

Felix Hauser, Philip Schaffhauser

***Database-driven XML-enabled
Bibliography Management System***

*Diploma Thesis DA-2003.05
Winter Term 2002/2003*

*Tutor:
Dr. Erik Wilde*

*Supervisor:
Prof. Dr. Bernhard Plattner*

25.3.2003

Abstract

In this diploma thesis, a centralized bibliography management system is designed and implemented. We describe an architecture that not only allows users to share their bibliographic information, but also enables them to keep their personal style of creating references by supporting a highly extensible format for bibliographic data. Furthermore, the management system offers flexible search tools, and implements a powerful concept for exporting and thus reusing references. Our architecture is based on XML technologies (most notably DOM, XML Schema, Schematron, and XSLT) and uses Apache's Web server in conjunction with PHP and MySQL.

Contents

1	Introduction	1
2	Requirements	3
3	Concept	4
3.1	Data Storage	5
3.1.1	Document-based Approach	5
3.1.2	Native XML Database Approach	5
3.1.3	Relational Database Approach	5
3.2	Platform	6
3.3	Design	6
3.3.1	Import	6
3.3.2	Search	7
3.3.3	Export	7
3.3.4	Normal Users versus Administrators	8
4	Implementation	9
4.1	BibXML Format	9
4.2	Validation of BibXML Documents	17
4.2.1	Schema Languages	18
4.2.2	BibSchema	21
4.2.3	Dependencies between BibSchema, XML Schema, and Schematron Schemas	27
4.2.4	Validating and Importing BibSchema Schemas	32
4.2.5	Process of Validating BibXML Documents	34
4.3	Database Structure	37
4.4	Data Import	41
4.5	Search	42
4.6	Selection	44
4.7	Export Management	44

4.8	Macro Management	46
4.9	User Management and System Access	47
5	Results	49
6	Future Work	50
7	Conclusions	52
A	User Guide	54
A.1	Login	54
A.2	Overview	54
A.3	Search	55
A.3.1	Simple Search	55
A.3.2	Advanced Search	56
A.3.3	Regular Expressions	57
A.3.4	Search Results	58
A.4	Selection	59
A.5	Schema Overview	60
A.6	Data Import	60
A.7	Macro Management	62
B	Administrator Guide	63
B.1	Overview	63
B.2	Schema Management	64
B.3	Export Management	64
B.4	User Management	66
B.5	MySQL Admin	67

C	Technical Documentation	69
C.1	System Setup	69
C.1.1	Apache	70
C.1.2	PHP	70
C.1.3	MySQL	71
C.1.4	Saxon	71
C.1.5	Xerces	71
C.1.6	Java 2	71
C.1.7	Bibliography Management System	72
C.2	System Maintenance	72
C.3	Most Common Extensions	73
C.3.1	BibXML Extensions	73
C.3.2	Export Filters for BIBTEX	74
C.4	Built-in XML Documents	75
C.4.1	import	75
C.4.2	schema	76
C.5	PHP Functions	78
C.5.1	export_detail.php	78
C.5.2	export_management.php	78
C.5.3	fast_search.php	79
C.5.4	import.php	79
C.5.5	include.php	82
C.5.6	include_export.php	85
C.5.7	include_fast_search.php	85
C.5.8	include_transform.php	87
C.5.9	index.php	87
C.5.10	macro_management.php	88
C.5.11	schema_detail.php	88
C.5.12	schema_management.php	88
C.5.13	schema_overview.php	88

C.5.14 schema/validation.php	88
C.5.15 schema/xml_printer.php	93
C.5.16 selection.php	94
C.5.17 user.php	94
C.5.18 welcome.php	94

D CD ROM	95
-----------------	-----------

List of Figures

1	Conceptual Overview	4
2	System Architecture	7
3	Two-step Validation Process	17
4	Classification of XML Schema Languages	18
5	BibSchema, XML Schema, and Schematron	20
6	Redefined Validation Process	20
7	Validation Process Overview Indicating File Paths	29
8	Dependencies between BibSchema and XML Schema Schemas	30
9	Substitution Group 1: Entry Elements	31
10	Substitution Group 2: Field Elements	31
11	Substitution Group 3: Special Item Elements	32
12	Head Element Declarations in <i>xmlSchemaBase.xsd</i>	35
13	Member Element Declarations	36
14	Dependencies between BibSchema and Schematron Schemas	36
15	Conceptual Database Schema	37
16	Data Import	41
17	Export Mode Management Table	45
18	User Management Tables	48
19	Login Page	54
20	Overview Page	55
21	Simple Search Interface	55
22	Advanced Search Interface	56
23	Search Results	58
24	Selection	59
25	Schema Overview	60
26	Start of Data Import	61
27	Data Import Confirmation Dialogue	61
28	Macro Management	62
29	Administrator Overview Page	63

30	Schema Management	65
31	Export Management	66
32	User Management	67
33	phpMyAdmin	68

List of Tables

1	BIB _T E _X Example	2
2	BIB _T E _X Example	9
3	BibXML Document Skeleton	10
4	BibXML Document Using Entry and Field Elements	10
5	BibXML Document With Two References	12
6	Improved BibXML Document Using Cross References	13
7	BibXML Macros	13
8	BibXML Document Defining Two Macros	14
9	BibXML Document Referencing Two Macros	14
10	System Information of a BibXML Entry	15
11	BibSchema Skeleton	21
12	Definition <code>month</code> Field Element	22
13	Definition of Person and Special Text Field Elements	23
14	Two Ways of Using a Person Field Element	23
15	Special Text Field	24
16	Definition of Special Item <code>tex</code>	25
17	Definition of Repeatable <code>author</code> Field	25
18	Definition of <code>book</code> Entry Element	26
19	Definition of a <code>book</code> Entry Element by Referencing a Field Element (<code>ae:title</code>) from Another Namespace	27
20	BibSchema Document To Be Transformed	33
21	Transformed Schematron Schema	33
22	Temporary XSLT Importing All Mode Stylesheets	45
23	BIB _T E _X Translation Table (partial)	46
24	XSLT Stylesheet <i>import/special.xsl</i>	47
25	Regular Expression Syntax	57
26	File Extensions	75
27	CD ROM Contents	95

1 Introduction

A bibliography, sometimes also referred to as *references*, *works cited*, or *works consulted*, is usually thought of as an organized listing of sources that the author consulted during the research and writing process. A bibliography can include a variety of different resources, such as books, articles, reports, interviews, or even non-print sources like Web sites or video recordings. Each source in the bibliography is represented by its citation information (e.g. author(s), date of publication, title, and publisher's name and location), which highly depends on the type of the source. The primary function of bibliographic citations is to assist the reader in finding the sources used in the writing of a work.

Over the years, researchers usually gather a considerable amount of bibliographic information, which often includes several thousand cited resources. Since collecting such an amount of data is a very time consuming process, members of an institution would be well advised to store their bibliographic data in a centralized management system in order to make it accessible to other employees. Assuming the management system provides efficient search capabilities, the users of such a system could benefit from the availability of an extensive amount of bibliographic data. Furthermore, the system could provide the users with a Web-based interface for manipulating data, automatic backup, and very flexible ways of exporting data.

Nevertheless, in an institution like the ETH Zürich, many employees manage their own private bibliographies, because they are not willing to stick to predefined formats and other limitations, which usually come with a centralized system. A typical example for per-user bibliographies are BIBTEX [5] files. BIBTEX is a widespread format for bibliographies on the Internet. It is an extensible format in the sense that it defines several standard entry and field types and also accepts additional user-defined entry and field types. Many people make extensive use of this extensibility by defining their own private entry and field types in order to store various kinds of information about their resources. Listing 1 illustrates this situation with an example of a BIBTEX entry that uses the standard entry type `misc`, several standard fields, and the two non-standard fields `uri` and `topic`. In this example, the `uri` field simply contains the resource's URI, while the `topic` field contains structured information. It contains a list of weighted references to a topic map of Web technologies¹, thus categorizing the resource according to these topics.

¹Available online at <http://wildesweb.com/glossary/>

```
@misc{xns,  
  author =      "Tim Bray and Dave Hollander and Andrew Layman",  
  title =      "Namespaces in XML",  
  howpublished = "W3C, REC-xml-names-19990114",  
  month =      jan,  
  year =       1999,  
  uri =        "http://www.w3.org/TR/1999/REC-xml-names-19990114",  
  topic =      "xml[0.8] xmlns[1]" }
```

Listing 1: BIB_TE_X Example

The consequence of the widespread use of extensible formats like BIB_TE_X is that if a management system should bring the members of an institution to store their bibliographic data in a centralized repository, it is an imperative for this system to support format extensions of the kind mentioned above.

2 Requirements

It was the goal of this diploma thesis to design and implement a centralized bibliography management system using XML technologies. This thesis should motivate members of institutions such as the ETH Zürich to give up their established habit of maintaining their own private bibliographies and create an incentive to store bibliographic data in a centralized system in order to make it accessible to other employees. To achieve this goal, the system had to comply with the following requirements:

- Besides making bibliographic information accessible to the whole institution, the system should enable members of the institution to retain their personal style of creating their bibliographies (see Listing 1). Thus, the system should support a format for bibliographies that can be extended by each user in an easy but flexible way.
- Furthermore, the bibliography management system should allow users to still handle their bibliographic data as their intellectual property. This requirement should be satisfied by granting users access rights.
- Users of the system should have the possibility to make use of flexible query tools in order to find specific data. Special attention should be given to the performance of such query tools. They should be able to cope with thousands of works cited within a reasonable time.
- A powerful concept for exporting references should be designed and implemented. The user should be presented with very flexible and extensible ways of exporting the data. Therefore, the ability to install pluggable export filters should be provided at user level. In a first step, export filters for BIBTEX and XML (in order to reuse the data) should be implemented.
- All services of the bibliography management system should be easily accessible via Web-based interfaces.
- Last but not least, it was an imperative to develop a system that did not require the operator to buy any licenses. This requirement implied the exclusive use of open source technologies.

3 Concept

Given the requirements described above, the following concept was developed: Firstly, to facilitate user-friendly, software-, and location-independent system access, a Web interface should be provided. Secondly, we needed a Web server and an appropriate programming language as a platform for all operations to be performed. And finally, a place to store bibliographic and other data was required. These considerations lead to the rough concept depicted in Figure 1.

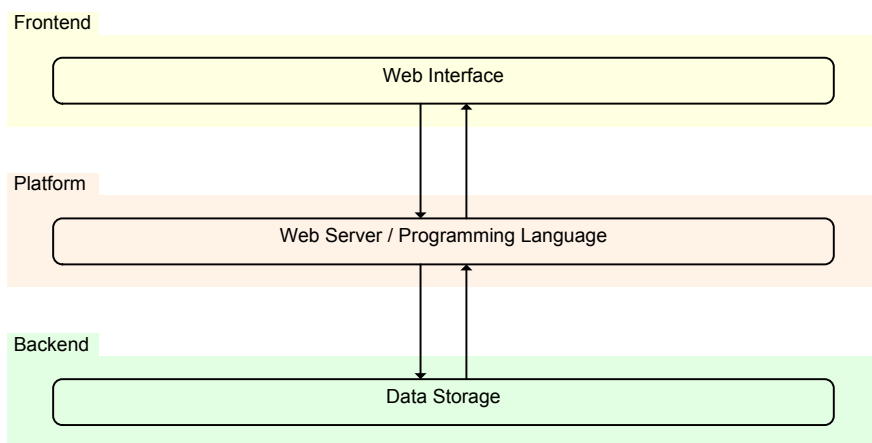


Figure 1: Conceptual Overview

We decided to make XML [1] play an important role as data exchange format in our system, especially in all import and export functions but also within the system. This not only because XML is ideal to manage data in a structured and flexible way, but also because of the large and still growing collection of accompanying tools and technologies and the broad support in the entire industry. Particularly schemas (e.g., W3C's XML Schema² [6], Schematron³) for validating XML documents and XSLT stylesheets [3] for transforming XML documents are of great use for many tasks.

²Information on XML Schema: <http://www.w3.org/XML/Schema/>

³Information on Schematron: <http://www.ascc.net/xml/resource/schematron/>

3.1 Data Storage

3.1.1 Document-based Approach

Several approaches for storing bibliographic data in the system are imaginable. A very simple possibility would be a document-based approach where all bibliographic data is stored in a single or multiple XML documents. We will subsequently refer to such an XML representation of bibliographic data as *BibXML* (more information is given in Section 4.1). XML technologies would be used to access and modify the documents. As such BibXML documents can become quite large (several megabytes), serious concerns about performance, scalability, and multi-user access arise for this approach.

3.1.2 Native XML Database Approach

A database would certainly solve these scalability problems. And as XML is already used as data exchange format in our system, an XML database would certainly bring some advantages over a relational database like easy document import and the possibility to query XML substructures. Unfortunately, most native XML databases are commercial and those that are not can currently not compete with the large functionality of relational databases. Two examples of open source native XML databases are eXist⁴ and Apache's Xindice⁵.

3.1.3 Relational Database Approach

Another approach is to map the BibXML format to a relational database. The advantage of this solution is that these databases are very sophisticated and stable, have a large functionality, and are very well supported by other software. One disadvantage is that most noncommercial relational databases do currently not support queries to XML substructures. A very popular open source database is MySQL⁶ [8]. MySQL offers good performance, stability, and is very well supported by many programming languages. These evaluations lead to the decision to pursue a relational database approach and choose MySQL to manage the bibliographic data.

