formula` uses an extension of OMDoc to OpenMath that allows to represent formulae as directed acyclic graphs preventing exponential blowup. It is licensed by the OpenMath standard, since pure OpenMath trees can be generated automatically from it.

MathML [IM98] is another XML-based markup scheme for mathematics, in contrast to OpenMath, it is more concerned with presentation markup (trying to reach LaTeX quality on the web) than with logical markup, moreover, it is mainly concerned with the K-12 fragment of mathematics (Kindergarten to $12^{th}$ grade). OpenMath is well-integrated with MathML:

- the basic content dictionaries of OpenMath mirror the MathML constructs, there are converters between the two formats.
- MathML supports the `semantics` element that can be used to annotate MathML presentations of mathematical objects with their OpenMath encoding, and OpenMath supports the `presentation` attribute that can be used for annotating with MathML presentation.
- OpenMath is the designated extension mechanism for MathML beyond K-12 mathematics.

Therefore, it is not a limitation of the presentational capabilities to use OpenMath for marking up mathematical objects. As MathML can be viewed by the WebEQ plug-in and is going to be natively supported by the primary browsers MS Internet Explorer and Netscape Navigator in version 6 (see `http://www.mozilla.org` for Mozilla, the open source version), MathML will be the primary presentation language for OMDoc.

Since OMDoc is an extension of OpenMath, it inherits its connections to XML and MathML. The structure of OMDoc documents is defined in the OMDoc document type definition DTD (cf. [Koh00b] or `http://www.mathweb.org/ilo/omdoc`, where you can also find worked examples (including part of a mathematical textbook [BS82] and an interactive book [CCS99] (IDA))).

An OMDoc document is bracketed by the XML tags `<omdoc>` and `</omdoc>`, and consists of a sequence of OMDoc items, that contain specialized representations for text, assertions, theories, definitions,... (see below). In contrast to markup languages like LaTeX, OMDoc does not partition the documents into specific units like chapters, sections, paragraphs, by tags and nesting information, but makes these document relations explicit with `omgroup` elements (see section 7.3). This choice is motivated by the generality of the document classes and the fact that the relative position of OpenMath documents can be determined in the presentation phase. In particular, since OpenMath documents can be hypertext documents, or generated from a database, it can be impossible to determine the structure of a document in advance, therefore we consider document structure information as presentation information and describe it in section 7.3.

The general pattern "definition, theorem, proof" has long been considered paradigmatic of mathematical documents like textbooks and papers. To support this structure, OMDoc provides elements for mathematical items and theory items which we will describe in sections 4 and 5. Since proofs have a more complex internal structure, we will defer them to section 6. Before we come to these, we will describe the structure of intermediate explanatory text (section 3). Finally, we will reserve section 7 for auxiliary items like exercises, applets, etc.

## 3 Text Items

The OMDoc text items are XML elements that can be used to accommodate and classify the explanatory text parts in mathematical documents. We have two kinds of them:

`CMP` These text items are used for comments and describing mathematical properties inside other OMDoc elements. They have an `xml:lang` attribute that specifies the language, they are written in, thus using groups of `CMPs` with different languages can be used for OMDoc internationalization. In conformance with the XML recommendation, we use the ISO 639 two-letter country codes (`en` $\widehat{=}$ English, `de` $\widehat{=}$ German, `fr` $\widehat{=}$ French, `nl` $\widehat{=}$ Dutch...).
`CMPs` may contain arbitrary text interspersed with OPENMATH objects (`OMOBJ` elements) (see the OPENMATH standard [CC98] for details), `omlets` (see section 7) and hyperlinks (see below). No other elements are allowed. In particular, presentation elements like paragraphs, emphases, itemizes,... are forbidden, since OMDoc is concerned with *logical* markup. Generating presentation markup from this is the duty of specialized presentation components, e.g. XSL style sheets, which can base their decisions on presentation information (see section 7.3) and the `rsrelation` element described in this section.

`ref` elements are used to specify hyperlinks via the XLINK/XPOINTER specification (see `http://www.w3c.orgTR/{xlink/xptr}`). If the reference object is defined in the same document, then it is sufficient to specify its `id` attribute in the `xlink:href` attribute, otherwise, it must include the relevant `URL` or `xpointer` material.

`omtext` OMDoc text elements can appear on top level (in `omdoc` elements). They have an `id` attribute, so that they can be cross-referenced and contain

1. an (optional) `metadata` declaration (we use the well-known Dublin Core schema, cf. `http://purl.org/dc/` or see [Koh00b])
2. an (optional) `rsrelation` element specifying the rhetorical structure relation of the text to other OMDoc elements.
3. a non-empty set of `CMP` elements that contain the text proper.

