x

Preface

In documentwe will present an abstract library for processing collections of OMDoc documents.

Chapter 1

OMDoc resources

In this chapter we will describe various public resources for working with the OMDoc format.

1.1 The OMDoc Web Site, Wiki, and Mailing List

The main web site for the OMDoc format is `http://www.omdoc.org`. It hosts news about developments, applications, collaborators, and events, provides access to an list of "frequently asked questions" (FAQ), and current and old OMDoc specifications and provides pre-generated examples from the OMDoc distribution.

There are two mailing lists for discussion of the OMDoc format:

`omdoc@omdoc.org` is for announcements and discussions of the OMDoc format on the user level. Direct your questions to this list.

`omdoc-dev@omdoc.org` is for developer discussions.

For subscription and archiving details see the OMDoc resources page for mailing lists [**OMDoc-mailinglists:URL**].

Finally, the OMDoc web site hosts a Wiki [**OMDoc:wiki**] for user-driven documentation and discussion.

1.2 The OMDoc distribution

All resources on the OMDoc web site are available from the MathWeb Subversion repository [**OMDoc:svn**] for anonymous download. Subversion (svn) is a collaborative version control system – to support a distributed community of developers in accessing and developing the OMDoc format, software, and documentation, see [**omdoc.org:svn**] for a general introduction to the setup. The resources for version 1.6 of the OMDoc format which is described in this book are accessible on the web at `https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.6` via a regular web browser[1]. The svn server allows anonymous read access to the general public. To check out the OMDoc distribution, use

```
svn co https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.6
```

This will create a directory `omdoc-1.6`, with the sub-directories

[1]Ongoing development of the OMDoc format can be accessed via the head revision of the repository at `https://svn.omdoc.org/repos/omdoc/trunk`.

directories	content
`bin`, `lib`, `oz`, `thirdParty`	programs and third-party software used in the administration and examples
`css`, `xsl`	style sheets for displaying OMDoc documents on the web, see Chapter ?? for a discussion.
`doc`	The OMDoc documentation, including the specification, papers about a the OMDoc format and tools.
`dtd`, `rnc`	The OMDoc document type definition and the RelaxNG schemata for OMDoc
`examples`	Various example documents in OMDoc format.
`projects`	various contributed developments for OMDoc. Documentation is usually in their `doc` sub-directory

After the initial check out, the OMDoc distribution can be kept up to date by the command `svn -q update` in the top-level directory from time to time. To obtain write access contact `svnadmin@omdoc.org`.

1.3 The OMDoc bug tracker

`MathWeb.org` supplies a BugZilla bug-tracker [**bugzilla:URL**] at `http://bugzilla.mathweb.org:8000` to aid the development of the OMDoc format. BugZilla is a server-based discussion forum and bug tracking system. We use it to track, archive and discuss tasks, software bugs, and enhancements in our project. Discussions are centered about threads called "bugs" (which need not be software bugs at all), which are numbered, can be searched, and can be referred to by their URL. To use BugZilla, just open an account and visit the OMDoc content by querying for the "product" OMDoc. For offline use of the bug-tracker we recommend the excellent Deskzilla application [**deskzilla:URL**], which is free for open-source projects like OMDoc.

Further development of the OMDoc format will be public and driven by the discussions on BugZilla, the OMDoc mailing list, and the OMDoc Wiki (see Section 1.1).

1.4 An XML catalog for OMDoc

Many XML processes use system IDs (in practice URLs) to locate supporting files like DTDs, schemata, style sheets. To make them more portable, OMDoc documents will often reference the files on the `omdoc.org` web server, even in situations, where they are accessible locally e.g. from the OMDoc distribution. This practice not only puts considerable load on this server, but also slows down or even blocks document processing, since the XML processors have to retrieve these files over the Internet.

Many processors can nowadays use XML catalogs to remap public identifiers and URLs as an alternative to explicit system identifiers. A catalog can convert public `IDs` like the one for the OMDoc DTD (`-//OMDoc//DTD OMDoc V1.6//EN`) into absolute URLs like `http://www.omdoc.org/dtd/omdoc.dtd`. Moreover, it can replace remote URLs like this one with local URLs like `file:///home/kohlhase/omdoc/dtd/omdoc.dtd`. This offers fast, reliable access to the DTDs and schemata without making the documents less portable across systems and networks.