⁴eXist homepage: <http://exist.sourceforge.net/>

⁵Xindice homepage: <http://xml.apache.org/xindice/>

⁶MySQL homepage: <http://www.mysql.com/>

3.2 Platform

The choice of MySQL was accompanied by the decision to use Apache's⁷ Web server and PHP⁸ as programming language. This because these three components plus Linux constitute the popular open source Web platform LAMP (Linux/Apache/MySQL/PHP) and work very well together. (We are using Windows instead of Linux, but since there are no Windows-specific parts in our system, porting it to another platform is trivial.)

PHP is a widely-used scripting language that is especially suited for Web development. PHP can be plugged into Apache's HTTP server as a module and then executes PHP code that is embedded into HTML documents. PHP provides special function classes for many technologies, including MySQL, XML, XSLT and session management, is quite easy to learn but still very powerful. This makes it the ideal programming language for our project.

3.3 Design

In this section the design of the system is explained. Please use Figure 2 as a visualization for all explanations. A more detailed description of the design and the implementation of the system follows in Chapter 4.

3.3.1 Import

To import bibliographic data into the management system, users have to write a BibXML representation of their data, and pass it to the system using the *Data Import* component of the Web interface. The first task of the management system is to validate the imported document against relevant schemas in order to ensure that the document complies with the BibXML format. Users have the possibility to extend the BibXML format accepted by the management system by defining their own format extensions in specialized schema documents. These additional user-defined schemas are imported into the system using the *Schema Management* component. The system then validates the submitted schema documents against built-in schemas, and finally stores them in the filesystem for future use. From then on, BibXML documents using these format extensions can be imported through the *Data Import* component, the *Import Engine* validates them using predefined and user-defined schemas, and then inserts the data into the database.

⁷Apache homepage: <http://www.apache.org/>

⁸PHP homepage: <http://www.php.net/>

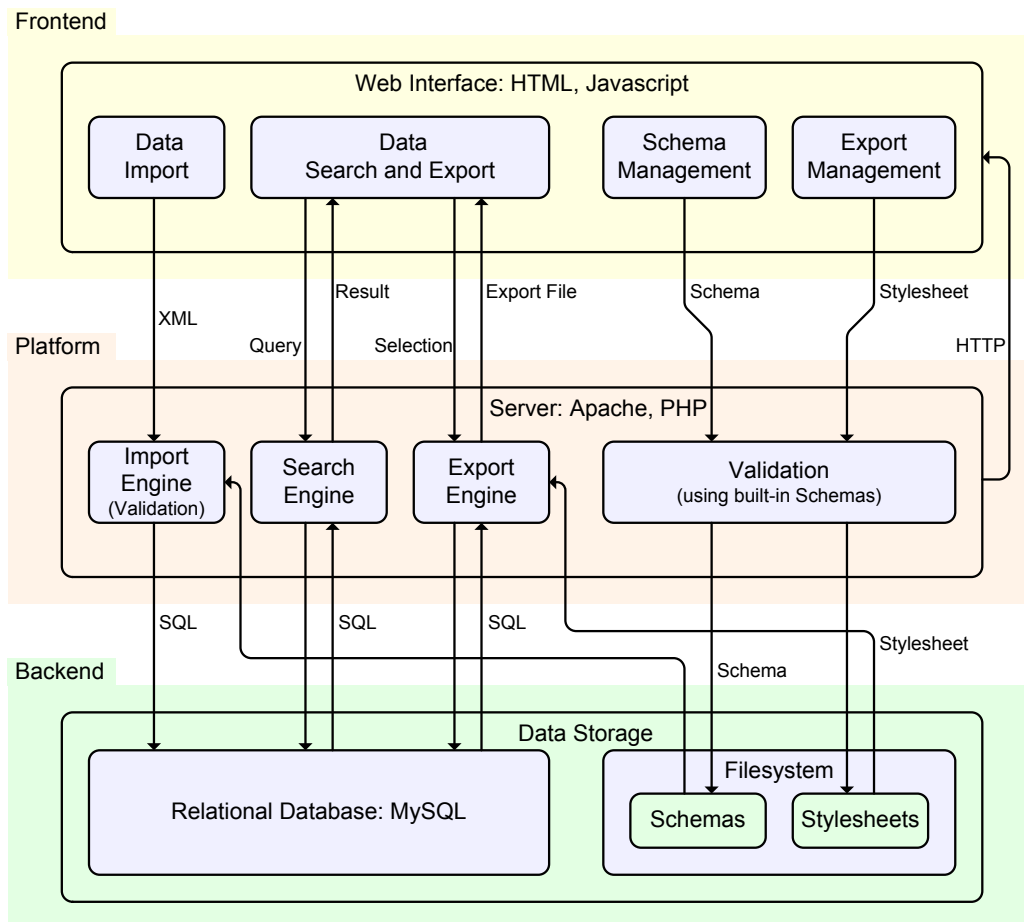


Figure 2: System Architecture

3.3.2 Search

The bibliographic data in the database can be searched through the search component of the Web interface. The *Search Engine* converts the search query from the Web Interface into an SQL query for the database, takes the results, and presents them in the Web interface. These search results can be collected in a shopping-cart-like system for later management or export.

3.3.3 Export

The basic export back into the BibXML format is quite simple and straight forward. The *Export Engine* takes the selection made by the user, collects

the data from the database, creates a BibXML document out of the data, and makes the resulting document available in the Web interface.

The system can also be extended with export filters for other user-defined formats. For this purpose, user-defined stylesheets can be uploaded using the *Export Management* component. These stylesheets are checked for XSLT 1.0 conformance, and then stored in the system for future use. From then on, the system offers data export into this user-defined format as an option.

In addition to the stylesheets, a special translation table can be uploaded for each user-defined export format. Such a translation table is used to convert special characters into their required representation in the export format. The translation table can be uploaded through the Web interface, is validated, and then stored in the system.

The export into user-defined formats is realized by the *Export Engine* by applying the corresponding stylesheet and translation table to the BibXML file generated during basic export described above. The export into the popular BIBTEX [5] format is already implemented as a first extension.

3.3.4 Normal Users versus Administrators

The bibliography management system distinguishes between *normal users* and *administrators* of the system. The majority of the users are normal users. They are restricted to use the system to import, search and export bibliographic data. As opposed to normal users, administrators are designated users having unrestricted access to the system. Besides having the possibility to import, search and export bibliographic data, they are responsible for the availability of the system's services: This includes tasks like importing schema extensions and output filters into the system, or managing the users of the system. A detailed description of the normal users' and the administrators' possibilities and duties can be found in Appendices A, B, and C.2. Details about the implementation of user management and system access are included in Section 4.9.

4 Implementation

In the following sections, the components of our bibliography management system and their implementation are explained. Important features will be discussed in detail, while the reader interested in all details of the system should additionally consult the User Guide in Appendix [A](#), the Administrator Guide in Appendix [B](#), and the Technical Documentation in Appendix [C](#).

4.1 BibXML Format

To import bibliographic data into the management system, users have to write an XML representation of their data that the system accepts and is able to handle. Throughout this thesis, such an XML representation of bibliographic data is called a *BibXML* document. In this section, the format of BibXML documents is described in detail. Readers who are familiar with `BIBTEX` will recognize the similarities between the `BIBTEX` and the BibXML formats. This is because the `BIBTEX` format was used as a template when designing the BibXML format.

Before going into format details, we introduce some naming conventions from the `BIBTEX` community: References in a bibliography are called *entries* of the bibliography, and the *fields* of such an entry correspond to the citation information of the entry. Additionally, each entry or field is said to have a certain *type*. Using this terminology, Listing [2](#) can be described as showing an example of a `BIBTEX` entry, which uses the entry type `proceedings` and four fields of field types `title`, `address`, `year`, and `booktitle`.

```
@proceedings{iwaca92,
  title = "Proceedings of the IWACA Workshop",
  address = "Munich, Germany",
  year = 1992,
  booktitle = "Proceedings of the IWACA Workshop"
}
```

Listing 2: `BIBTEX` Example

An Example BibXML Document

The BibXML format can best be described using a concrete example. Step by step, we will compose a BibXML document and introduce the different

characteristics of this format.

BibXML uses XML syntax to describe the entries and fields of a bibliography. The root element of a BibXML document is a `bibliography` element in the `http://bibtextml.org/base` namespace. By convention, this namespace is mapped to the prefix `b`, but users are free to pick another prefix if they prefer. From this point forward, it is assumed that the prefix `b` is mapped to the `http://bibtextml.org/base` namespace. Entries of the bibliography are grouped in a `b:entries` element, which is included directly under the `b:bibliography` element. This is depicted in Listing 3.

```
<?xml version="1.0"?>
<b:bibliography xmlns:b="http://bibtextml.org/base">
  <b:entries>
    <!-- entries go here -->
  </b:entries>
</b:bibliography>
```

Listing 3: BibXML Document Skeleton

We can now take the next step in terms of composing a bibliography, by adding entries and fields to our document. Each entry or field is represented by a certain element depending on the type of the entry or the field. Furthermore, field elements are contained directly within the entry element they are describing. Applying this concept, the `BIBTEX` entry in Listing 2 is represented in BibXML as shown in Listing 4.

```
<?xml version="1.0"?>
<b:bibliography xmlns:b="http://bibtextml.org/base"
  xmlns:s="http://bibtextml.org/standard">
  <b:entries>
    <s:proceedings key="iwaca92">
      <s:title>Proceedings of the IWACA Workshop</s:title>
      <s:address>Munich, Germany</s:address>
      <s:year>1992</s:year>
      <s:booktitle>Proceedings of the IWACA Workshop</s:booktitle>
    </s:proceedings>
  </b:entries>
</b:bibliography>
```

Listing 4: BibXML Document Using Entry and Field Elements

At this point, it is interesting to see that the entry and field elements are not in the `http://bibtexml.org/base` namespace but in a new namespace (`http://bibtexml.org/standard`). The reason for the introduction of a new namespace is the following: As stated in Chapter 2, the management system has to support a format for bibliographies that can be extended by each user. Concretely, this means that users should have the possibility to define their own entry and field elements by extending the BibXML format accordingly. And this is exactly how the entry and field elements used in Listing 4 were added to the BibXML format: A user defined the entry element `s:proceedings` and the four field elements `s:title`, `s:address`, `s:year` and `s:booktitle` in the namespace `http://bibtexml.org/standard`, and imported this definitions into the system in order to extend the BibXML format with new elements. And we used this extension to describe our proceedings reference. How exactly this format extension works is irrelevant at this point. This will be the topic of Section 4.2. For now, it is only important to keep in mind that the BibXML format can be extended by user-defined entry and field elements. There is one more comment to be made concerning Listing 4: The order of field elements inside the corresponding entry element (as well as the order of entry elements inside the `b:entries` element) is irrelevant.

In a next step, we add a second entry to the bibliography. This time, we want to reference an article that is part of the already referenced proceedings of the IWACA workshop. To add this reference, we make use of the appropriate user-defined entry (`s:inproceedings`) and field (`s:author`, `s:title`, `s:booktitle`, `s:pages`, `s:address`, `s:year`) elements in the `http://bibtexml.org/standard` namespace. Listing 5 shows the resulting BibXML document with the two entries.

There are two things to be noticed about this BibXML document: First, each entry element needs to have a `key` attribute. This mandatory attribute needs to have a unique value among all entries of the system (entries in the same document as well as entries which have already been imported into the system). This key uniquely identifies each entry. Second, three fields (`s:booktitle`, `s:address`, and `s:year`) of the `wea92` entry are equal to fields of the `iwaca92` entry with respect to their type and their value. This equality of certain fields is forced by the fact that the `wea92` entry references an article that is part of the proceedings of a workshop that itself is referenced by the `iwaca92` entry. This forced equality of fields becomes problematic when we want to change the values of such fields. Suppose we wanted to complete the address of the workshop with the postal code of Munich, the street name, and the house number. We would have to change the value of the `s:address` field of the `iwaca92` entry as well as the `s:address` fields of all `s:inproceedings`

```

<?xml version="1.0"?>
<b:bibliography xmlns:b="http://bibtexml.org/base"
  xmlns:s="http://bibtexml.org/standard">
  <b:entries>
    <s:proceedings key="iwaca92">
      <s:title>Proceedings of the IWACA Workshop</s:title>
      <s:address>Munich, Germany</s:address>
      <s:year>1992</s:year>
      <s:booktitle>Proceedings of the IWACA Workshop</s:booktitle>
    </s:proceedings>
    <s:inproceedings key="wea92">
      <s:author>
        <b:bibperson>
          <b:firstname>Alfred</b:firstname>
          <b:lastname>Weaver</b:lastname>
        </b:bibperson>
      </s:author>
      <s:title>The Xpress Transfer Protocol</s:title>
      <s:booktitle>Proceedings of the IWACA Workshop</s:booktitle>
      <s:pages>253-259</s:pages>
      <s:address>Munich, Germany</s:address>
      <s:year>1992</s:year>
    </s:inproceedings>
  </b:entries>
</b:bibliography>

```

Listing 5: BibXML Document With Two References

entries referencing that workshop. This situation can be avoided by cross referencing the `iwaca92` entry from the `wea92` entry as shown in Listing 6.

The special `crossref` attribute tells the system that the `wea92` entry inherits any fields it is missing from the entry it references, `iwaca92`. In this case, it inherits the three fields `s:address`, `s:year`, and `s:booktitle`. Note that, at least for the most common `BIBTEX` styles, the `booktitle` information is irrelevant for the proceedings references. The `s:booktitle` field appears in the `iwaca92` entry only so that the entries that cross reference it may inherit the field. No matter how many articles from this workshop exist in the system, this `s:booktitle` field needs to appear only once. Finally, it is important to know that nesting cross references is not allowed. This means that a cross referencing entry must not be cross referenced itself.