The `rsrelation` element allows to markup the discourse structure markup!discourse structure of a document in form of so-called discourse relations following the the well-known "Rhetorical Structure Theory" RST [MT83,Hor98] content model, which models a text as a tree whose leaves are the sentences (or phrases) and whose internal nodes model the relations between their daughters. This generalizes markup schemes of text fragments offered e.g. by LaTeX into categories like "Introduction", "Remark", or "Conclusion". This is sufficient for simple markup of existing mathematical texts and to replay them verbatim in a browser, but is insufficient e.g. for generating individualized, presentations at multiple levels of abstractions from the representation. The OMDoc text model – if taken to its extreme – allows to specify the respective role and contributions of smaller text units, even down to the sub-sentence level, and make the structure of mathematical texts "machine understandable".

Concretely, the `rsrelation` element specifies the relation type in a `type` attribute and the RST tree daughters in attributes `for` (for the head daughter) and `from` for the others. At the moment OMDoc uses a variant of the RST [MT83] content model that supports the relation types `introduction`, `conclusion`, `thesis`, `antithesis`, `elaboration`, `motivation`, `evidence`, `linkage` with the obvious meanings, motivated by the application to mathematical argumentative texts (see also [Hor98]). The relation type also determines the default presentation.

# 4 Theory Items

Traditionally, mathematical knowledge has been partitioned into so-called **theories**theory, often centered about certain mathematical objects like groups, fields, or vector spaces. Theories theories have been formalized as collections of

- signature declarations (the symbols used in a particular theory, together with optional typing information).
- axioms (the logical laws of the theory).
- theorems; these are in fact logically redundant, since they are entailed by the axioms.

In software engineering a closely related concept is known under the label of an (algebraic) specification, that is used to specify the intended behavior of programs. There, the concept of a theory (specification) is much more elaborated to support the structured development of specifications. Without this structure, real world specifications become unwieldy and unmanageable.

In OMDoc, we support this structured specification of theories; we build upon the technical notion of a development graphgraph!development [Hut99], since this supplies a simple set of primitives for structured specifications and also supports management of theory change. Furthermore, it is logically equivalent to a large fragment of the emerging CASL standard [CoF98] for algebraic specification (see [AHMS00]).

All specification languages support mechanisms for specifying signature and axiom information, in particular, most also support abstract data types as a convenient shorthand for sets of inductively defined objects and recursive functions on these. We will subsume these under the label of simple theories and discuss their representation in OMDoc in the next section. After that we will use section 4.2 to discuss the issue of structuring and reusing theories by importing material from other theories.

## 4.1 Simple Theories

Theories are specified by the `theory` element in OMDoc. Since signature and axiom information is particular to a given theory, the `symbol`, `definition`, `axiom` elements must be contained in a theory as sub-elements.

`symbol` This element specifies the symbols for mathematical concepts, such as 1 for the natural number "one", + for addition, = for equality, or `group` for the property of being a group. The `symbol` element has an `id` attribute

```
<theory id="monoid-thy">...
  <symbol id="monoid">
    <commonname xml:lang="en">monoid</commonname>
    <commonname xml:lang="de">Monoid</commonname>
    <commonname xml:lang="it">monoide</commonname>
    <type system="simply-typed">
       set[any] -> (any -> any -> any) -> any -> bool
    </type>
  </symbol>...
</theory>
```

**Fig. 3.** An OMDoc symbol declaration

which uniquely identifies it. This information is sufficient to allow referring back to this symbol as an OpenMath symbol. For instance the symbol declaration in Fig. 3 gives rise to an OpenMath symbol that can be referenced as `<OMS cd="monoid" name="monoid"/>` If the document containing this `symbol` element were stored in a data base system, the OpenMath symbol could be looked up by its common name. The type information specified in the `signature` element characterizes a monoid as a three-place predicate (taking as arguments the base set, the operation and a neutral element).

`definition` Definitions give meanings to (groups of) symbols (declared in a `symbol` element elsewhere) in terms of already defined ones. For example the number 1 can be defined as the successor of 0 (specified by the Peano axioms). Addition is usually defined recursively, etc.

The OMDoc `definition` element supports several kinds of definition mechanisms specified in the `type` attribute currently:

`simple` The `FMP` (see section 5) contains an OpenMath representation of a logical formula that can be substituted for the symbol specified in the `item` attribute of the definition.

`inductive` The formal part is given by a set of recursive equationsequation!recursive whose left and right hand sides are specified by the `pattern` and `value` elements in `requation` elements. The termination proof necessary for the well-definedness of the definition can be specified in the `just-by` attribute of the definition.

`implicit` Here, the `FMP` elements contain a set of logical formulae that uniquely determines the value of the symbols that are specified in the `items` slot of the definition. Again, the necessary proof of unique existence can be specified in the `just-by` attribute.

`obj` This can be used to directly give the concept defined here as an OpenMath object, e.g. as a group representation generated by a computer algebra system.

Fig. 4 gives an example a (simple) definition of a monoid.