To facilitate the use of catalogs, the OMDoc distribution provides a catalog file `lib/omdoc.cat`. This catalog file can either be imported into the system's catalog[2] using a `nextCatalog` element of the form

```
<nextCatalog xml:id="omdoc.cat" catalog="file:///home/kohlhase/omdoc/lib/omdoc.cat"/>
```

or by making it known directly to the XML processor by an application-specific method. For instance for `libxml2` based tools like `xsltproc` or `xmllint`, it is sufficient to include the path to `omdoc.cat` in the value of the `XML_CATALOG_FILES` environment variable (it contains a whitespace-separated list of FILES).

[2]This catalog is usually at `file:///etc/xml/catalog` on Unix systems; unfortunately there is no default location for Windows machines.

1.5 External Resources

The OMDoc format has been used on a variety of projects. Chapter **??** gives an overview over some of the projects (use the project home pages given there for details), a up to date list of links to OMDoc projects can be found at `http://omdoc.org/projects.html`. These projects have contributed tools, code, and documentation to the OMDoc format, often stressing their special vantage points and applications of the format.

[=validating]

Part I

Validating OMDoc Documents

In Chapter **??** we have briefly discussed the basics of validating XML documents by document type definitions (DTDs) and schemata. In this chapter, we will instantiate this discussion with the particulars of validating OMDoc documents.

Generally, DTDs and schemata are context-free grammars for trees[3], that can be used by a validating parser to reject XML documents that do not conform to the constraints expressed in the OMDoc DTD or schemata discussed here.

Note that none of these grammars can enforce all constraints that the OMDoc specification in Part **??** of this book imposes on documents. Therefore grammar-based validation is only a necessary condition for OMDoc-**validity** . Still, OMDoc documents should be validated to ensure proper function of OMDoc tools, such as the ones discussed in Chapters **??** and **??**. Validation against multiple grammars gives the best results. With the current state of validation technology, there is no clear recommendation, which of the validation approaches to prefer for OMDoc. DTD validation is currently best supported by standard XML applications and supports default values for attributes. This allows the author who writes OMDoc documents by hand to elide implicit attributes and make the marked-up text more readable. XML- and RelaxNG schema validation have the advantage that they are namespace-aware and support more syntactic constraints. Neither of these support mnemonic XML entities, such as the ones used for Unicode characters in Presentation-MathML, so that these have to be encoded as Unicode code points. Finally RelaxNG schemata do not fully support default values for attributes, so that OMDoc documents have to be normalized[4] to be RelaxNG-valid.

We will now discuss the particulars of the respective validation formats. As the RelaxNG schema is the most expressive and readable for humans we consider it as the normative grammar formalism for OMDoc, and have included it in ?rnc? for reference.

[3]Actually, a recent extension of the XML standard (XLink) also allows to express graph structures, but the admissibility of graphs is not covered by DTD or current schema formalisms.

[4]An OMDoc document is called **normalized** , iff all required attributes are present. Given a DTD or XML schema that specifies default values, there are standard XML tools for XML-normalization that can be pipelined to allow RelaxNG validation, so this is not a grave restriction.

1.6 Validation with Document Type Definitions

The OMDoc document type definition [**OMDocDTD:URL**] can be referenced by the public identifier `"-//OMDoc//DTD OMDoc V1.6//EN"` (see Section 1.4). The DTD driver file is `omdoc.dtd`, which calls various DTD modules.

DTD-validating XML parsers are included in almost all XML processors. The author uses the open-source RXP [**Tobin:RXP**] and XMLLINT [**Veillard:libxml2**] as stand-alone tools. If required, one may validate OMDoc documents using an SGML parser such as nsgmls, rather than a validating XML parser. In this case an SGML declaration defining the constraints of XML applicable to an SGML parser must be used (see [**Clark:csx97**] for details).