The BibXML format also offers a macro mechanism. Macros can be used as shown in Listing 7.

```

<?xml version="1.0"?>
<b:bibliography xmlns:b="http://bibtexml.org/base"
  xmlns:s="http://bibtexml.org/standard">
  <b:entries>
    <s:proceedings key="iwaca92">
      <s:title>Proceedings of the IWACA Workshop</s:title>
      <s:address>Munich, Germany</s:address>
      <s:year>1992</s:year>
      <s:booktitle>Proceedings of the IWACA Workshop</s:booktitle>
    </s:proceedings>
    <s:inproceedings key="wea92" crossref="iwaca92">
      <s:author>
        <b:bibperson>
          <b:firstname>Alfred</b:firstname>
          <b:lastname>Weaver</b:lastname>
        </b:bibperson>
      </s:author>
      <s:title>The Xpress Transfer Protocol</s:title>
      <s:pages>253-259</s:pages>
    </s:inproceedings>
  </b:entries>
</b:bibliography>

```

Listing 6: Improved BibXML Document Using Cross References

```

<?xml version="1.0"?>
<b:bibliography xmlns:b="http://bibtexml.org/base"
  xmlns:s="http://bibtexml.org/standard">
  <b:macros>
    <b:macro name="iwaca92">Proceedings of the IWACA Workshop</b:macro>
    <b:macro name="munich">Munich, Germany</b:macro>
  </b:macros>
  <b:entries>
    <s:proceedings key="iwaca92">
      <s:title><b:macro ref="iwaca92"/></s:title>
      <s:address><b:macro ref="munich"/></s:address>
      <s:year>1992</s:year>
      <s:booktitle><b:macro ref="iwaca92"/></s:booktitle>
    </s:proceedings>
  </b:entries>
</b:bibliography>

```

Listing 7: BibXML Macros

Macros are defined using `b:macro` elements. The `name` attribute of the `b:macro` element specifies the name of the macro (`iwaca92` and `munich` in our example), by which it can be referenced, and the text content of the `b:macro`

element specifies the value of the macro (`Proceedings of the IWACA Workshop` and `Munich, Germany` in our example), by which a macro reference will be replaced. Macro definitions are grouped in a `b:macros` element, which is contained directly within the `b:bibliography` element. As with cross references, no nesting is allowed with macros. Macro references can be inserted inside field elements at any place where the replacement of the macro reference with the macro value yields a valid BibXML document. Macro references are declared using the `b:macro` element with a `ref` attribute. The `ref` attribute contains the name of the macro it is referencing. It is important to know that macro references in a BibXML document can reference macros that are defined in the same document as well as macros that have been defined in another document and have already been imported into the system. Therefore, it would be perfectly legal to first import the BibXML document shown in Listing 8 and then import the document shown in Listing 9.

```
<?xml version="1.0"?>
<b:bibliography xmlns:b="http://bibtexml.org/base"
  xmlns:s="http://bibtexml.org/standard">
  <b:macros>
    <b:macro name="iwaca92">Proceedings of the IWACA Workshop</b:macro>
    <b:macro name="munich">Munich, Germany</b:macro>
  </b:macros>
</b:bibliography>
```

Listing 8: BibXML Document Defining Two Macros

```
<?xml version="1.0"?>
<b:bibliography xmlns:b="http://bibtexml.org/base"
  xmlns:s="http://bibtexml.org/standard">
  <b:entries>
    <s:proceedings key="iwaca92">
      <s:title><b:macro ref="iwaca92"/></s:title>
      <s:address><b:macro ref="munich"/></s:address>
      <s:year>1992</s:year>
      <s:booktitle><b:macro ref="iwaca92"/></s:booktitle>
    </s:proceedings>
  </b:entries>
</b:bibliography>
```

Listing 9: BibXML Document Referencing Two Macros

The last feature of the BibXML format is the possibility to specify system information for each entry of the document. The system information of

an entry contains information about the modification of this entry, as well as information about access rights. This information is included inside a `b:system` element, which is an optional child of an entry element. Listing 10 shows the full system information of an entry using all four possible child elements of a `b:system` element: `b:modification`, `b:owner`, `b:group`, and `b:others`. If the `b:system` element is being used to describe an entry, at least one of these four elements needs to be present as its child.

```

<b:system>
  <b:modification>
    <b:user>smith</b:user>
    <b:datetime>2002-11-11T22:22:22</b:datetime>
  </b:modification>
  <b:owner rights="rw">smith</b:owner>
  <b:group rights="r">tik</b:group>
  <b:others rights="-"/>
</b:system>

```

Listing 10: System Information of a BibXML Entry

The following list describes all possible element descendants (as shown in Listing 10) of the `b:system` element:

- `b:modification`: This element contains information about the last modification of the entry. It needs a `b:user` and/or a `b:datetime` element child. Only the system administrator (see Section 3.3.4) is allowed to specify this information for an entry. If a normal user does make use of the `b:modification` element, it is ignored by the system and the user gets an appropriate notice.
 - `b:user`: The `b:user` child of the `b:modification` element contains the user name of the person who last modified the corresponding entry. If the entry is imported by a normal user (only administrators are allowed to specify information about the modification) or if the importing administrator does not specify the user who made the last modification, the user name of the importing person (the normal user or the administrator respectively) is used as the default value for the `b:user` element.
 - `b:datetime`: The `b:datetime` child of the `b:modification` element contains the date and time of the last modification of the corresponding entry. Its format is determined by the built-in simple type `xsd:dateTime` of XML Schema [6, 7]. If the entry is

imported by a normal user (only administrators are allowed to specify information about the modification) or if the importing administrator does not specify the date and time of the last modification, the date and time of the import event are used as the default value for the `b:datetime` element.

- **b:owner**: This element specifies the owner of the entry by containing the appropriate user name. Only the system administrator is allowed to set the owner of an entry. If a normal user specifies the owner, it is ignored by the system and the user is notified with an appropriate message. If an entry is imported by a normal user or if the importing administrator does not specify the owner of the entry, the user name of the importing person is used as the default value for the `b:owner` element. The optional `rights` attribute of the `b:owner` element gives the user the possibility to set access rights for the owner of the entry (also normal users can make use of this feature). There are four possible values for the attribute: `rw` (read and write rights), `r` (only read rights), `w` (only write rights) and `-` (neither read nor write rights). If this attribute is not set, the default value `rw` is used by the system.
- **b:group**: This element specifies a group that is granted special access rights for the entry compared to other users. As with the `b:owner` element, access rights are set by using the optional `rights` attribute. If the element is missing in the imported entry, no group is granted special rights. If the element specifies a certain group name but the `rights` attribute is missing, the default value `r` is used by the system. The `b:group` element can be used by normal users and administrators in the same way.
- **b:others**: The `b:others` element is an empty element with a mandatory `rights` attribute. It serves users and administrators to set access rights (`rw`, `r`, `w`, or `-`) for all users who are not the owner of the entry and not part of the group having access rights for this entry. So, if for example a user is part of the group having only read rights for a certain entry and others are granted read and write rights, this user does not have write but only read rights. If the `b:others` element is omitted, others have read rights only.

4.2 Validation of BibXML Documents

To import bibliographic data into the management system, users have to create a BibXML representation of their data (see Section 4.1) and submit it to the system. The first task of the management system is the validation of the imported document. The system has to check whether the imported document complies with the BibXML format or not. If the document is a valid BibXML document, the management system is able to handle the specified data and accepts the document for further processing. If the document turns out to be invalid, it is rejected. Essentially, the validation of a BibXML document is a two-step process as shown in Figure 3.

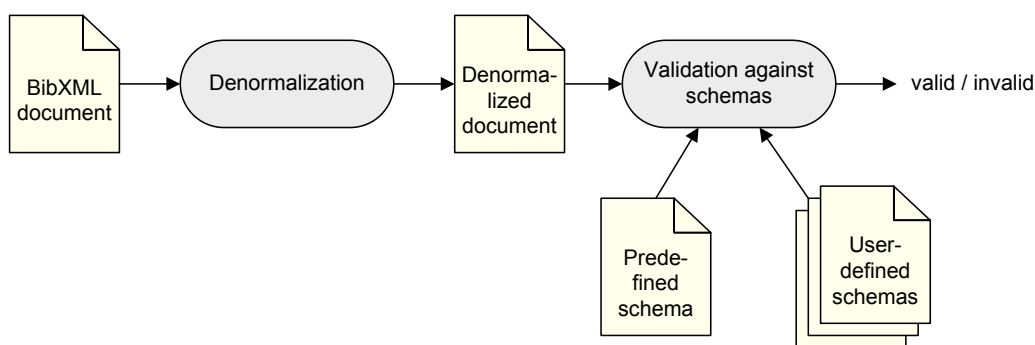


Figure 3: Two-step Validation Process

In a first step, the imported document is denormalized by resolving its macro and cross references. This denormalization also works with documents referencing macros or entries of the database. Then, the denormalized document is validated against a predefined schema and additional user-defined schemas. The predefined schema is a built-in schema of the system that defines the overall structure of the BibXML document. User-defined schemas are schema extensions defined by individual users: users have the possibility to extend the BibXML format accepted by the management system by defining their own entry and field elements in appropriate schema documents. These additional user-defined schemas are then passed to the system and used besides the predefined schema to validate BibXML documents (see Figure 2).

A core part of our diploma thesis was the schema design for the BibXML structure. The following Section will first compare different schema languages that were considered for the schema design. In Section 4.2.2, we introduce our own definition of a specialized schema language (*BibSchema*) and explain

its relation to other schema languages (*XML Schema* and *Schematron*). Furthermore, the process of importing a user-defined schema extension into the system is described. After having explained all aspects about the schema concept, the complete process of validating a BibXML document is discussed in Section 4.2.5.

4.2.1 Schema Languages

Several XML schema languages have been proposed to describe XML data structures and constraints. Consequently, the very first step of the schema design was to choose the schema language(s) best meeting the demands of our system. Figure 4 shows a classification⁹ of the six schema languages considered for our schema design.

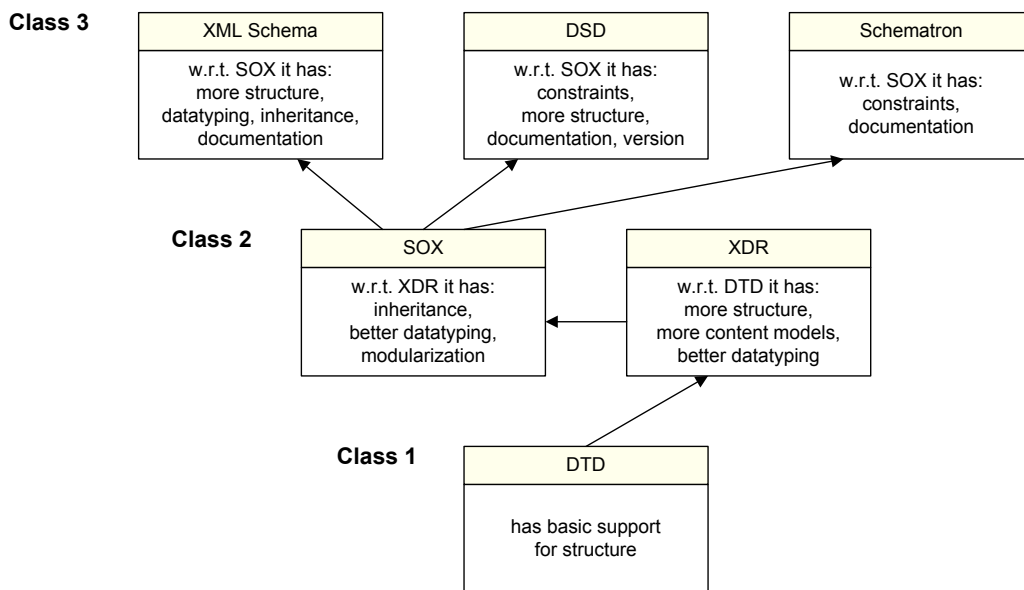


Figure 4: Classification of XML Schema Languages

With respect to their expressive power, the six languages can be categorized into the following three classes:

⁹This classification was introduced by Dongwon Lee and Wesley W. Chu, and is available at <http://cobase.cs.ucla.edu/tech-docs/dongwon/ucla-200008.html>.

- **Class 1:** DTD has the weakest expressive power. It severely lacks the support for datatypes. A schema for bibliographic data using DTD has been proposed by [4].
- **Class 2:** XDR and SOX have more expressive power than DTD. Nevertheless, their support for datatypes is not sufficient (e.g., there are no user-defined types).
- **Class 3:** XML Schema [6, 7], Schematron [2], and DSD have the strongest expressive power. Whereas XML Schema supports features for defining extensive datatypes and structure, Schematron provides a very flexible pattern language (using XPath expressions) that can describe sophisticated constraints. DSD tries to support common features supported by XML Schema (e.g., structures) and Schematron (e.g., constraints) along with some additional features.

The choice of the schema language(s) to be used has been based on three considerations: First of all, the schema language should be extensible in order to allow users to extend the schema and use these schema extensions to import any data they are interested in. Furthermore, the language should support a sophisticated mechanism for defining datatypes. Datatypes are especially needed when defining field types. The language should also provide the possibility to define so-called *co-constraints*. Co-constraints are dependencies between different parts of an XML document. These dependencies are a central part of the way constraints are defined for entries.