For a description of abstract data types see [Koh00b]

## 4.2 Complex Theories and Inheritance

Not all definitions and axioms need to be explicitly stated in a theory; they can be inherited from other theories, possibly transported by signature morphism. The inheritance information is stated in an `imports` element.

```
<definition id="mon.d1" item="monoid" type="simple">
  <CMP>
    A structure (M, *, e), in which (M, *) is a semi-group
    with unit e is called a monoid.
  </CMP>
</definition>
```

**Fig. 4.** A Definition of a monoid

`imports` This element has a `from` attribute, which specifies the theory which exports the formulae.
For instance, given a theory of monoids using the symbols `set`, `op`, `neut` (and `axiom` elements stating the associativity, closure, and neutral-element axioms of monoids), a theory of groups can be given by the theory definition using `import` in Fig. 5.

```
<theory id="group">
  <imports id="group.import" from="monoid" type="global"/>
  <axiom><CMP> Every object in
    <OMOBJ><OMS cd="monoid" name="set"/></OMOBJ> has an inverse.
  </CMP></axiom>
</theory>
```

**Fig. 5.** A theory of groups based on that of monoids

`morphism` The morphism is a recursively defined function (it is given as a set of recursive equations using the `requation` element, described above). It allows to carry out the import of specifications modulo a certain renaming. With this, we can e.g. define a theory of rings given as a tuples $(R, +, 0, -, *, 1)$ by importing from a group $(M, \circ, e, i)$ via the morphism $\{M \mapsto R, \circ \mapsto +, e \mapsto 0, i \mapsto -\}$ and from a monoid $(M, \circ, e)$ via the $\{M \mapsto R^*, \circ \mapsto *, e \mapsto 1\}$, where $R^*$ is $R$ without 0 (as defined in the theory of monoids).

`inclusion` This element can be used to specify applicability conditions on the import construction. Consider for instance the situation given in Fig. 6, where the theory of lists of natural numbers is built up by importing from the theories of natural numbers and lists (of arbitrary elements). The latter imports the element specification from the parameter theory of elements, thus to make the actualization of lists to lists of natural numbers, all the symbols and axioms of the parameter theory must be fulfilled by the natural numbers. For instance if the parameter theory specifies an ordering relation on elements, this must also be present in theory `Nat`, and have the same properties there. These requirements can be specified in the `inclusion` element of OMDoc. Due to lack of space, we will not elaborate this and refer the reader to [Hut99,Koh00b].

## 4.3 OpenMath Documents and Content Dictionaries

In the examples we have already seen that OMDoc documents contain definitions of mathematical concepts, which need to be referred to using OpenMath symbols. In particular, documents describing theories like `barshe.omdoc` or `ida.omdoc` even reference OpenMath symbols they define themselves. Thus

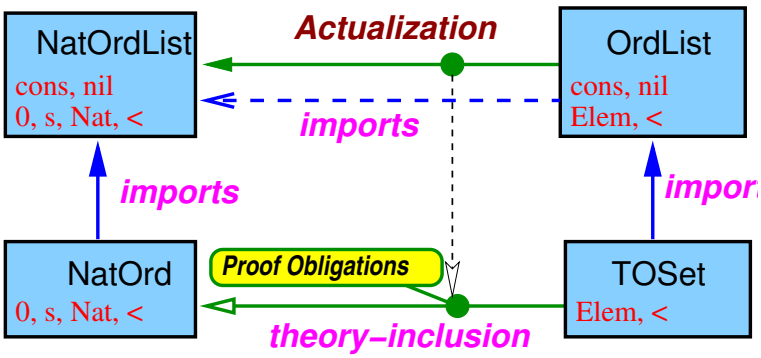


**Fig. 6.** A Structured Specification of Lists

it is necessary to generate OpenMath content dictionariescontent dictionary from OMDoc documents, or develop an alternative mechanism to establish symbol identity in `OMS`. The generation of content dictionaries is already supported in the MBase system, but can also be achieved by writing specialized Xsl style sheets. For the purposes of this paper, we will only assume that one of these measures has been taken.

## 5 Mathematical Items

We will now present the mathematical items that are not integral parts of a theory, since they are optional (they can be derived from the material specified in the theory), they can be specified outside a theory element. We have the following elements:

`FMP` This is the general element for representing mathematical formulae as OpenMath objects, for instance the formula in Fig. 2. As logical formulae often come as sequents, i.e. a conclusion is drawn from a set of assumptions, OMDoc also allows the content of an `FMP` to be a (possibly empty) set of `assumption` elements followed by a `conclusion` (all of which contain `OMOBJ` elements). The intended meaning is that the `FMP` asserts that the conclusion is entailed by the assumptions in the current context. As a consequence, `<FMP><conclusion>`$A$`</conclusion></FMP>` is equivalent to `<FMP>`$A$`</FMP>`.