To allow DTD-validation, OMDoc documents should contain a document typedeclaration of the following form:

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.6//EN"
                       "http://www.omdoc.org/dtd/omdoc.dtd">
```

The URI may be changed to that of a local copy of the DTD if required, or it can be dropped altogether if the processing application has access to an XML catalog (see Section 1.4). Whether it is useful to include document type declarations in documents in a production environment depends on the application. If a document is known to be DTD- or even OMDoc-valid, then the validation overhead a `DOCTYPE` declaration would incur from a validating parser[5] may be conserved by dropping it.

1.6.1 Parametrizing the DTD

The OMDoc DTD makes heavy use of parameter entities, so we will briefly discuss them to make the discussion self-contained. Parameter entity declarations are declarations of the form

```
<!ENTITY % assertiontype "theorem|proposition|lemma|%otherassertiontype;">
```

in the DTD. This one makes the abbreviation `%assertiontype;` available for the string "`theorem|proposition|lemma`
(in the DTD of the document in Listing 1.1). Note that parameter entities must be fully defined before they can be referenced, so recursion is not possible. If there are multiple parameter entity declarations, the first one is relevant for the computation of the replacement text; all later ones are discarded. The internal subset of document type declaration is pre-pended to the external DTD, so that parameter entity declarations in the internal subset overwrite the ones in the external subset.

The (external) DTD specified in the `DOCTYPE` declaration can be enhanced or modified by adding declarations in square brackets after the DTD URI. This part of the DTD is called the internal subset of the `DOCTYPE` declaration, see Listing 1.1 for an example, which modifies the parameter entity `%otherassertiontype;` supplied by the OMDoc DTD to extend the possible values of the `type` attribute in the `assertion` element for this document. As a consequence, the `assertion` element with the non-standard value for the `type` attribute is DTD-valid with the modified internal DTD subset.

Listing 1.1: A Document Type Declaration with Internal Subset

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.6//EN"
                       "http://www.omdoc.org/dtd/omdoc.dtd"
  [<!ENTITY % otherassertiontype "observation">]>
...
<assertion type="observation">...</assertion>
...
```

[5]The XML specification requires a validating parser to perform validation if a `DOCTYPE` declaration is present

1.6.2 DTD-based Normalization

Note that if a OMDoc fragment is parsed without a DTD, i.e. as a well-formed XML fragment, then the default attribute values will not be added to the XML information set. So simply dropping the DOCTYPE declaration may change the semantics of the document, and OMDoc documents should be normalized[6] first. Normalized OMDoc documents should carry the `standalone` attribute in the XML processing instruction, so that a normalized OMDoc document has the form given in Listing 1.2.

Listing 1.2: A normalized OMDoc document without DTD

```
<?xml version="1.0" standalone="yes"?>
<omdoc xml:id="something" version="1.6" xmlns="http://omdoc.org/ns">
 ...
</omdoc>
```

The attribute `version` and the namespace declaration `xmlns` are fixed by the DTD, and need not be explicitly provided if the document has a `DOCTYPE` declaration.

1.6.3 Modularization

In OMDoc1.2 the DTD has been modularized according to the W3C conventions for DTD modularization [**AltBou:mox01**]. This partitions the DTD into DTD modules that correspond to the OMDoc modules discussed in Part **??** of this book.

These DTD modules can be deselected from the OMDoc DTD by changing the **module inclusion entities** in the local subset of the document type declaration. In the following declaration, the module PF (see Chapter **??**) has been deselected, presumably, as the document does not contain proofs.

```
<!DOCTYPE omdoc PUBLIC "-//OMDoc//DTD OMDoc V1.6//EN"
                              "http://www.omdoc.org/dtd/omdoc.dtd"
  [<!ENTITY % omdoc.pf.module "IGNORE">]>
```

Module inclusion entities have the form `%omdoc.`⟪*ModId*⟫`.module;`, where ⟪*ModId*⟫ stands for the lower-cased module identifier. The OMDoc DTD repository contains DTD driver files for all the sub-languages discussed in Section **??**, which contain the relevant module inclusion entity settings. These are contained in the files `omdoc-`⟪*SlId*⟫`.dtd`, where ⟪*SlId*⟫ stands for the sub-language identifier.

Except for their use in making the OMDoc DTD more manageable, DTD modules also allow to include OMDoc functionality into other document types, extending OMDoc with new functionality encapsulated into modules or upgrading selected OMDoc modules individually. To aid this process, we will briefly describe the module structure. Following [**AltBou:mox01**], DTD modules come in two parts, since we have inter-module recursion. The problem is for instance that the `omlet` element can occur in mathematical texts (`mtext`), but also contains `mtext`, which is also needed in other modules. Thus the modules cannot trivially be linearized. Therefore the DTD driver includes an entity file ⟪*ModId*⟫`.ent` for each module ⟪*ModId*⟫, before it includes the grammar rules in the element modules ⟪*ModId*⟫`.mod` themselves. The entity files set up parameter entities for the qualified names and content models that are needed in the grammar rules of other modules.