In order to support the definition of datatypes, the decision was made to use the schema language XML Schema. But XML Schema does not allow the specification of co-constraints, and therefore a second schema language had to be used for this purpose. We decided to make use of the Schematron schema language, which is extensible and has its strength in letting the user define sophisticated co-constraints, but has only poor support for datatypes. These characteristics make Schematron the ideal complement for XML Schema.

But the usage of two relatively complex schema languages results in a problem concerning the extensibility of the schema: It is unrealistic to ask users to define an XML Schema and a Schematron part for each schema extension, and it would also have been a very complex task to check and guarantee the consistency of these distinct parts. Consequently, we designed a specialized schema language called BibSchema, which allows users to define datatypes and co-constraints for their extensions in an easy way. BibSchema depends on XML Schema and Schematron in the following way (see Figure 5): In order to extend the schema, users write BibSchemas defining their

new entry and field types. The datatype part of this BibSchema document is then transformed into an appropriate XML Schema document, and the co-constraint part is transformed into a Schematron document. These additional XML Schema and Schematron schemas are then used by the system to validate bibliographic data. The transformation is implemented using XSLT stylesheets. Using this concept, Figure 3 can be refined as shown in Figure 6.

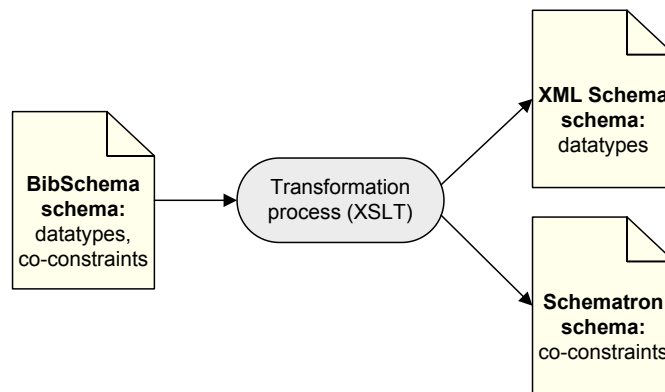


Figure 5: BibSchema, XML Schema, and Schematron

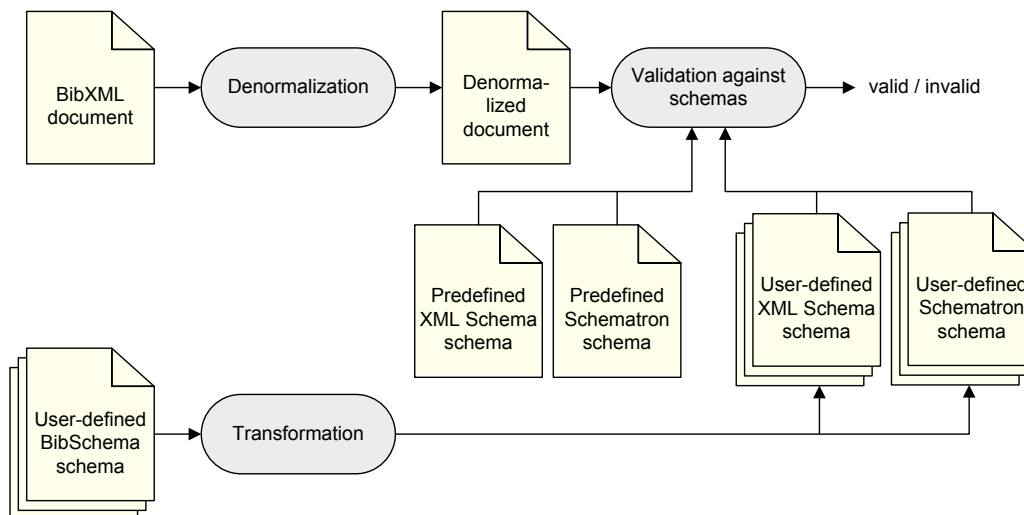


Figure 6: Redefined Validation Process

4.2.2 BibSchema

The schema language BibSchema has been introduced in the last section. In this section, we describe how BibSchema can be used to define schema extensions. BibSchema uses XML syntax and is specialized for defining entry and field types for bibliographies. Users extend the schema of the management system by defining their own entry and field elements in a BibSchema document. After passing the schema document to the system, they can use their new entry and field elements to describe bibliographic data.

The root element of the BibSchema document is a `schema` element in the `http://bibtextml.org/bibschema` namespace. By convention, this namespace is mapped to the prefix `bs`, but users are free to pick another prefix if they prefer. From this point forward, it is implied that the prefix `bs` is mapped to the `http://bibtextml.org/bibschema` namespace (Section 4.1 implied that the prefix `b` is mapped to the `http://bibtextml.org/base` namespace). Definitions of entry and field elements are included directly under the `bs:schema` element. This layout is shown in Listing 11.

```
<?xml version="1.0"?>
<bs:schema xmlns:bs="http://bibtextml.org/bibschema"
  defaultRefAndTargetNS="http://bibtextml.org/my_extension">
  <!-- entry and field element definitions go here -->
</bs:schema>
```

Listing 11: BibSchema Skeleton

As can be seen in Listing 11, the root element `bs:schema` has a mandatory `defaultRefAndTargetNS` attribute. This attribute serves two purposes: Its first purpose is to declare that this schema applies to documents in the `http://bibtextml.org/my_extension` namespace, which is referred to as the target namespace of the schema document. That means that the elements defined in this schema are in the `http://bibtextml.org/my_extension` namespace. The second purpose of the `defaultRefAndTargetNS` attribute will be explained later in this subchapter.

Defining Field Elements

The definition of a new field element is shown in Listing 12, where the field `month` is declared. A field element is defined by using the `bs:field` element. Its mandatory `name` attribute sets the name of the field element. In order

to define the type of the new element, one of the elements `restriction`, `list`, `union` (for the definition of simple types), or `sequence`, `choice`, `all` (for the definition of complex types) from the `http://www.w3.org/2001/XMLSchema` namespace can be used as a child element of the `bs:field` element. Just like in an XML Schema document, these six elements from the `http://www.w3.org/2001/XMLSchema` namespace can be used to describe simple and complex types. With respect to these six elements, the only difference between a BibSchema and an XML Schema document is, that in a BibSchema document, they are children of `bs:field` elements instead of `simpleType` or `complexType` elements, and that BibSchema does not allow the user to define named types. In Listing 12, the simple type `string` from the `http://www.w3.org/2001/XMLSchema` namespace has been restricted using facets to describe the type of the new field element `month`.

Furthermore, there is a special behavior of BibSchema that has to be taken into account when defining namespaces: Only the two namespaces `http://bibtextml.org/bibschema` and `http://www.w3.org/2001/XMLSchema` (to be declared as the default namespace) may be declared in the document and the declaration of these two namespaces has to be in the root element. The reason for this restriction in declaring namespaces is the inability of most XML Schema validators to cope with arbitrary declarations of namespaces.

```

<?xml version="1.0"?>
<bs:schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:bs="http://bibtextml.org/bibschema"
  defaultRefAndTargetNS="http://bibtextml.org/my_extension">
  <bs:field name="month">
    <restriction base="string">
      <enumeration value="January"/>
      <enumeration value="February"/>
      <enumeration value="March"/>
      ...
    </restriction>
  </bs:field>
</bs:schema>

```

Listing 12: Definition `month` Field Element

Using predefined types is a second way to define field elements: An empty `bs:field` element can be used with a `name` attribute specifying the name of the new field element, and a `isPerson` or `isSpecialText` attribute identifying the type of the new element. The default value of the `isPerson` and

`isSpecialText` attributes is `false`. This situation is shown in Listing 13.

```
<?xml version="1.0"?>
<bs:schema xmlns:bs="http://bibtexml.org/bibschema"
  defaultRefAndTargetNS="http://bibtexml.org/my_extension">
  <!-- definition of a person field element -->
  <bs:field name="author" isPerson="true"/>
  <!-- definition of a special text field element -->
  <bs:field name="booktitle" isSpecialText="true"/>
</bs:schema>
```

Listing 13: Definition of Person and Special Text Field Elements

The first definition assigns a predefined complex type for persons to an author field element. The `author` element can then be used in the BibXML document in two ways as shown in Listing 14.

```
<!-- using a person field element to describe a physical person -->
<author>
  <b:bibperson>
    <b:firstname>Michael</b:firstname>
    <b:middlename>Andrew</b:middlename>
    <b:lastname>Smith</b:lastname>
    <b:suffix>Jr.</b:suffix>
  </b:bibperson>
</author>
<!-- using a person field element to describe a corporation -->
<author>
  <b:name>Macromedia, Inc.</b:name>
</author>
```

Listing 14: Two Ways of Using a Person Field Element

Using the `b:bibperson` element, persons are described with their first, middle, and last name as well as their suffix. At least a first or a last name is required, and a middle name is only allowed if a first name is present. An optional suffix can be added to the last name of a person. A `b:name` child is used to identify the corporation or institution where a resource was created, rather than the individual persons who participated in the creation.

The second definition of Listing 13 declares that a `booktitle` field element may contain simple text mixed with `b:special` elements (what is referred to as *special text*) as shown in Listing 15. The usage of special text


```
<booktitle><b:special><b:text>LaTeX2e</b:text><tex>\LaTeXe{</tex>
</b:special>: A Document Preparation System</booktitle>
```

Listing 15: Special Text Field

inside certain field elements has the following reason: Suppose a reference to a book with the title *L^AT_EX 2_ε: A Document Preparation System*. Many people write their papers in L^AT_EX and want to have the possibility to export their references to the BIB_TE_X format. So, they want the mentioned title to be output as `\LaTeXe{}`: A Document Preparation System using the appropriate L^AT_EX control sequence `\LaTeXe{}`. On the other side, users want to be able to find their references using the search routines of the management system. But the problem is that the system does not understand L^AT_EX control sequences like the one used in the mentioned title. So, should users enter their field values with or without L^AT_EX control sequences? The solution to this problem is the `b:special` element. The subtree of the `b:special` element shown in Listing 15 guarantees full L^AT_EX compatibility (when exporting the reference into the BIB_TE_X format, an appropriate stylesheet¹⁰ replaces the `b:special` element with the value of its `tex` child, `\LaTeXe{}`), as well as a full compatibility with the search routines of the system (when searching and displaying the entry, the system replaces the `b:special` element with the value of its `b:text` child, `LaTeX2e`). The `b:text` element is a mandatory child of the `b:special` element. But the `tex` child is a user-defined element. We refer to such user-defined elements that can be used as children of a `b:special` element as *special items*. The definition of a special item like `tex` is very similar to the definition of field elements as can be seen in the BibSchema document of Listing 16. Element `bs:specialItem` is used for the definition of the new special item. Its mandatory `name` attribute sets the name of the special item element. In order to define the type of the new element, one of the elements `restriction`, `list`, `union` (for the definition of simple types), and `sequence`, `choice`, `all` (for the definition of complex types) from the <http://www.w3.org/2001/XMLSchema> namespace can be used as a child element of the `bs:specialItem` element.

Furthermore, `bs:field` and `bs:specialItem` elements have an optional `mixed` attribute. If its value is set to `true` (the default value is `false`), the field or special item element gets a mixed content model. Obviously, this feature can only be used if a `sequence`, `choice`, or `all` element is used as

¹⁰The export of bibliographic data using stylesheets is described in Section 4.7.

```

<?xml version="1.0"?>
<bs:schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:bs="http://bibtexml.org/bibschema"
  defaultRefAndTargetNS="http://bibtexml.org/my_extension">
  <bs:specialItem name="tex">
    <restriction base="string"/>
  </bs:specialItem>
</bs:schema>

```

Listing 16: Definition of Special Item `tex`

a child of the `bs:field` or `bs:specialItem` element. A `bs:field` element (but not a `bs:specialItem` element) is also allowed to have a `repeatable` attribute as shown in Listing 17. By default, each field element is only allowed to occur once inside the same entry element. But if a field element is defined with the `repeatable` attribute set to `true` (the default value is `false`), then there is no restriction on how often the field element may occur inside the same entry element.

```

<?xml version="1.0"?>
<bs:schema xmlns:bs="http://bibtexml.org/bibschema"
  defaultRefAndTargetNS="http://bibtexml.org/my_extension">
  <bs:field name="author" isPerson="true" repeatable="true"/>
</bs:schema>

```

Listing 17: Definition of Repeatable `author` Field

Defining Entry Elements

A new entry element is defined using a `bs:entry` element with a mandatory `name` attribute specifying its name. By default, an entry element may contain any field elements for which an appropriate BibSchema definition exists. But usually, entry types should be more restrictive with regard to their field information. For example, it would probably be reasonable for a book entry to require a title, a publisher, and a year field. Furthermore, a book entry should have either an author field or an editor field, but not both. In BibSchema, there is a special mechanism for defining such dependencies, which is introduced in Listing 18.

```

<?xml version="1.0"?>
<bs:schema xmlns:bs="http://bibtexml.org/bibschema"
  defaultRefAndTargetNS="http://bibtexml.org/my_extension">
  <bs:entry name="book">
    <bs:field ref="title"/>
    <bs:field ref="publisher"/>
    <bs:field ref="year"/>
    <bs:one>
      <bs:field ref="author"/>
      <bs:field ref="editor"/>
    </bs:one>
    <bs:maxOne>
      <bs:field ref="volume"/>
      <bs:field ref="number"/>
    </bs:maxOne>
  </bs:entry>
</bs:schema>

```

Listing 18: Definition of `book` Entry Element

The BibSchema document of Listing 18 has to be read like that: The `book` entry element has to contain a `title`, a `publisher` and a `year` field element. In addition to these elements, it must also contain exactly one of the field elements `author` or `editor`, and at most one of the field elements `volume` or `number`. In order to describe the content model of an entry element, field elements are referenced using `bs:field` elements with a mandatory `ref` attribute specifying the name of the referenced field. If the field names used in `ref` attributes are not prefixed, then they are by default in the namespace specified by the `defaultRefAndTargetNS` attribute of the root element. This is the second purpose of the `defaultRefAndTargetNS` attribute besides declaring the target namespace of the BibSchema. In case of Listing 18, all field elements referenced are in the `http://bibtexml.org/my_extension` namespace. If we change the situation and suppose that the field element `title` is not defined in the `http://bibtexml.org/my_extension`, but in the `http://bibtexml.org/another_extension` namespace, then the BibSchema of Listing 18 has to be rewritten as seen in Listing 19.