`assertion` This is the element for all statements (proven or not) about mathematical objects (see Fig. 7). Traditional mathematical documents discern various kinds of these: theorems, lemmatalemma, corollariescorollary, conjectures, problems, etc. These all have the same structure (formally, a closed logical formula). Their differences are largely *pragmatic* (theorems are normally more important in some theory than lemmata) or proof-theoretic (conjectures become theorems once there is a proof). Therefore, we represent them in the general `assertion` element and leave the type distinction to a `type` attribute. These type specifications in OMDoc documents should only be regarded as defaults, since e.g. reusing a mathematical paper as a chapter in a larger monography, may make it necessary to downgrade a theorem (e.g. the main theorem of the paper) and give it the status of a lemma in the overall work.

```
<assertion id="ida.c6s1p4.l1" type="lemma">
  <CMP> A semi-group has at most one unit.</CMP>
</assertion>
```

**Fig. 7.** An assertion about semigroups

`alternative-def` Since there there can be more than one definition per symbol, OMDoc supplies the `alternative-def` element that can be specified outside a theory that can be specified outside a given theory.

`example` In mathematical practice, examples play an equally great role as proofs, e.g. in concept formation (as witnesses for definitions or as either supporting evidence, or as counterexamples for conjectures). Therefore, examples are given status as primary objects in OMDoc. Conceptually, we model an example for a mathematical concept $\mathbf{C}$ as a triple $(\mathcal{W}, \mathbf{A}, \mathcal{P})$, where $\mathcal{W} = (\mathcal{W}_1, \ldots, \mathcal{W}_n)$ is an $n$-tuple of mathematical objects, $\mathbf{A}$ is an assertion of the form $\mathbf{A} = \exists W_1 \ldots W_n.\mathbf{B}$, and $\mathcal{P}$ is a proof that shows $\mathbf{A}$ by exhibiting the witnesses $\mathcal{W}_i$ for $W_i$. The example $(\mathcal{W}, \exists W_1 \ldots W_n.\neg\mathbf{B}, \mathcal{P})$ is a counterexample to a conjecture $\mathbf{T} := \forall W_1 \ldots W_n.\mathbf{B}$, and $(\mathcal{W}, \mathbf{A}, \mathcal{P}')$ a supporting example for $\mathbf{T}$.

OMDoc specifies this intuition in an element `example` that contains a set of OpenMath objects (the witnesses), and has the attributes

- `item` (for what concept or assertion is it an example),
- `type` (one of the keywords `for` or `against` for the function)
- `assertion` (a reference to the assertion $\mathbf{A}$ mentioned above)
- `proof` (a reference to the constructive proof $\mathcal{P}$)

Consider for instance the structure $\mathcal{W} := (A^*, \circ)$ of the set of words over an alphabet $A$ together with word concatenation $\circ$. Then $(\mathcal{W}, \exists W.\texttt{monoid}(W), \mathcal{P}_1)$ is an example for the concept of a monoid (with the empty word as the neutral element), if e.g. $\mathcal{P}_1$ uses $\mathcal{W}$ to show the existence of $W$. The example $(\mathcal{W}, \exists V_{\texttt{monoid}}.\neg\texttt{group}(V), \mathcal{P}_2$ and a proof that uses $\mathcal{W}$ as a witness for $V$, it is a counterexample to the conjecture $\mathbf{C} := \forall V_{\texttt{monoid}}.\texttt{group}(V)$, since $\mathbf{Q} \to \neg\mathbf{C}$. Fig. 8 gives the OMDoc representation of this example of an example.

```
<example id="mon.ex1" item="monoid" type="for"
         assertion="strings-are-monoids" proof="sam-pf">
  <CMP>The set of strings with concatenation</CMP>
  <OMOBJ><OMS cd="simple-monoids" name="strings"/></OMOBJ>
</example>
<example id="mon.ex2" item="monoid" type="against"
         assertion="monoids-are-groups" proof="mag-pf">
  <CMP>The set of strings with concatenation is not a group</CMP>
  <OMOBJ><OMS cd="simple-monoids" name="strings"/></OMOBJ>
</example>
```

**Fig. 8.** An OMDoc representation of an example

Finally, there are OMDoc elements that allow to structure the knowledge in theories. We have already seen the possibility to define (parts of) theories by

so-called theory morphism specified in `imports` and `include` elements in section 4.2. Following Hutter's development graph [Hut99], we can use the knowledge about theories to establish so-called inclusion morphismsmorphism!inclusion that establish the source theory as included (modulo renaming by a morphism) in the target theory. This information can be used to add further structure to the theory graph and help maintain the knowledge base with respect to changes of individual theories.

An `axiom-inclusion` element contains a `morphism` (see section 4.2), and the attributes `from` and `to` specify the source and target theories. For any axiom in the source theory there must be an assertion in the target theory (whose `FMP` is just the image of the `FMP` of the axiom under the morphism) with a proof. These are represented by an empty `by` element, which has the attributes `axiom`, `assertion`, and `proof` with the obvious meanings.