1.6.4 Namespace Prefixes for OMDoc elements

Document type definitions do not natively support XML namespaces. However, clever coding tricks allow them to simulate namespaces to a certain extent. The OMDoc DTD follows the approach of [**AltBou:mox01**] that parametrizes namespace prefixes in element names to deal gracefully with syntactic effects of namespaced documents like we have in OMDoc.

Recall that element names are **qualified name** s, i.e. pairs consisting of a namespace URI and a local name. To save typing effort, XML allows to abbreviate qualified names by namespace

[6]The process of DTD-normalization expands all parsed XML entities, and adds all default attribute values

declarations via `xmlns` pseudo-attribute: the element and all its descendants are in this namespace, unless they have a namespace attribute of their own or there is a namespace declaration in a closer ancestor that overwrites it. Similarly, a namespace abbreviation can be declared on any element by an attribute of the form `xmlns:nsa="nsURI"`, where `nsa` is a name space abbreviation, i.e. a simple name, and `nsURI` is the URI of the namespace. In the scope of this declaration (in all descendants, where it is not overwritten) a qualified name `nsa:n` denotes the qualified name `nsURI:n`.

The mechanisms described in [**AltBou:mox01**] provide a way to allow for namespace declarations even in the (namespace-agnostic) DTD setting simply by setting a parameter entity. If `NS.prefixed` is declared to be `INCLUDE`, using a declaration such as `<!ENTITY % NS.prefixed "INCLUDE">` either in the local subset of the `DOCTYPE` declaration, or in the DTD file that is including the OMDoc DTD, or the DTD modules presented in this appendix, then all OMDoc elements should be used with a prefix, for example `<omdoc:definition>`, `<omdoc:CMP>`, etc. The prefix defaults to `omdoc:` but another prefix may be declared by declaring in addition the parameter entity `omdoc.prefix`. For example, `<!ENTITY % omdoc.prefix "o">` would set the prefix for the OMDoc namespace to `o:`.

Note that while the Namespaces Recommendation [**BraHol:xmlns99**] provides mechanisms to change the prefix at arbitrary points in the document, this flexibility is not provided in this DTD (and is probably not possible to specify in any DTD). Thus, if a namespace prefix is being used for OMDoc elements, so that for example the root element is:

```
<omdoc:omdoc xmlns:omdoc="http://omdoc.org/ns">
...
</omdoc:omdoc>
```

then the prefix must be declared in the local subset of the DTD, as follows:

```
<!DOCTYPE omdoc:omdoc PUBLIC "−//OMDoc//DTD OMDoc V1.6//EN"
                             "http://www.omdoc.org/dtd/omdoc.dtd"
  [<!ENTITY % NS.prefixed "INCLUDE"><!ENTITY % omdoc.prefix "omdoc">]>
```

The OMDoc DTD references six namespaces:

language	namespace	prefix
MathML	`http://www.w3.org/1998/Math/MathML`	`m:`
OpenMath	`http://www.openmath.org/OpenMath`	`om:`
XSLT	`http://www.w3.org/1999/XSL/Transform`	`xsl:`
Dublin Core	`http://purl.org/dc/elements/1.1/`	`dc:`
Creative Commons	`http://creativecommons.org/ns`	`cc:`
OMDoc	`http://omdoc.org/ns`	`omdoc:`

These prefixes can be changed just like the OMDoc prefix above.

1.7 Validation with RelaxNG Schemata

RelaxNG [**Vlist:rng03**] is a relatively young approach to validation developed outside the W3C, whose XML schema paradigm was deemed overburdened. As a consequence, RelaxNG only concerns itself with validation, and leaves out typing, normalization, and entities. RelaxNG schemata come in two forms, in XML syntax (file name extension `.rng`) and in compact syntax (file name extension `.rnc`). We provide the RelaxNG schema [**OMDocRNC:URL**] as the normative validation schema for OMDoc. As compact syntax is more readily understandable by humans, we have reprinted it as the normative grammar for OMDoc documents in ?rnc?. Just as in the case for the OMDoc DTD, we provide schemata for the OMDoc sub-languages discussed in Section **??**. These are contained in the driver files `omdoc-`⟪*SlId*⟫`.rnc`, where ⟪*SlId*⟫ stands for the sub-language identifier.