First, an `uri` and a `prefix` attribute of a `bs:ns` element have to be used to link a prefix (`ae`) to the appropriate namespace (`http://bibtexml.org/another_extension`). Then, the declared prefix can be used inside the `ref` attribute of a `bs:field` element in order to reference field elements from another namespace than the one set in the `defaultRefAndTargetNS` attribute.

```

<?xml version="1.0"?>
<bs:schema xmlns:bs="http://bibtexml.org/bibschema"
  defaultRefAndTargetNS="http://bibtexml.org/my_extension">
  <bs:ns uri="http://bibtexml.org/another_extension" prefix="ae"/>
  <bs:entry name="book">
    <bs:field ref="ae:title"/>
    <bs:field ref="publisher"/>
    <bs:field ref="year"/>
    <bs:one>
      <bs:field ref="author"/>
      <bs:field ref="editor"/>
    </bs:one>
    <bs:maxOne>
      <bs:field ref="volume"/>
      <bs:field ref="number"/>
    </bs:maxOne>
  </bs:entry>
</bs:schema>

```

Listing 19: Definition of a `book` Entry Element by Referencing a Field Element (`ae:title`) from Another Namespace

There are three different grouping mechanisms for defining dependencies between field elements of an entry:


- `bs:one`: Exactly one of the field elements referenced by the `bs:field` children must be present in the entry element.
- `bs:maxOne`: At most one of the field elements referenced by the `bs:field` children may be present in the entry element.
- `bs:minOne`: At least one of the field elements referenced by the `bs:field` children must be present in the entry element.

4.2.3 Dependencies between BibSchema, XML Schema, and Schematron Schemas

As outlined in Section 4.2.1, the BibSchema schema language depends on XML Schema and Schematron in the following way: The datatype part of the BibSchema document is transformed into an appropriate XML Schema document, and the co-constraint part is transformed into a Schematron document (see Figure 5). These XML Schema and Schematron documents are then used by the system to validate bibliographic data. By using this concept, we do not have to write a specialized BibSchema validator, but we

can make use of existing XML Schema and Schematron validators. The *skeleton1-5.xsl*¹¹ implementation of Schematron 1.5 was used in conjunction with the XSLT processor *Saxon*¹² to validate data against Schematron documents. Unfortunately, there does not seem to exist a single XML Schema processor that fully implements the specification. Thus, we spent a lot of time testing different processors (mainly *XSV*¹³, *xsdvalid*¹⁴, and *Xerces*¹⁵). We decided to use the *Xerces2 Java Parser 2.3.0 Release*, because it seems to implement the biggest subset of XML Schema.

In the rest of this section and in the following two sections (4.2.4 and 4.2.5), we introduce details about the implementation of the validation process. These sections are intended for software developers who want to understand the implementation in detail. In order to allow developers to easily find and study the corresponding code fragments, we explicitly name the participating files. By convention, all relative file paths mentioned in this report are relative to the installation directory of the management system, which can be declared in *include.php*. Usually, this directory is the document root directory of the Apache installation (*htdocs*) or a subdirectory of it.

 The process of transforming a BibSchema document into an XML Schema and a Schematron document includes the two stylesheets *schema/xmlSchemaTransform.xsl* and *schema/schematronTransform.xsl*. The stylesheet *schema/xmlSchemaTransform.xsl* transforms the x-th user-defined BibSchema document into the XML Schema document *schema/x/x.xsd* (the first imported schema is transformed into *schema/1/1.xsd*, the second into *schema/2/2.xsd* and so on). The stylesheet *schema/schematronTransform.xsl* is used to transform the x-th user-defined BibSchema document into the Schematron document *schema/x/x.sch*. Furthermore, the predefined XML Schema document for validating bibliographic data is the *schema/xmlSchemaBase.xsd* document and the predefined Schematron document is *schema/schematronBase.sch*. This situation is illustrated in Figure 7.

A central key for understanding the schema design is the fact that the XML Schema documents (*schema/xmlSchemaBase.xsd*, *schema/1/1.xsd*, *schema/2/2.xsd*, ...) and the Schematron documents (*schema/schematronBase.xsd*, *schema/1/1.sch*, *schema/2/2.sch*, ...) are completely independent of each other. First, the XML Schema documents are used for validating bibliographic data, and only if the data turns out to be valid with respect to these

¹¹Available online at <http://www.ascc.net/xml/schematron/1.5/skeleton1-5.xsl>

¹²Available online at <http://saxon.sourceforge.net/>

¹³Available online at <http://www.ltg.ed.ac.uk/~ht/xsv-status.html>.

¹⁴Available online at <http://www.xmlmind.com/xsdvalid.html>

¹⁵Available online at <http://xml.apache.org/xerces2-j/index.html>

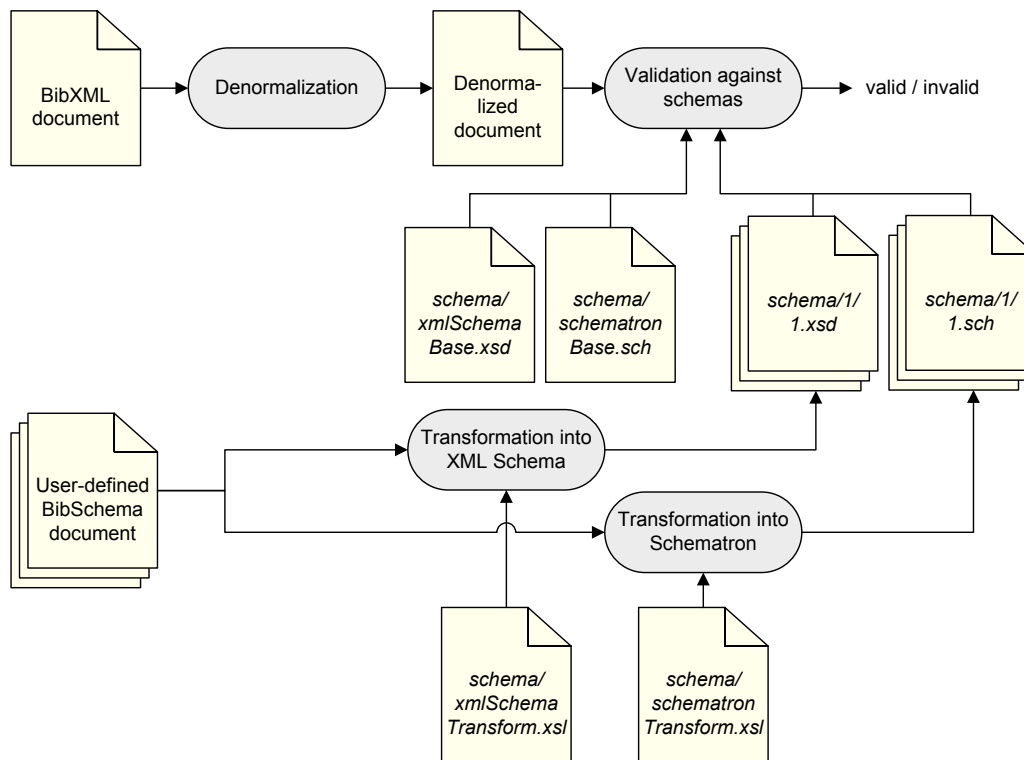


Figure 7: Validation Process Overview Indicating File Paths

documents, the data is validated against the Schematron documents as well.

Dependencies between BibSchema and XML Schema Schemas

Figure 8 shows the dependencies between BibSchema and XML Schema schemas. The *schema/xmlSchemaBase.xsd* schema defines the overall structure of the BibXML document, and the *schema/x/x.xsd* schemas (from this point forward, it should be understood that *schema/x/x.xsd* stands for the whole group *schema/1/1.xsd*, *schema/2/2.xsd*, ...) specify the individual user-defined entry, field, and special item elements that can be used in BibXML documents.

The connection between the *schema/xmlSchemaBase.xsd* and the *schema/x/x.xsd* documents is implemented by declaring an appropriate substitution group hierarchy. The substitution group mechanism of XML Schema is a flexible way to designate element declarations as substitutes for other ele-

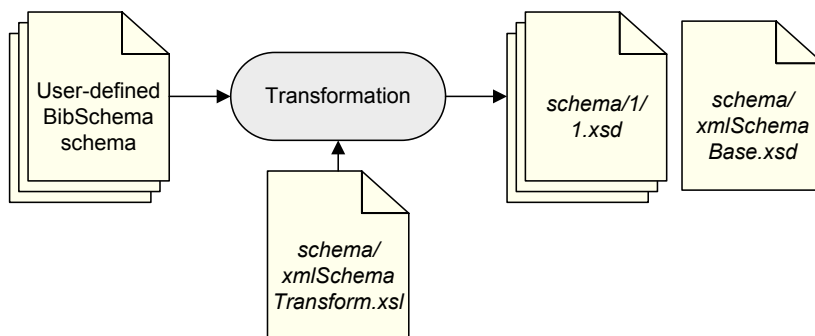


Figure 8: Dependencies between BibSchema and XML Schema Schemas

ment declarations in content models. Each substitution group consists of a head and one or more members. Whenever the head element declaration is referenced in a content model, one of the member elements may be substituted in place of the head. In order to allow the schema to be extended with user-defined entry, field, and special item elements, we make use of the substitution group mechanism in the following way: In the predefined *schema/xmlSchemaBase.xsd* document, we declare the abstract substitution group heads `b:entry`, `b:field`, and `b:specialItem`. Further, the declarations of entry, field, and special item elements in user-defined BibSchema documents are transformed into declarations of according substitution group members in the *schema/x/x.xsd* documents.

There are three substitution groups defined in our system. The first substitution group (see Figure 9) has the declaration of the abstract element `b:entry` as its head, and the declarations of user-defined entry elements like `me:book`, `me:article`, and `me:misc` (in the `http://bibtexml.org/my_extension` namespace) as its members. This means that at any place where `b:entry` appears in a content model, any of the `me:book`, `me:article`, or `me:misc` elements may appear in the instance.

The second substitution group (see Figure 10) has the declaration of the abstract element `b:field` as its head, and the declarations of user-defined field elements like `me:title`, `me:author`, and `me:year` as its members. This means that at any place where `b:field` appears in a content model, any of the `me:title`, `me:author`, or `me:year` elements may appear in the instance. The same characteristics apply to the third substitution group for special item elements, which is illustrated in Figure 11.

Listing 12 shows the relevant code sections of *schema/xmlSchemaBase.xsd*,

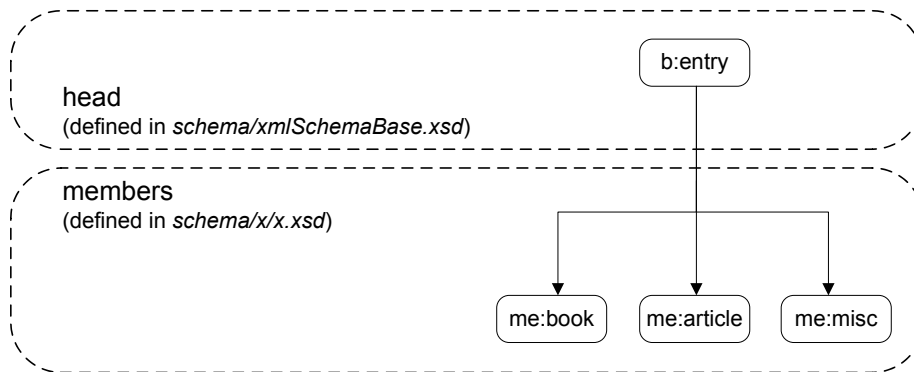


Figure 9: Substitution Group 1: Entry Elements

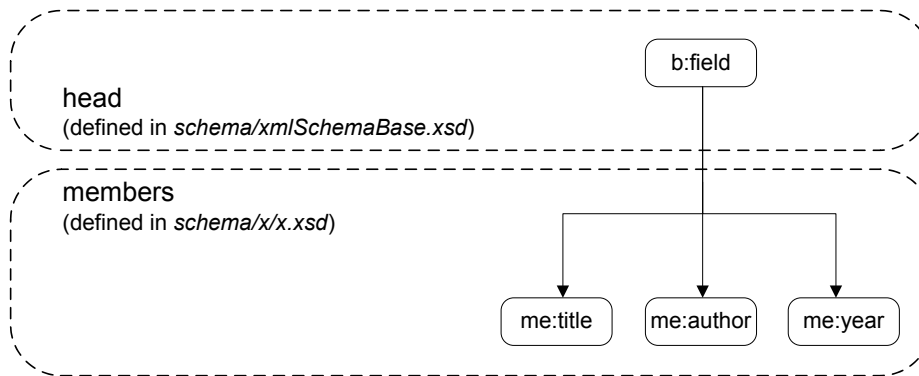


Figure 10: Substitution Group 2: Field Elements

where the abstract head elements `b:entry`, `b:field`, and `b:specialItem` are declared and referenced. The transformed BibSchema declarations for entry, field, and special item elements are shown in Listing 13, where three member elements `me:book`, `me:title`, and `me:tex` are declared (one member for each substitution group).