A `theory-inclusion` is a global variant of `axiom-inclusion` that can be obtained as a path of `axiom-inclusions` (or other `theory-inclusion`) which are specified in the `by` attribute.

## 6 Proofs

*Proofs*proof are representations of evidence for the truth of *assertion*. Like in the case of definitions, there can in general be more than one proof for a given assertion. Furthermore, it will be initially infeasible to totally formalize all mathematical proofs needed for the correctness management of the knowledge base in one universal proof format, therefore OMDoc supports a proof format whose structural and formal elements are derived from the $\mathcal{PDS}\mathcal{PDS}$proof plan data structure[4] structure developed for the ΩMEGA system, but also allows natural language representations at every level. In the future, it may be necessary and advantageous to allow various other proof representations there like proof scripts (ΩMEGA replay files, ISABELLE proof scripts,...), references to published proofs, resolution proofs, etc, to enhance the coverage.

This mixed representation enhances multi-modal proof presentation [Fie97], and the accumulation of proof information in one structure. Informal proofs can be formalized [Bau99]; formal proofs can be transformed to natural language [HF96].

The OMDoc `proof` environment contains a list of proof steps. Such `derive` steps have the attributes `id` (so it can be referred to) and the optional `type` attribute. It can contain the following child elements (in this order)

`CMP` This gives the natural language representation of the proof step.

[4] The **P**roof plan **D**ata **S**tructure ($\mathcal{PDS}$) was introduced in the ΩMEGAΩMEGA [BCF+97] system to facilitate hierarchical proof planning and proof presentation at more than one level of abstraction. In a $\mathcal{PDS}$, expansions of nodes justified by tactic applications are expanded, but the information about the tactic itself is not discarded in the process as in tactical theorem provers like ISABELLE or NUPRL. Thus proof nodes may have justifications at multiple levels of abstraction in a hierarchical proof data structure.

The rest of the children form the formal content of the derive step, together, they represent the information present e.g. in a $\mathcal{PDS}$ node.

`assumption, conclusion` A formal representation of the local assumptions and the assertion made by this proof step, they contain `CMP` and `FMP` elements. Local assumptions should not be referenced to outside the derive step they were made in. Thus the derive step serves as a grouping device for local assumptions.

`method` is an OpenMath symbol representing a proof method, tactic, or inference rule that justifies the assertion made in the `conclusion` element.

`premise` These are empty elements whose `xlink:href` attribute is used to refer to the proof- or local assumption nodes that the `method` was applied to to yield this result.

`proof` If a derive step is a logically (or even mathematically) complex step that can be expanded into sub-steps, then the embedded `proof` element can be used to specify the sub-derivation (which can have similar expansions in embedded `proof` environments again).

This embedded `proof` allows us to specify generic markup for the hierarchic structure of proofs.

```
<derive id="barshe.2.1.2.proof.a.proof.D2.1">
  <CMP>By <OMOBJ><OMS cd="barshe" name="alg-prop-reals.A2"/></OMOBJ>
       we have z + (a + (−a)) = a + (−a)
  </CMP>
  <conclusion>(z + a) + (−a) = z + (a + (−a))</conclusion>
  <method><OMS cd="omega-base-calc" name="foralli*"/>c
    <parameter><OMOBJ><OMV name="z"/></OMOBJ></parameter>
    <parameter><OMOBJ><OMV name="a"/></OMOBJ></parameter>
    <parameter>−a</parameter>
  </method>
  <premise xlink:href="alg-prop-reals.A2"/>
</derive>
```

**Fig. 9.** A `derive` proof step

## 7 Auxiliary Elements

In this section we will present OMDoc elements that are not strictly mathematical content, but have useful functions mathematical documents or knowledge bases. For the OMDoc representations of things like exercises we refer the reader to [Koh00b] and concentrate on the representation of applets and presentation information instead.

### 7.1 Non-Xml Data and Program Code in OMDoc

Sometimes mathematical services have to be able to communicate (e.g. to the MBase system for storage) data in non-Xml syntax, or whose format is not sufficiently fixed to warrant for a general Xml encoding. Examples of this are pieces of program code, like tactics of tactical theorem provers, linguistic data of proof presentation system, etc. One characteristic of such data seems to be that

it is private to certain applications, but may be relevant to more than one user. For this, OMDoc provides the `private` element, which contains a the usual `CMPs` and a `data` element described below. It has the attributes

`pto` specifies the system to which this data is private.
`pto-version` its version, this may be necessary, if the data (format) changes with versions.
`format/type` the type of the data and the format the data is in, the meaning of these fields is determined by the system itself.
`requires` specifies the identifiers of the items that this data depends upon, this will often be `code` items.
`theory` allows to specify the mathematical theory (see section 4) that the data is associated with.

The `data` element contains the data of a in a `CDATA` section (this is the XML way of allowing data that cannot be parsed by the XML parser). If the content of this field is too large to store directly in the OMDoc or often changes, then it can be substituted by a link, specified in the `xlink:href` attribute.