There is currently no standard way to associate a RelaxNG schema with an XML document[7]; thus validation tools (see `http://relaxng.org` for an overview) have to be given a grammar as an explicit argument. One consequence of this is that the information that an OMDoc document is intended for an OMDoc sub-languages must be managed outside the document separately from the document.

There are various validators for RelaxNG schemata, the author uses the open-source xmllint [**Veillard:libxml2**] as a stand-alone tool, and the nXML mode [**nxml-mode:URL**] for the Emacs editor [**Stallman:em02**] for editing XML files, as it provides powerful RelaxNG-based editing support (validation, completion, etc.).

1.8 Validation with XML Schema

For validation[8] with respect to XML schemata (see [**XML:Schema**]) we provide an XML schema for OMDoc [**OMDocXSD:URL**], which is generated from the RelaxNG schema in ?rnc? via the trang system described above. Again, schemata for the sub-languages discussed in Section **??** are provided as `omdoc-`⟪*SlId*⟫`.rnc`, where ⟪*SlId*⟫ stands for the sub-language identifier.

To associate an XML schema with an element, we need to decorate it with an `xsi:schemaLocation` attribute and the namespace declaration for XML schema instances. In Listing 1.3 we have done this for the top-level `omdoc` element, and thus for the whole document. Note that this mechanism makes mixing XML vocabularies much simpler than with DTDs, that can only be associated with whole documents.

Listing 1.3: An XML document with an XML Schema.

```
<?xml version="1.0"?>
<omdoc xml:id="example.with.schema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
       xsi:schemaLocation="http://omdoc.org/ns
                           http://omdoc.org/xsd/omdoc.xsd">
 ...
</omdoc>
```

[7]In fact this is not an omission, but a conscious design decision on the part of the RelaxNG developers.
[8]There are many schema-validating XML parsers, the author uses the open-source xmllint [**Veillard:libxml2**].

Part II

Transforming OMDoc by XSLT Style Sheets

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 XSLT style sheets [**Clark:xslt99**] to produce the desired output formats. There are several high-quality XSLT transformers freely available including `saxon` [**saxon˙web**], `xalan` [**xalan˙web**], and `xsltproc`[**xsltproc˙web**]. Moreover, XSLT is natively supported by the newest versions of the browsers MS Internet Explorer [**ie˙web**] and Mozilla [**mozilla˙web**].

XSLT style sheets can be used for several tasks in maintaining OMDoc, such as for instance converting other XML-based input formats into OMDoc (e.g. `cd2omdoc.xsl` for converting OpenMath content dictionaries into OMDoc format), or migrating between different versions of OMDoc e.g. the style sheet `omdoc1.1adapt1.2.xsl` that operationalizes all the syntax changes from Version 1.1 of OMDoc to version 1.2 (see ?changelog? for a tabulation). We will now review a set of XSLT style sheets for OMDoc, they can be found in the OMDoc distribution (see Section 1.2) or on the web at [**OMDocXSL:URL**].

1.9 Extracting and Linking XSLT Templates

One of the main goals of content markup for mathematical documents is to be independent of the output format. In Chapter **??**, we have specified the conceptual infrastructure provided by the OMDoc language, in this section we will discuss the software infrastructure needed to transform OMDoc documents into various formats.

The `presentation` elements for symbols in OpenMath or Content-MathML formulae allow a declarative specification of the result of transforming expressions involving these symbols into various formats. To use this information in XSLT style sheets, the content of `presentation` elements must be transformed into XSLT templates, and these must be linked into the generic transformation style sheet. The OMDoc distribution provides two meta-style-sheets for these tasks.

The first one — `expres.xsl` — compiles the content of the `presentation` and `omstyle` elements in the source file into XSLT templates. The style sheet takes the parameter `report-errors`, which is set to '`no`' by default; setting it to '`yes`' will enable more verbose error reports. The OMDoc distribution provides Unix `Makefiles` that specify the target ⟪*base*⟫`-tmpl.xsl` for each OMDoc file ⟪*base*⟫`.omdoc`, so that the templates file can be generated by typing `make` ⟪*base*⟫`-tmpl.xsl`. Note that `expres.xsl` follows the references in the `ref` elements (it `ref`-normalizes the document see Section **??**) before it generates the templates[9].