Dependencies between BibSchema and Schematron Schemas

Figure 14 shows the dependencies between BibSchema and Schematron schemas. The `schema/schematronBase.sch` document defines constraints that are common to all entry, field, and special item elements. The `schema/x/x.sch` doc-

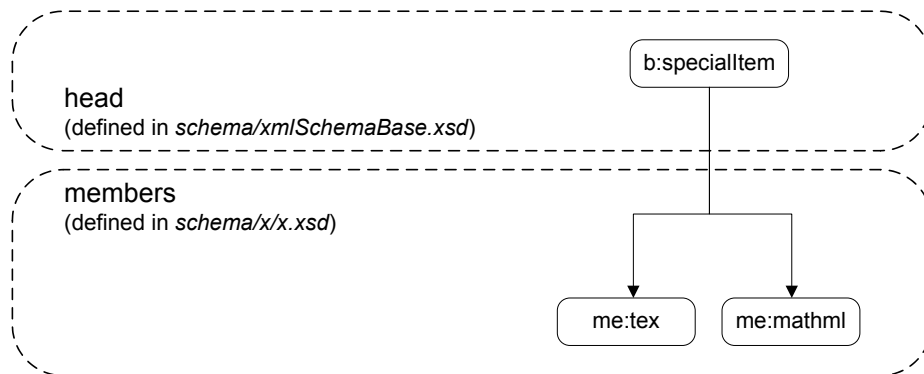


Figure 11: Substitution Group 3: Special Item Elements

uments contain specific constraints for the individual user-defined entry and field elements. As opposed to the XML Schema documents, the predefined and the user-defined Schematron documents are completely independent of each other (therefore they do not have to be connected with import statements or alike). That is because a Schematron schema allows everything that is not explicitly forbidden. In contrast, XML Schema disallows everything that is not explicitly allowed.

The transformation of a BibSchema schema into a Schematron document is demonstrated using the simple BibSchema example shown in Listing 20. It defines an entry element `proceedings` in the `http://bibtexml.org/my_extension` namespace, which has to contain the mandatory fields `title` and `year` and at most one `volume` or `number` field. These constraints are transformed into appropriate Schematron assertions shown in Listing 21.

4.2.4 Validating and Importing BibSchema Schemas

The complete process of validating and eventually importing a BibSchema schema is implemented by the PHP function `check_bib_schema()` in the `schema/validation.php` file. At first, the user has to upload the BibSchema schema using the *Schema Management* interface. The uploaded document is then checked for well-formedness by Apache's Xerces processor. In the next step, the target namespace specified by the `defaultRefAndTargetNS` attribute of the root element of the schema is read and the system checks that this namespace is not already taken by another schema extension by consulting the

```

<?xml version="1.0" encoding="UTF-8"?>
<bs:schema xmlns:bs="http://bibtexml.org/bibschema"
  defaultRefAndTargetNS="http://bibtexml.org/my_extension">
  <bs:entry name="proceedings">
    <bs:field ref="title"/>
    <bs:field ref="year"/>
    <bs:maxOne>
      <bs:field ref="volume"/>
      <bs:field ref="number"/>
    </bs:maxOne>
  </bs:entry>
</bs:schema>

```

Listing 20: BibSchema Document To Be Transformed

```

<?xml version="1.0" encoding="UTF-8"?>
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">
  <sch:ns uri="http://bibtexml.org/my_extension" prefix="me"/>
  <sch:pattern name="mandatory fields">
    <sch:rule context="me:proceedings">
      <sch:assert test="me:title">Field 'me:title' is missing.
    </sch:assert>
      <sch:assert test="me:year">Field 'me:year' is missing.
    </sch:assert>
      <sch:assert test="count(me:volume[1] | me:number[1])&lt;=1">
At most one of the following fields may be present: 'me:volume',
'me:number' .</sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>

```

Listing 21: Transformed Schematron Schema

database. If the specified namespace is still available, the document is validated against the Schematron document *schema/schematron1ForBibSchema.sch*. This simple Schematron document ensures that the namespace declarations in the imported document are correct (using the special conventions about declaring namespaces in BibSchema documents from Section 4.2.2). Next, the detailed structure of the BibSchema document is validated using the XML Schema and Schematron schemas *schema/xmlSchemaForBibSchema.xsd* and *schema/schematron2ForBibSchema.sch*. If the BibSchema document is valid with respect to these two schemas, external field references are verified (the referenced external fields need to exist), and the schema is transformed into two temporary XML Schema and Schematron schemas using the stylesheets *schema/xmlSchemaTransform.xsl* and *schema/schematronTransform.xsl*. The

system then makes sure that the two temporary schemas conform to the corresponding schema specifications. If they do, the system checks again that the target namespace of the imported schema document has not been taken in the meantime. If this check succeeds, the BibSchema document is definitively accepted, the database is updated accordingly, and the two temporary XML Schema and Schematron schemas are copied to the destinations *schema/x/x.xsd* and *schema/x/x.sch*.

4.2.5 Process of Validating BibXML Documents

The process of validating a BibXML document is implemented by the PHP function *validate_instance()* in the *schema/validation.php* file. At first, the user has to upload the BibXML document using the *Data Import* interface. The uploaded document is then checked for well-formedness by Apache's Xerces processor. Next, the document is validated against the Schematron schema *schema/prevalidation1.sch* in order to make sure that certain basic constraints are satisfied. Macro definitions of the BibXML document are then inspected, and the system checks that they are not already defined by other users, and that their name follows the BIBTEX rules. In the next step, cross referenced external entries are retrieved from the database and merged with the uploaded BibXML document. Then, this merged document is validated against the Schematron schema *schema/prevalidation2.sch*, which makes sure that cross references are used correctly (e.g., nesting of cross references is forbidden). Then, cross references are resolved by applying the stylesheet *schema/resolveCrossrefs.xsl* to the BibXML document. In this step, entries inherit any fields they are missing from the entries they cross reference. Next, referenced external macros are retrieved from the database and inserted into the document. Then, macro references (all macro references are internal at this point) are resolved by applying the stylesheet *schema/resolveMacros.xsl* to the document. Now, the BibXML document is completely denormalized, and it can finally be validated against the predefined and user-defined schema documents of the system (*schema/xmlSchemaBase.xsd*, *schema/schematronBase.sch*, *schema/x/x.xsd*, *schema/x/x.sch*).

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://bibtexml.org/base" ... >
  <xs:element name="bibliography">
    <xs:complexType>
      <xs:all>
        <xs:element name="entries" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="entry" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        ...
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="entry" type="Entry" abstract="true"/>
  <xs:complexType name="Entry">
    <xs:sequence>
      <xs:element ref="field" maxOccurs="unbounded"/>
      ...
    </xs:sequence>
  </xs:complexType>
  <xs:element name="field" type="xs:anyType" abstract="true"/>
  <xs:complexType name="Special">
    <xs:choice minOccurs="2" maxOccurs="unbounded">
      <xs:element ref="specialItem"/>
      ...
    </xs:choice>
  </xs:complexType>
  <xs:element name="specialItem" type="xs:anyType" abstract="true"/>
  ...
</xs:schema>

```

Figure 12: Head Element Declarations in *xmlSchemaBase.xsd*

```

<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:b="http://bibtexml.org/base"
  targetNamespace="http://bibtexml.org/my_extension" ... >
<import namespace="http://bibtexml.org/base"
  schemaLocation="../xmlSchemaBase.xsd"/>
<element name="book" type="b:Entry" substitutionGroup="b:entry"/>
<element name="title" substitutionGroup=" b:field ">
  <simpleType>
    <!-- descendants of BibSchema element bs:field with
      attribute name="title" go here -->
  </simpleType>
</element>
<element name="tex" substitutionGroup="b:specialItem">
  <simpleType>
    <!-- descendants of BibSchema element bs:specialItem with
      attribute name="tex" go here -->
  </simpleType>
</element>
</schema>

```

Figure 13: Member Element Declarations

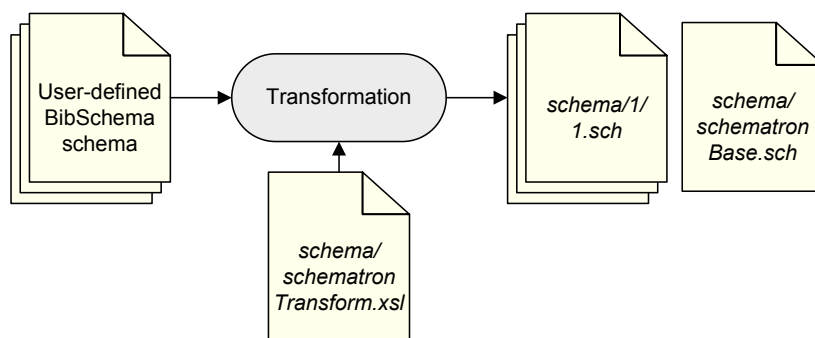


Figure 14: Dependencies between BibSchema and Schematron Schemas

4.3 Database Structure

For storing and managing the bibliographic data, the system uses the MySQL database. The structure of this database is designed in a way that the BibXML format with its extensibility and all other features is optimally supported. We are using tables of the type InnoDB¹⁶. These special tables are available in the “max” distribution of MySQL and support foreign key constraints with referential integrity, row level locking, and transactions. These features are very helpful to ensure the integrity and consistency of the database, and are unfortunately not yet implemented in the standard MySQL tables. An overview of all tables, their attributes, and the relations between them is shown in Figure 15. We are using *italics* when referring to table or attribute names from the figure. Required attributes (which can not be NULL) are printed in **bold**. *PK* stands for primary key, and *FK* for foreign key. A detailed description of each table follows below.

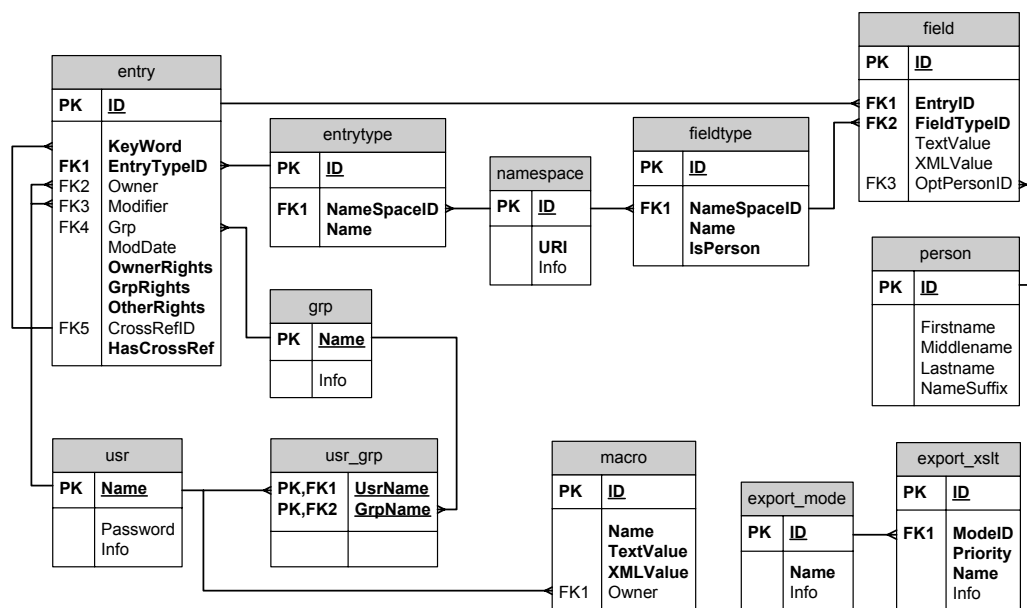


Figure 15: Conceptual Database Schema

¹⁶More information about InnoDB: <http://www.innodb.com/>

Table *entry*

This table contains all information associated with BibXML entries. The *ID* attribute uniquely identifies each entry, and is used as a foreign key in the table *field*, expressing that each entry has one or more fields. The attribute *KeyWord* corresponds to the key used in the BibTeX/BibXML entry. The mandatory attribute *EntryTypeID* is a foreign key from the table *entrytype*, and specifies the type of the entry.

Owner and *Modifier* are foreign keys from the table *usr* and specify the owner and modifier of the entry. *Grp* is a foreign key from the table *grp* and specifies which user groups have access to the entry. *ModDate* contains the modification date of the entry.

OwnerRights, *GrpRights*, and *OthersRights* specify the access rights for the entry in the same way as on Unix systems. Possible values are “r” for read only, “w” for write only, “rw” for read and write and “-” for no rights at all.

CrossRefID is NULL by default, but is a foreign key when referencing another entry. *HasCrossRef* is a binary attribute that is set to “1” if the entry is referencing another entry and to “0” otherwise. *HasCrossRef* can be used to detect orphaned entries because *CrossRefID* is set to NULL but *HasCrossRef* keeps its value “1” when the referenced entry is deleted.

Table *entrytype*

This table is used to manage all available entry types in the system. It contains the primary key attribute *ID*, the foreign key *NamespaceID* from table *namespace* that identifies a unique namespace, and the local name (*Name*) of the entry type.

Table *namespace*

This table is used to manage all available namespaces. It contains the primary key attribute *ID*, a unique URI (*URI*), and optional additional information (*Info*) for this namespace.

Table *field*

This table contains all information associated with a BibXML field. The *ID* attribute uniquely identifies each field. The attribute *EntryID* is a foreign

key from the table *entry* and specifies to which entry the field belongs. The foreign key *FieldTypeID* from the table *fieldtype* specifies the type of the field.

XMLValue contains the content of the field. *TextValue* is the text value generated by applying the stylesheet *import/special.xsl* to the content of *XMLValue*.

The optional foreign key *OptPersonID* from the table *person* is used if the *fieldtype* is defined to be a person (*IsPerson* set to "1"), and if it contains content of the type *bibperson* (please see Section 4.2.2 for more information). If the field contains information of the type *person name*, then the attribute *TextValue* contains the information.

Table *fieldtype*

This table is used to manage all available fieldtypes. It contains the primary key attribute *ID*, the foreign key *NameSpaceID* from table *namespace* that identifies a unique namespace, and the local name (*Name*) of the fieldtype. The binary attribute *IsPerson* is set to "1" if the field type is of type *person*, and to "0" otherwise.

Table *person*

This table contains all information associated with a person. The *ID* attribute identifies each person uniquely. *Firstname*, *Middlename*, *Lastname*, and *Namesuffix* contain the respective information from the *bibperson* syntax described in Section 4.2.2.

Table *usr*

This table is used to manage the users of the system. Primary key *Name* is the actual user name. The user password is stored encrypted by a one-way hash function in the attribute *Password*. Additional information can be added in the *Info* attribute.

Table *grp*

The *grp* table is very simple with the group name as primary key *Name*, and attribute *Info* for additional information.

Table *usr_grp*

Table *usr_grp* consists of the two primary keys from the tables *usr* and *grp* as foreign keys, and associates users and groups in a many-to-many relationship.

Table *macro*

This table is used to manage all available macros. The primary key *ID* identifies each macro uniquely. The macro can be referenced in fields by its name (*Name*). The macro value is stored in *XMLValue* and *TextValue* in the same way as described for table *field* above. Access rights are simplified in such a way that only the owner (*Owner* as foreign key from table *usr*) can manage a macro.

Table *export_mode*

This table contains all available export modes with the primary key *ID*, the name (*Name*) of the export mode, and additional information (*Info*). The *ID* is used as a foreign key in the table *export_xslt*, expressing that each export mode has one or more XSLT stylesheets.

Table *export_xslt*

This table contains information about all available XSLT stylesheets in the system. The primary key *ID* identifies each stylesheet uniquely. The attribute *ModeID* is a foreign key from the table *export_mode* and specifies to which export mode the stylesheet belongs. *Priority* contains the per mode (*modeID*) priority of each stylesheet, a lower number means higher import precedence. The system keeps the priorities consistent using only low numbers for each mode (1,...,[number of stylesheets with this *ModeID*]). Attribute *Name* specifies the name for each stylesheet and additional information can be added in the *Info* attribute.

A comment on the macro feature

As macros can appear anywhere in the value of a field, the macro feature can not be supported by using foreign keys and referential integrity. And because macros represent a form of normalization, it makes no sense to resolve all the macros before inserting them into the database. Therefore, if the value of a BibXML field like