The `code` element is for embedding pieces of code into an OMDoc document. This element has the same attributes as the `private` element, like it, it can contain `CMP`, and `data` fields. Furthermore, it can contain documentation elements `input`, `output` and `effect` that specify the behavior of the procedure defined by the code fragment.

## 7.2 Applets in OMDoc

`omlet` elements contain OMDoc specifications of applets (program code that can in some way executed during document manipulation). `omlets` generalize the well-known *applet* concept in two ways: The computational engine is not restricted to *plug-ins*, of the browser (current servlet technology can be used and specified using `code` and `omlet` elements in OMDocs) and the program code can be specified and distributed more easily. Making document-centered computation easier to manage.

```
<code id="callmint">
  <input>None</input>
  <output>The result</output>
  <effect>None</input>
  <data><![CDATA[... the call-mint code goes here...]]></data>
</code>
<derive id="monp_1">
  <CMP> <omlet type="js" function="callMint">Intros.</omlet></CMP>
  <method><OMS name="Intros" cd="COQ"/></method>
</derive>
```

**Fig. 10.** An `omlet`

Like the HTML `applet` tag, the `omlet` element can be used to wrap any (set of) well-formed element. It has the following attributes.

`type` This specifies the computation engine that should execute the code. Depending on the application, this can be a programming language, such as `javascript` (`js`) or Oz, or a process that is running (in our case the $\mathcal{LOUI}$ or ΩMEGA services).

`function` The code that should be executed by the omlet is specified in the `function` attribute. This points to an OMDoc code element that is somehow accessible (e.g. in the same OMDoc). This indirection allows us to reuse the machinery for storing code in OMDocs. For a simple example see Fig. 10.

`argstr` allows to specify an (optional) argument string for function, so that the program in the can be kept general. A call to the $\mathcal{LOUI}$ interface, would then have the form in Fig. 11. Here, the code in the `code` element `sendtoloui` (which we have not shown) would be java code that simply sends the argstr to $\mathcal{LOUI}$'s remote control port.

The expected behavior of the `omlet` can be implemented in the Xsl style sheet, that in the case of e.g. translation to Mozilla will put the `callmint` code directly into the generated `html`.

```
<CMP> Let's prove it
  <omlet id="bla type="java" function="sendtoloui"
         argstr="load(problem='monoid_uniq)">
    interactively
  </omlet>
</CMP>
```

**Fig. 11.** An `omlet` calling an external process

## 7.3 Presentation

In the introduction we have stated that one of the design intentions behind OMDoc is to separate content from presentation, and leave the latter to the user. In this section, we will briefly touch upon presentation issues. The technical side of this is simple: OMDoc documents are regular Xml documents that can be processed by Xsl [Dea99] style sheet to produce conventional presentations from OMDoc representations of mathematical documents. At the moment, we have Xsl style sheets to convert OMDoc to HtML (one each specialized to the respective browsers), LaTeX, and to the input languages of the ΩMEGA, InKa, and *λClam* systems (they can be found at `http://www.mathweb.org/ilo/omdoc`). At the moment, these hard-code certain presentation decisions for the overall appearance of the documents, but we are working on style sheet generators that make these user adaptive.

The mathematical concepts and symbols introduced in an OMDoc document (`symbol` elements) often carry typographic conventions, that cannot be determined by general principles alone. Therefore, they need to be specified in the document itself, so that typographically good representations can be generated from this (and subsequent) documents. The `presentation` element in Fig. 12 allows to add Xsl style sheet information to symbols, where they are

```
<presentation format="TeX">
  <xsl:template match="OMA[OMS[position()=1 and
                               @name='monoid' and
                               @cd='ida.monoid']]">
    (<xsl:apply-templates select="*[2]"/>,
     <xsl:apply-templates select="*[3]"/>,
     <xsl:apply-templates select="*[4]"/>)\in{\bf MON}
  </xsl:template>
</presentation>
```

**Fig. 12.** XSL Presentation for the symbol in Fig. 3

defined. In this case, the style sheet information will cause an OPENMATH expression

```
<OMA>
  <OMS cd="ida" name="monoid"/><OMV name="M"><OMV name="o"><OMV name="e">
</OMA>
```

to be rendered as $(M, o, e) \in \mathbf{MOD}$ in a TEX or LATEX document derived from `ida.xml` via a suitable XSL style sheet. Of course, this information will need to be included into the respective style sheets. This is easily realized by a two-stage style sheet process: in the first pass, a general (higher-order) style sheet extracts the presentation information from the relevant OMDOC documents, and in the second stage, this is used to present the `OMOBJs` in the source OMDOC.