The second style sheet — `exincl.xsl` — generates link table for a specific OMDoc document. This style sheet `ref`-normalizes the document and outputs an XSLT style sheet that includes all the necessary template files. `expres.xsl` takes two parameters: `self` is the name of the source file name itself[10]. The `Makefiles` in the OMDoc distribution specify the target ⟪*base*⟫`-incl.xsl`, so that the link table can be generated by typing `make` ⟪*base*⟫`-incl.xsl`.

Let us now consider the example scenario in Figure 1.1: Given an OMDoc document ⟪*document*⟫`.omdoc` that uses symbols from theories `a`, `b`, `c`, `d`, and `e` which are provided by the OMDoc documents ⟪*background*⟫`.omdoc`, ⟪*special*⟫`.omdoc` and ⟪*local*⟫`.omdoc`, we need to generate the template files ⟪*background*⟫`-tmpl.xsl`, ⟪*special*⟫`-tmpl.xsl`, and ⟪*local*⟫`-tmpl.xsl` (via `expres.xsl`) as well as ⟪*document*⟫`-incl.xsl` (via `exincl.xsl`). Now it is only necessary to include the link table ⟪*document*⟫`-incl.xsl` into a generic transformation style sheet to specialize it with the notation information specified in the `presentation` elements in theories `a`, `b`, `c`, `d`, and `e`.

The transformation architecture based on the `Makefiles` provided with the OMDoc distribution does the linking by creating a specialized style sheet ⟪*document*⟫`2html.xsl` that simply includes the generic OMDoc transformation style sheet `omdoc2html.xsl` (see Section 1.11), and the style sheet ⟪*document*⟫`-incl.xsl`. Changing this simple control style sheet allows to add site- or language-specific templates (by adding them directly or including respective style sheets). An analogous processing path leads ⟪*document*⟫`.tex` using `omdoc2tex.xsl` and from there to PDF using tools like `pdflatex`.

Other processing architectures may be built up using on-demand technologies, e.g. servlets, mediators, or web services, but will be able to follow the same general pattern as our simpleminded implementation in `Makefiles`.

We will make use of this general architecture based on extraction and linking via XSLT style sheets in the transformation of OMDoc documents below.

1.10 OMDoc Interfaces for Mathematical Software Systems

One of the original goals of the OpenMath, Content-MathML and OMDoc languages is to provide a communication language for mathematical software systems. The main idea behind this is to supply systems with interfaces to a universally accepted communication language standard

[9]In the current implementation, `expres.xsl` generates one large template that combines the XSLT code for all target formats. This simplifies the treatment of the default presentations as requested by the specification in Section **??**, but hampers mixing presentation information from multiple sources. An implementation based on modes would probably have advantages in this direction in the long run.

[10]For some reason XSLT processors do not provide access to this information portably.

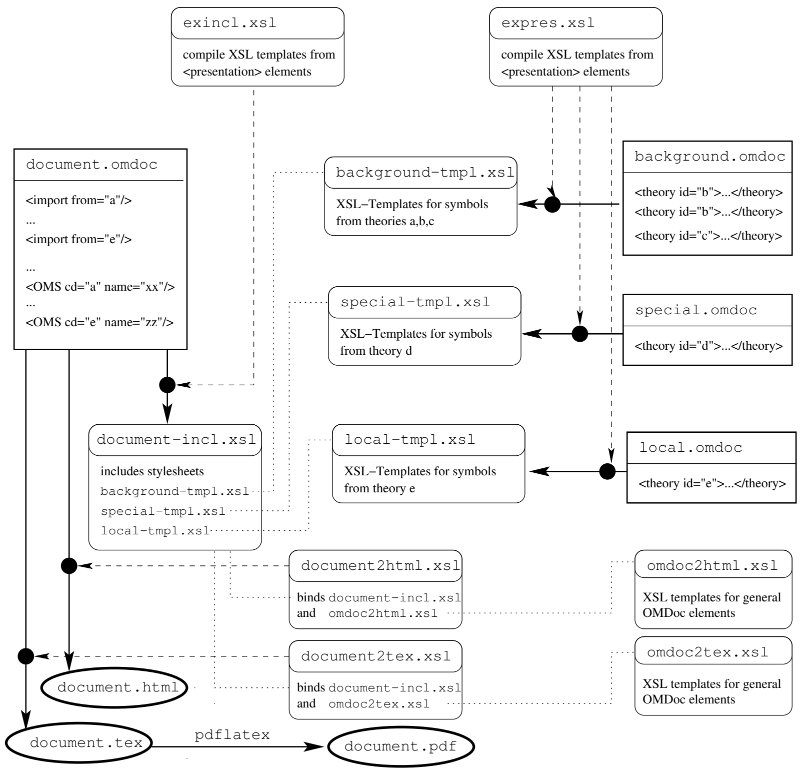
exincl.xsl
compile XSL templates from <presentation> elements
expres.xsl
compile XSL templates from <presentation> elements
document.omdoc
<import from="a"/>
...
<import from="e"/>
...
<OMS cd="a" name="xx"/>
...
<OMS cd="e" name="zz"/>
background-tmpl.xsl
XSL–Templates for symbols from theories a,b,c
background.omdoc
<theory id="b">...</theory>
<theory id="b">...</theory>
<theory id="c">...</theory>
special-tmpl.xsl
XSL–Templates for symbols from theory d
special.omdoc
<theory id="d">...</theory>
document-incl.xsl
includes stylesheets
background-tmpl.xsl
special-tmpl.xsl
local-tmpl.xsl
local-tmpl.xsl
XSL–Templates for symbols from theory e
local.omdoc
<theory id="e">...</theory>
document2html.xsl
binds document-incl.xsl and omdoc2html.xsl
omdoc2html.xsl
XSL templates for general OMDoc elements
document2tex.xsl
binds document-incl.xsl and omdoc2tex.xsl
omdoc2tex.xsl
XSL templates for general OMDoc elements
document.html
document.tex
pdflatex
document.pdf