```
<title xmlns="http://bibtexml.org/standard">
  A guide to <macro xmlns="http://bibtexml.org/base" ref="db"/> design
</title>
```

contains references to macros, the *XMLValue* in the database table *field*, in our case

```
A guide to <macro xmlns="http://bibtexml.org/base" ref="db"/> design
```

still contains the macro reference. This leads to some problems with the search routine, which will be discussed in Section 4.5.

4.4 Data Import

The data import into the system is managed through the menu item *Data Import* in the Web interface, and executed by *import.php*. The specific steps of the import process are shown in Figure 16 and are explained below.

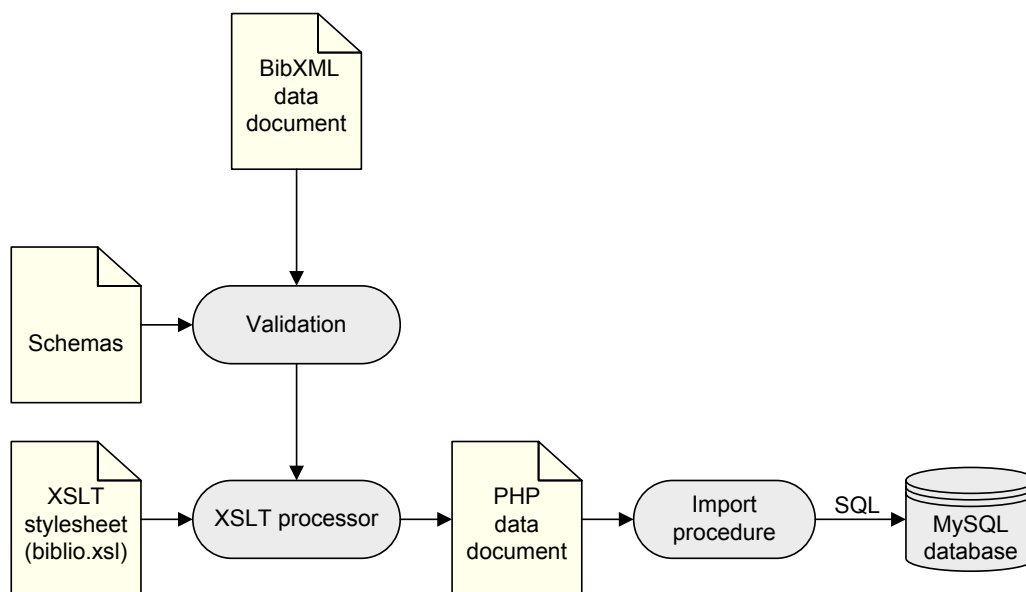


Figure 16: Data Import

As a first step, the user has to specify the document he would like to import by indicating its path and choosing from some import options in the HTML form. When the user submits the form, the import document

is uploaded to the server together with the selected options. If the upload is successful, the import document is validated with the function *validate_instance()* to ensure that the document is proper BibXML as defined by the existing schemas. See Section 4.2 for more information about the validation process.

If the validation is successful, the stylesheet *import/biblio.xsl* is applied to the valid BibXML document through the function *xslt_process()*. The result of this transformation is a PHP file containing all import data and instructions on how to import the data into the MySQL database in PHP syntax. This PHP file is processed by the main import file (*import.php*) resulting in an interactive import process that will ask the user for action if necessary, give status information to the user, and insert all the import data into the database.

4.5 Search

The bibliographic data in the system can be searched through the menu item *Search* in the Web interface. The system provides a simple as well as an advanced search interface, which are both described below. The general process of searching is in both cases the same. The user enters the search criteria in an HTML form, and submits it to the server. The search engine, implemented in the file *fast_search.php*, transforms this search request into a MySQL query, queries the database, and presents the results in the Web interface using the function *display_results()*.

In Section 4.3 we describe how macros are stored in the value of a field. This causes serious problems for the search routine. How can the macro value be searched for something, if only the macro reference is stored in a field?

In a first attempt to solve this problem, we tried creating a temporary table with all entry and field information relevant for the corresponding search query. In this temporary table the macro references were resolved, the fields were updated with the macro values, and the table could be queried as if no macros existed at all. Unfortunately, this solution led to serious performance problems. Some search queries that were not very sophisticated needed more than two minutes for a database with approximately 1600 entries. The reason is that MySQL is very fast at reading or searching tables, but very slow at writing or updating tables. For these reasons, we decided to choose another approach.

Instead of trying to actually resolve the macros, which leads to slow write operations, the system now uses a different technique with no write operations

at all. In a first step, the system takes the search keyword specified by the user and searches in the attribute *XMLValue* of the table *macro* for this keyword. That way, all relevant macros containing the keyword are found, and their names are cached. In a second step, the system now searches the actual entries with their fields not only for the keyword, but also for references to the relevant macros found before. The references to macros are found using the quite efficient search function for regular expressions (*RLIKE*) of MySQL. With this technique, even not very sophisticated searches are quite fast, and most notably much faster than with the approach described above. A disadvantage of this approach is that search queries over macro boundaries are not possible. Of course it is still very advisable to narrow any search as much as possible by using the advanced search interface.

Simple Search

This simple search interface consists of a single text field where the search keyword can be entered. The keyword does not have to be a simple word, but can be a regular expression that allows a very sophisticated search. The exact syntax for these search expressions is explained in the "User Guide" in Appendix A. The system will search for the search expression in the attributes *entry.KeyWord*, *field.XMLValue*, *field.TextValue*, and in all attributes of the table *person*.

Advanced Search

The advanced search interface features multiple input fields for selective search for distinct entry and field information, and supports the same search for regular expressions as the simple search. For user convenience, many input fields are implemented as drop-down menus that are dynamically created from the database, and thus do only contain the options that actually exist in the database. For example, if there are only entries belonging to the users "Smith" and "Anderson" in the system, the drop-down menu for *Owner* will only contain these two options and the wildcard "—ANY—". The advanced interface offers a very sophisticated way to search the database and in most cases should be preferred over the simple interface, because the search can be constrained better leading to faster search results.

Search Results

Search results are presented in a table with all found entries listed, and can be expanded to show all fields as well. In this case, the field data is dynamically collected from the database, and the result page is reloaded. The results can also be sorted by various criteria, exported into all available export formats, or added to the selection, which is explained in the next section.

4.6 Selection

The selection works like the shopping cart system used by many E-shops and is available through the menu item *Selection* in the Web interface. Search results (entries) from multiple searches can be added to the selection, and in that way collected for later reuse or processing. In the selection, the collected entries are presented in the same expandable table used for the direct search results. They can be selected for export, deleted from the database (if the user has sufficient rights), removed from the selection, and reordered with a few mouse clicks. The system keeps track of the current content of the selection, the content's order, and expansion status using the PHP session variables that will only be deleted after logout.

4.7 Export Management

Besides the export into the extensible BibXML format, the system also supports the export into user defined formats. These user defined export modes are realized by uploading appropriate XSLT stylesheets that can be applied to the BibXML file, generating the desired export format. The *Export Management* is used to create and manage these export modes and their stylesheets. As soon as an export mode is created and at least one stylesheet added to it, the export mode is available as an export option for search results and the entries in the selection. The export mode for BIB_TE_X is already completely implemented as an example for a user defined mode.

To create a new export mode, the user must first specify an appropriate name and should provide some information that characterizes the mode. The mode will then be created and stored in the database. Afterwards, the user must upload at least one stylesheet with a suitable name to activate the export mode. This stylesheet is stored in the file system and associated with the corresponding mode through the database as explained in Section 4.3. If there are several stylesheets in the system for one mode, then the

Mode	Info	XSLT Precedence*				Translation		
BibTeX	The BibTeX Mode	base	up	down	remove	details	remove details	delete mode
		special	up	down	remove	details		
		person	up	down	remove	details		
		month	up	down	remove	details		
		number	up	down	remove	details		
		topic	up	down	remove	details		
		default	up	down	remove	details		
ModeX	Just an example	main	up	down	remove	details		delete mode

* Higher position means higher import precedence.

Figure 17: Export Mode Management Table

user can also specify the import precedence of each stylesheet by moving the stylesheets up and down in the list of all stylesheets that is presented in the Web interface (see Figure 17). The stylesheet with a higher position in the list will also have higher import precedence. The export into a certain format is realized by applying a temporary XSLT file that imports all stylesheets of this mode to the BibXML file. The content of such a temporary file is shown in Listing 22. XSLT specifies that the stylesheet that is imported last has the highest import precedence.

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="stylesheet_with_low_priority.xsl"/>
  <xsl:import href="stylesheet_with_medium_priority.xsl"/>
  <xsl:import href="stylesheet_with_high_priority.xsl"/>
  ...
</xsl:stylesheet>

```

Listing 22: Temporary XSLT Importing All Mode Stylesheets

The system creates this temporary XSLT file dynamically by using the information from the database. The user does never have to worry about this temporary file or the filenames of the stylesheets. Only the XSLT stylesheet list in the Web interface shown in Figure 17 must be ordered as desired and the rest is taken care of by the system.

In addition to the stylesheets, a special translation table can be added to a mode. Such a translation table is used to convert special characters into

their required representation in the export format. The translation table can be uploaded through the Web interface, is validated, and then stored in the system. An example of such a translation table that replaces special characters like ä by their L^AT_EX control sequences (in this case `\{a}`) for export into B^IB_TE_X, is shown in Listing 23.

```

<?xml version="1.0"?>
<translations xmlns="http://bibtexml.org/translations">
  <translation>
    <input>&#228;</input>
    <output>\{a}</output>
  </translation>
  <translation>
    <input>&#246;</input>
    <output>\{o}</output>
  </translation>
  <translation>
    <input>&#252;</input>
    <output>\{u}</output>
  </translation>
  ...
</translations>

```

Listing 23: B^IB_TE_X Translation Table (partial)

The translation itself is implemented in PHP and is executed before the XSLT stylesheets mentioned above are applied. For this purpose PHP uses the translation table and replaces all occurrences of the strings defined in the `<input>` elements by their counterparts in the `<output>` elements in all text nodes of the BibXML file that needs to be exported.

Performing the character translations before the XSLT stylesheets are applied, makes it possible to use the XML structures to translate only the bibliographic data (in the text nodes) and spare all markup. This would be much more difficult in a format like B^IB_TE_X.