The presentation elements discussed up to now, allow to specify the presentation of OPENMATH elements. To specify the overall structure of mathematical texts, such as books, chapters, sections, or paragraph, but also enumerations, itemizes, lists, we use the `omgroup` element. We use a general construct that specifies the presentation in the `type` attribute, since the presentation component (style sheet) may need to decide on that. `omgroup` elements contain an optional `metadata` element and then a sequence of `omgroup` and `ref` elements. The first allow to define a recursive document structure, and the second are used to refer to other OMDOC elements by the use of `xlink` attributes (most notably `xlink:href` for hyperlinks).

Note that this representation that relies on explicit (hyper)-references instead of nesting information allows to specify more than one document using the mathematical material specified in the other OMDOC items. In particular, it becomes possible to specify and store more than one linearization of the material in a document, or generate linearizations or "guided tours" see [SBC+00] for a details.

## 8 Conclusion

We have proposed an extension to the OPENMATH standard that allows to represent the semantics and structure various kinds of mathematical documents, including articles, textbooks, interactive books, courses. We have motivated and described the language and presented an XML document type definition for it.

We are currently testing this in the development of a user-adaptive interactive book including proof explanation based on IDA [CCS99] in close collaboration with the authors. This case study unites several of the application areas discussed in the introduction. The re-representation of IDA in the OMDOC format

makes it possible to machine-understand the structure of the document, read it into the MBase [FK00,KF00] knowledge base system without loss of information, preserving the structure, and generate personalized sub-documents or linearizations of the structured data based on a simple user model. Furthermore, the OMDoc representation supports the formalization of (parts of) the mathematical knowledge in IDA and makes it accessible to the ΩMEGA mathematical assistant system [BCF+97], which can prove some of the problems either fully automatically (by proof planning) or in interaction with the authors. This newly developed formal data (it is not present in IDA now) will enable the reader to read and experiment with the proofs behind the mathematical theory, much as she can in the present version with the integrated computer algebra system GAP [S+95]. Finally, OMDoc will serve as the input format for the Lima system (see [Bau99]), an experimental natural language understanding system specialized to mathematical texts (this can be used to develop formalization in `FMPs` from the text in the respective `CMPs`).

In the context of this project, we have developed first authoring tools for OMDoc that try to simplify generating OMDoc documents for the working mathematician. There is a simple OMDoc mode for `emacs`, and a LaTeX style [Koh00a] that can be used to generate OMDoc representations from LaTeX sources and thus help migrate existing mathematical documents. A second step will be to integrate the LaTeX to OpenMath conversion tools. Michel Vollebregt has built a program that traverses an OMDoc and substitutes various representations for formulae (including the Mathematica, GAP, and Maple representations) with the corresponding OpenMath representations.

## Acknowledgments

The work presented in this report was supported by the "Deutsche Forschungsgemeinschaft" in the special research action "Resource-adaptive cognitive processes" (SFB 378), Project ΩMEGA.

The author would like to thank Armin Fiedler, Andreas Franke, Martin Pollet, and Julian Richardson for productive discussions, and the RIACA group (specifically Arjeh Cohen, Olga Caprotti, Michel Vollebregt, and Manfred Riem) for valuable input from the IDA case study.

## References

[AHMS00] Serge Autexier, Dieter Hutter, Heiko Mantel, and Axel Schairer. Towards an evolutionary formal software-development using CASL. In C. Choppy and D. Bert, editors, *Proceedings Workshop on Algebraic Development Techniques, WADT-99*. Springer, LNCS 1827, 2000.

[AK00] Alessandro Armando and Michael Kohlhase. Communication protocols for mathematical services based on KQML and OMRS. In Manfred Kerber and Michael Kohlhase, editors, *CALCULEMUS-2000, Systems for Integrated Computation and Deduction*, 2000, AKPeters. forthcoming.

[AvLS96] J. Abbot, A. van Leeuwen, and A. Strotmann. Objectives of OpenMath. Technical report 12, RIACA, Technische Universiteit Eindhoven, The Netherlands, June 1996.

[AZ00] Alessandro Armando and Daniele Zini. Towards Interoperable Mechanized Reasoning Systems: the Logic Broker Architecture. In A. Poggi, editor, *AI*IA-TABOO Workshop 'From Objects to Agents: Evolutionary Trends of Software Systems'*, 2000.

[Bau99] Judith Baur. Syntax und semantik mathematisher texte — ein prototyp. Master Thesis, Saarland University, 1999.

[BBS99] Christoph Benzmüller, Matthew Bishop, and Volker Sorge. Integrating TPS and ΩMEGA. *Journal of Universal Computer Science*, 5(2), 1999.

[BCF+97] The ΩMEGA group. ΩMEGA: Towards a mathematical assistant. In William McCune, editor, *CADE-14*, LNAI 1249, pages 252–255, 1997. Springer Verlag.

[BPSM97] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML). W3C Recommendation TR-XML, December 1997. `http://www.w3.org/TR/PR-xml.html`.

[BS82] Robert G. Bartle and Donald Sherbert. *Introduction to Real Analysis*. Wiley, 2 edition, 1982.