Figure 1.1: The OMDoc presentation Process

(an interlingua), and so achieve interoperability for n systems with only $2n$ translations instead of n^2. As we have seen in Section **??**, OpenMath and Content-MathML provide a good solution at the level of mathematical objects, which is sufficient for systems like computer algebra systems. OMDoc adds the level of mathematical statements and theories to add support for automated reasoning systems and formal specification systems.

To make practical use of the OMDoc format as an interlingua, we have to support building OMDoc interfaces. An XSLT style sheet is a simple way to come up with (the input half) of an OMDoc interface. A more efficient way would be to integrate an XML directly into the system (suitable XML parsers are readily available for almost all programming languages nowadays).

Usually, the task of writing an XSLT style sheet for such a conversion is a relatively simple task, since the input language of most mathematical software system is isomorphic to a subset of OMDoc. This suggests the general strategy of applying the necessary syntax transformations (this has to be supplied by the style sheet author) on those OMDoc elements that carry system-relevant information and transforming those that are not (e.g. Metadata and `CMP` elements for most systems) into comments. Much of the functionality is already supplied by the style sheet `omdoc2sys.xsl`, which need only be adapted to know about the comment syntax.

The task of translating an OMDoc document into system-specific input has two sub-tasks. We will discuss them using the concrete example of the `omdoc2pvs.xsl` style sheet that transforms OMDoc documents to the input language of the Pvs theorem prover [**OwRu92**]: The first task is to translate elements at the statement- and theory level to the input language this is hand-coded by supplying suitable templates for the OMDoc statement and theory elements in an extension of the `omdoc2sys.xsl` style sheet. The second task is to translate the formulae to the input language. Here, the system usually has a particular way of expressing complex formulae like function applications and binding expressions; in the concrete case of Pvs, function application uses a prefix function argument syntax, and n-ary binding expressions, where the scope is separated by a colon from the variable list. This information must also be encoded in respective templates for the `om:OMA`, `om:OMBIND`, `om:OMV` elements from OPENMATH and the `m:apply` and `m:ci` from Content-MATHML. For the symbol elements, we have to distinguish two cases: the predefined symbols of the system language and the object symbols that are introduced by the user to formalize a certain problem. In both cases, the transformation procedure needs input on how these symbols are to be represented in the system language. For the object symbols we assume that there are suitable `theory` structures available, which declare them in `symbol` elements, thus we can assume that these `theory` structures also contain `use` elements with appropriate `format` attribute in the `presentation` elements for those symbols that need special representations in the system language. For the predefined symbols of the system language, we assume the same. To be able to transform an OMDoc document into system input, we need a language definition theory, i.e. an OMDoc document that contains a `theory` which provides `symbol`s for all the predefined words of the system language. This theory must also contain `presentation` elements with `use` children specialized the input formats of all systems targeted for communication.