[BSBG98] R. Boulton, K. Slind, A. Bundy, and M. Gordon. An interface between CLAM and HOL. In Jim Grundy and Malcolm Newey, editors, *TPHOLS-98*, pages 87–104, 1998.

[CC98] Olga Caprotti and Arjeh M. Cohen. Draft of the Open Math standard. The Open Math Society, `http://www.nag.co.uk/projects/OpenMath/omstd/`, 1998.

[CCS99] Arjeh Cohen, Hans Cuypers, and Hans Sterk. *Algebra Interactive!* Springer Verlag, 1999. Interactive Book on CD.

[CoF98] Language Design Task Group CoFI. Casl - the CoFI algebraic specification language - summary, version 1.0. Technical report, `http://www.brics.dk/Projects/CoFI`, 1998.

[DCN+00] Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham. The Prosper toolkit. In *Proc. TACAS-2000*, 2000.

[Dea99] Stephen Deach. Extensible stylesheet language (xsl) specification. W3c working draft, W3C, 1999. `http://www.w3.org/TR/WD-xsl`.

[FF94] T. Finin and R. Fritzson. KQML — a language and protocol for knowledge and information exchange. In *Proceedings of the 13th Intl. Distributed Artificial Intelligence Workshop*, pages 127–136, 1994.

[FHJ+99] Andreas Franke, Stephan M. Hess, Christoph G. Jung, Michael Kohlhase, and Volker Sorge. Agent-oriented integration of distributed mathematical services. *Journal of Universal Computer Science*, 5:156–187, 1999.

[Fie97] Armin Fiedler. Towards a proof explainer. In Siekmann et al. [SPH97], pages 53–54.

[Fie99] Armin Fiedler. Using a cognitive architecture to plan dialogs for the adaptive explanation of proofs. In Thomas Dean, editor, *Proceedings IJCAI-99*, pages 358–363, 1999. Morgan Kaufmann.

[FK99a] Andreas Franke and Michael Kohlhase. Communicating with MBASE in KQML. Internet Draft `http://www.mathweb.org/mbase`, 1999.

[FK99b] Andreas Franke and Michael Kohlhase. System description: MATHWEB, an agent-based communication layer for distributed automated theorem proving. In Harald Ganzinger, editor, *Proceedings CADE-16*, LNAI 1632, pages 217–221. Springer Verlag, 1999.

[FK00] Andreas Franke and Michael Kohlhase. System description: MBase, an open mathematical knowledge base. In David McAllester, editor, *CADE-17*, LNAI 1831, pages 455–459. Springer Verlag, 2000.

[HF96] Xiaorong Huang and Armin Fiedler. Presenting machine-found proofs. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings CADE-13*, LNAI 1104, pages 221–225, 1996. Springer Verlag.

[HF97] Xiaorong Huang and Armin Fiedler. Proof verbalization in *PROVERB*. In Siekmann et al. [SPH97], pages 35–36.

[Hor98] Helmut Horacek. Generating inference-rich discourse through revisions of RST-trees. In *Proceedings AAAI-98*, pages 814–820. MIT Press, 1998.

[Hut99] Dieter Hutter. Reasoning about theories. Technical report, DFKI, 1999.

[IM98] Patrick Ion and Robert Miner. Mathematical Markup Language (MathML) 1.0 specification. W3C Recommendation 1998. `http://www.w3.org/TR/REC-MathML/`.

[KF00] Michael Kohlhase and Andreas Franke. Mbase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Comutation*, 2000. forthcoming.

[Koh00a] Michael Kohlhase. Creating omdoc representations from LaTeX. Internet Draft `http://www.mathweb.org/omdoc`, 2000.

[Koh00b] Michael Kohlhase. OMDoc: Towards an openmath representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000.

[MT83] William Mann and Sandra Thompson. Rhethorical structure theory: A theory of text organization. Technical Report ISI/RR-83-115, ISI at University of Southern California, 1983.

[RHJ98] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification. W3C Recommendation 1998. `http://www.w3.org/TR/PR-xml.html`.

[S+95] Martin Schönert et al. *GAP – Groups, Algorithms, and Programming*. RWTH Aachen, Germany, 1995.

[SBC+00] The ΩMEGA group. Adaptive course generation and presentation. In P. Brusilovski, editor, *Proceedings of ITS-2000 workshop on Adaptive and Intelligent Web-Based Education Systems*, Montreal, 2000.

[Sho90] Y. Shoham. Agent-Oriented Programming. Technical report, Stanford University, 1990.

[SHS98] M. Kohlhase S. Hess, Ch. Jung and V. Sorge. An implementation of distributed mathematical services. In Arjeh Cohen and Henk Barendregt, editors, *CALCULEMUS and TYPES*, 1998.

[SPH97] J. Siekmann, F. Pfenning, and X. Huang, editors. *Proceedings of the First International Workshop on Proof Transformation and Presentation*, Schloss Dagstuhl , Germany, 1997.