Listing 1.4: A `symbol` in a Language Definition Theory

```
<symbol name="sigmatype">
 <metadata>
   <dc:description>
     The dependent function type constructor is a binding operator. The source type is
     the type of the bound variable X, the target type is represented in the body.
   </dc:description>
 </metadata>
</symbol>

<presentation xml:id="pr-sigmatype" for="#sigmatype" role="binding">
  <style format="pvs">
    <text>[</text>
    <recurse select="*[2]/*"/><text> -&gt; </text><recurse select="*[3]"/>
    <text>]</text>
  </style>
  <style format="nuprl">
    <recurse select="*[2]/*"/><text> -&gt; </text><recurse select="*[3]"/>
  </style>
</presentation>
```

The other direction of the translation needed for communication is usually much more complicated, since it involves parsing the often idiosyncratic output of these systems. A better approach is to write specialized output generators for these systems that directly generate OMDoc representations. This is usually a rather simple thing to do, if the systems have internal data structures that provide all the information required in OMDoc. It is sometimes a problem with these systems that they only store the name of a symbol (logical constant) and not its home theory. At other times, internal records of proofs in theorem provers are optimized towards speed and not towards expressivity, so that some of the information that had been discarded has to be recomputed for OMDoc output.

One of the practical problems that remains to be solved for interfaces between mathematical software systems is that of semantic standardization of input languages. For mathematical objects, this has been solved in principle by supplying a theory level in the form of OPENMATH or OMDoc content dictionaries that define the necessary mathematical concepts. For systems like theorem provers or theory development environments we need to do the same with the logics underlying these systems. For an effort to systematize logics into a hierarchy that fosters reuse and communication of systems, based on a series of experiments of interfacing with the theorem proving systems ΩMEGA [**BenzmuellerEtAl:otama97**], INKA [**HuSe:itng96**], PVS [**OwRu92**], λ*Clam* [**RicSmaGre:ppihol98**], TPS [**AnBi:tatps96**] and COQ [**CoqManual**] see Section **??**

1.11 Presenting OMDoc to Humans

We will now discuss the software infrastructure needed to transform OMDoc documents into human-readable form in various formats. We speak of of OMDoc **presentation** for this task.

Due to the complex nature of OMDoc presentation, only part of it can actually be performed by XSLT style sheets. For instance, sub-tasks like reasoning about the prior knowledge of the user, or her experience with certain proof techniques is clearly better left to specialized applications. Our processing model is the following: presenting an OMDoc is a two-phase process.

The first phase is independent of the final output format (e.g. HTML, MathML, or LaTeX) and produces another OMDoc representation specialized to the respective user or audience, taking into account prior knowledge, structural preferences, bandwidth and time constraints, etc. This phase usually generates a narrative-structured document from a knowledge-centered one.

The second phase is a formatting process that can be extracted by XSLT style sheets that transforms the resulting specialized document into the respective output format with notational- and layout preferences of the audience. We will only discuss the second one and refer the reader for ideas about the first process to systems like P.rex [**Fiedler:ddaoeo01**; **FiedlerHoracek:aietlp01**].

The presentation of the OMDoc document elements and statements is carried out by the style sheets `omdoc2html.xsl` for XHTML, `omdoc2html.xsl` for XHTML+MathML and `omdoc2tex.xsl` for LaTeX. These style sheets are divided into files according to the OMDoc modules and share a large common code base `omdoc2share.xsl`, basically the first two include the latter and only redefine some format-specific options. For instance, `omdoc2share.xsl` supplies an infrastructure for internationalization introduced in Section ??. This allows to generate localized presentations of the OMDoc documents, if enough information is present in the multilingual groups of `CMP` elements. `omdoc2share.xsl` takes a parameter `TargetLanguage`, whose value can be a whitespace-separated preference list of ISO 639 norm two-letter country codes. If `TargetLanguage` consists of a single entry, then the result will only contain this language with gaps where the source document contains no suitable `CMP`. Longer `TargetLanguage` preference lists will generally result in more complete, but multilingual documents. Apart from the language-specific elements in the source document, localization also needs to know about the presentation of certain keywords used in OMDoc markup, e.g. the German "Lemma" and the French "Lemme" for `<assertion type="lemma">`. This information is kept in the keyword table `lib/locale.xml` in the OMDoc distribution, which contains all the keywords necessary for presenting the OMDoc elements discussed so far. An alternative keyword table can be specified by the parameter `locale`.

Index