4.8 Macro Management

As explained in Section 4.3, macros are stored in the database and have an XML value, a text value, an owner, and a name, which is used to reference them. The XML value is the actual value of the macro. The text value is generated mainly for search purposes by applying the stylesheet

import/special.xsl, which is shown in Listing 24, to the XML value. This stylesheet generates the default text value from the XML value, and most notably replaces all `specialtext` elements by the content of their `text` children (see Section 4.2.2 for more information).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:base="http://bibtexml.org/base">
  <xsl:output encoding="ISO-8859-1" omit-xml-declaration="yes"/>
  <xsl:template match="/*">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="base:macro">
    <xsl:copy-of select="."/>
  </xsl:template>
  <xsl:template match="base:special">
    <xsl:apply-templates select="base:text"/>
  </xsl:template>
</xsl:stylesheet>
```

Listing 24: XSLT Stylesheet *import/special.xsl*

The *Macro Management* in the Web interface can be used to get an overview over all available macros and their owners. New macros can be added by specifying a macro name (which must be unique) and a macro value. The owner of a macro or an administrator can delete the macro or delegate ownership to another user. As always, this Web interface is dynamically created by PHP with information from the database. Modification requests are transmitted to the server and executed by PHP by altering the database.

4.9 User Management and System Access

User management is implemented by using the three database tables shown in Figure 18. These tables can be managed conveniently through the menu option *User Management* in the Web interface. This menu option is only available to administrators. Administrators are the built-in user *administrator* as well as all users who belong to the built-in group *administrator*. Please consult the administrator guide in Appendix B for more information.

System access is restricted by using the session handling functions provided by PHP. Currently, PHP is configured to use cookies to keep track

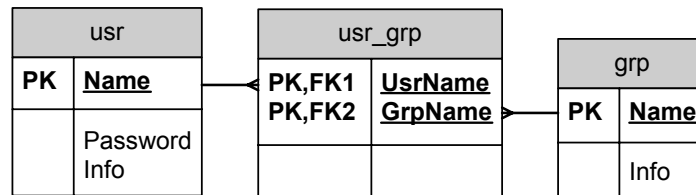


Figure 18: User Management Tables

of the user sessions. But it can be configured to use long URLs instead. Please consult the technical documentation in Appendix C for more information. When a user tries to log into the system, the specified login and password are compared to the values in the database, and access will only be granted if the login information is correct. The PHP session variables `$_SESSION["username"]` and `$_SESSION["user-is-admin"]` will only be set if the user is registered and logged in correctly. The `gatekeeper()` function, which is called at the beginning of each page, will use these session variables to determine if the user has access to a certain page or not. The `gatekeeper()` function also generates the navigation menu on top of each page, as administrators have more menu options available than regular users.

5 Results

The management system was tested extensively using an existing BIBTEX collection of 1643 entries and 450 macros. The corresponding BIBTEX file has a size of 558KB. In order to import the bibliographic data into the system, the BIBTEX file had first to be transformed into the BibXML format. This transformation was performed using tools provided by a former diploma thesis [4] and resulted in a 1.44MB BibXML file. The management system was running under Windows XP and used a 1.8GHz Pentium 4 processor and 512MB RAM. Under these conditions, it took the system 230 seconds to import the complete BibXML document into the empty system. 110 seconds were spent validating the document, and 120 seconds importing the data into the database. Further, all functions of the system were successfully tested using this imported data collection. With 1643 entries in the database, most search queries are completed within 5 seconds. But unsophisticated search queries can take a longer time. For example, searching for the character “e”, which is contained in every entry of the system, takes the system 40 seconds. Exporting all 1643 entries into the BibXML format was completed within 25 seconds and transforming this BibXML document into the BIBTEX format took the system 45 seconds.

We consider the high performance of the search routines to be a very positive result, because the search tool is expected to be the most frequently used tool of the system. Less impressive is the performance of the import process, but we think that 233 seconds for importing 1643 entries is an acceptable result. Ideas for speeding up this process are presented in Section 6.

6 Future Work

We regard our work as a first attempt to design and implement a centralized bibliography management system that motivates members of institutions such as the ETH Zürich to give up their established habit of maintaining their private bibliographies, and motivates them to store their bibliographic data in a centralized repository. Even though we can present a fully functioning system, we are aware of the fact that more research should be done in this area. Together with our tutor, Erik Wilde, we have identified the following issues that we consider to be worth investigating in more detail:

- *Web Forms for Importing and Modifying Data:* The implemented management system only allows users to import data by writing and importing complete BibXML documents describing the data. Furthermore, if users want to alter an existing entry, they have to delete the entire entry from the system, and import the new version of the entry as a BibXML document. Users would certainly appreciate the possibility to import and modify entries on a per-field level using dynamic Web forms.
- *BibSchema Classification and Cataloguing:* Currently, no attempt is made to prevent users from reinventing the wheel by defining multiple schemas for the same purpose. The problem of classifying and cataloguing existing schemas in a way that users reuse existing schemas is not an easy one. Our current (admittedly simple) way to deal with this problem is to restrict schema installation to system administrators, who are expected to check the schemas manually and thus prevent the emergence of redundant schemas.
- *Import Optimization:* Importing large amounts of records into our system is currently not performing very well due to our design of the import process. To speed up the import process, it would be possible to perform various optimizations. In a first step, it would be possible to use pre-compiled XSLT (Schematron validation uses XSLT, which could be pre-compiled). Going further, it would be possible to write custom validation code in a regular programming language. However, since validation must be open to extensions by new schemas, this would be a rather complex task.
- *Pluggable Import Filters:* In the same way as the system now supports pluggable export filters, it would be possible to support pluggable import filters, so that users can install such an import filter and then

import any kind of import data¹⁷, which as part of the import process is then transformed into the BibXML format required for internal storage.

- *XML Database Support*: The system as it is implemented now is based on a relational database, which brings with it all the problems of an XML-based data model stored in a relational database. It would be an interesting task to move the system to an XML database, which in particular would be interesting if the BibSchema extensions included complex XML structures that should be accessible via queries. Which directly leads to the next point:
- *XML Query Language (XQuery)*: Currently, the system accepts queries via a Web interface and maps them to SQL queries. This is sufficient as long as queries are targeted at field level. However, if queries need to recognize field structures, then it would be very useful to have a query language specialized for XML queries, such as XQuery. XQuery support would imply an underlying database supporting it, and the question whether this would be worth the effort highly depends on the kind of BibSchema extensions (and in particular, features to query that data) that should be supported. Looking at the current design of the system, this could also involve the following:
- *BibSchema Query Language*: In the same way as the system currently has its own schema language (which is mapped to XML standard technologies), the system could be extended to have its own query language, which would be specifically designed to support BibSchema with its extension mechanism, and possibly XQuery-style queries into BibSchema extensions.

¹⁷Depending on the support of import filters, this could either be any kind of XML (if only XSLT would be accepted), or any kind of text-based data (if a text-processing language such as Perl would be supported, too).

7 Conclusions

The concept described and implemented in this diploma thesis is a first approach towards a centralized system for managing bibliographic data. In our opinion, members of an institution would be very well advised to store their bibliographic data in such a centralized management system in order to make it accessible to other employees and benefit from the powerful tools provided by the system. Even though our implementation has proven to be a fully functioning system, we are aware of the fact that there are many interesting possibilities to further upgrade it with additional features.

During the design and implementation process of our project, we familiarized ourselves with a variety of different technologies. We put most of our effort into choosing and applying various XML technologies like DOM, XSLT, XML Schema, and Schematron. Furthermore, we learned a lot about PHP, MySQL, and the creation of dynamic Web applications.

References

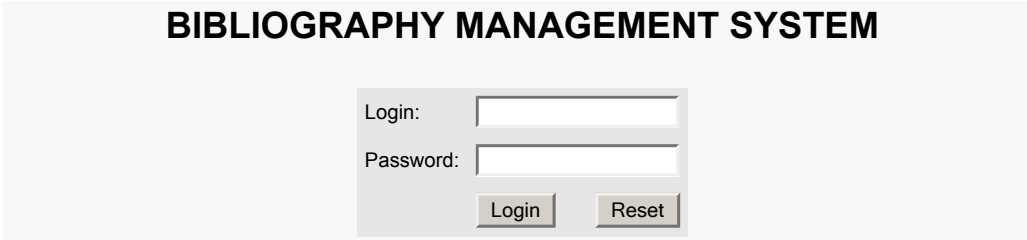
- [1] Richard Anderson, Mark Birbeck, and Michael Kay. *XML Professionell*. MITP-Verlag, Bonn, 2000.
- [2] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron. to be published as ISO/IEC 19757-3.
- [3] Michael Kay. *XSLT, Programmer's Reference*. Wrox Press, Birmingham, April 2001.
- [4] Brenno Lurati and Luca Previtali. BIB_TE_XXML: Design und Implementierung einer XML-basierten Lösung für BIB_TE_X-Literaturreferenzen. Master's thesis, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland, March 2001.
- [5] Oren Patashnik. BIB_TE_Xing. Technical report, Sun Microsystems, February 1988.
- [6] Eric van der Vlist. *XML Schema*. O'Reilly & Associates, Sebastopol, California, June 2002.
- [7] Priscilla Walmsley. *Definitive XML Schema*. Prentice Hall, Upper Saddle River, New Jersey, 2002.
- [8] Randy Jay Yager, George Reese, and Tim King. *MySQL & mSQL*. O'Reilly & Associates, Sebastopol, California, July 1999.

A User Guide

This user guide explains the features and the handling of the Web interface for the bibliography management system. For details about the various file formats and technical information, please consult Chapter 4 (*Implementation*) and Appendix C (*Technical Documentation*). If you experience any problems with the system or need additional information, feel free to ask the administrator of the system. We will use *italics* when referring to details depicted in the various figures of this user guide.

A.1 Login

To access the system, please obtain the URL of the system, your *Login* and your *Password* from the administrator and access the site with a browser of your choice. You should see the login page depicted in Figure 19. Please enter your *Login* and *Password* and hit the *Login* button.



BIBLIOGRAPHY MANAGEMENT SYSTEM

Login:

Password:

Figure 19: Login Page

A.2 Overview

After you successfully logged into the system, you see the overview page shown in Figure 20. You can always use the navigation bar on top of the page to access the various parts of the Web interface. In the navigation bar, the active menu point is highlighted. In the top left corner of the page, the active user name is displayed. The *Logout* command is used to end the current session and log out of the system. The components listed on the overview page (*Search*, *Selection*, *Data Import*, *Schema Overview*, *Macro Management*) are explained in the following sections.

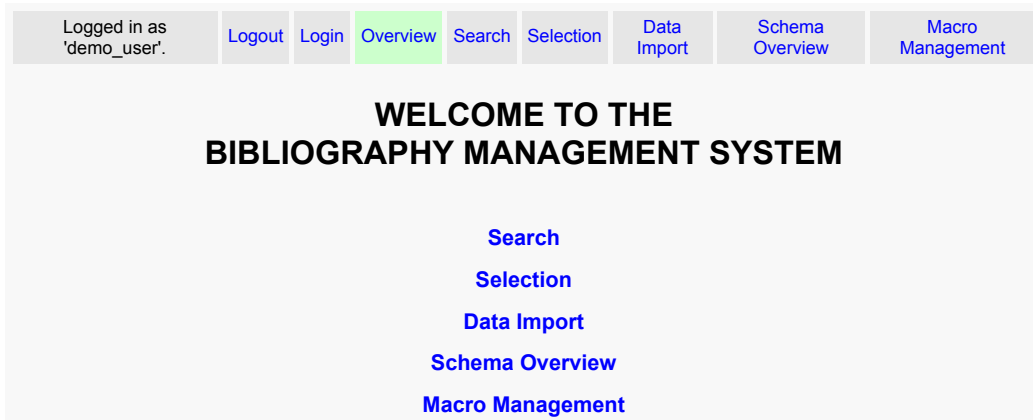


Figure 20: Overview Page

A.3 Search

To search the database of bibliographic information, two different search interfaces and an optional advanced search syntax (regular expressions) are at your disposal. These features are discussed in detail below.

A.3.1 Simple Search

The simple search interface is shown in Figure 21. It can be used for quick and simple search requests covering all fields and the keys of all entries in the database. Please note that unqualified search requests for words like “the” or for single characters should be avoided as they can take a very long time. The advanced search interface should be used whenever possible to get fast and precise search results. You can switch between the simple and the advanced search interface by using the appropriate link.

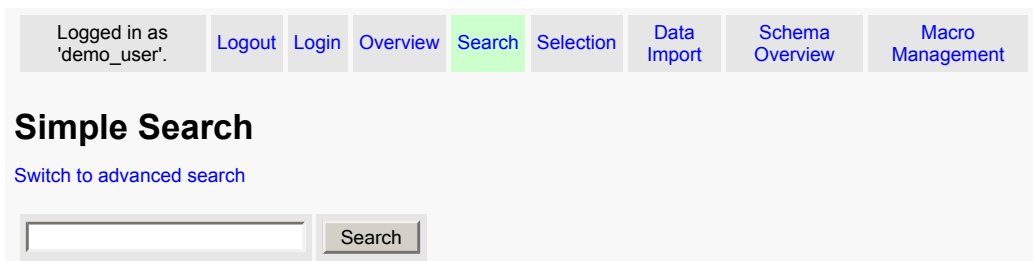


Figure 21: Simple Search Interface

A.3.2 Advanced Search

The advanced search interface can be used for specific searching and is shown in Figure 22. The search interface is divided into two parts: the upper part is used to search for information that is directly associated with entries (e.g., *Key*, *Owner*) and the lower part is used to search for information associated with fields (e.g., field *Value*, *Type*). These two parts can be connected with a logical *AND* or a logical *OR*. So, if you want to search for all entries with a certain *Key* that have fields of a certain *Type*, you have to use *AND* as connector. But if you want to search for all entries with a certain *Key* and for all entries with fields of a certain *Type*, you have to use *OR* as connector. The search options within each of the two parts can be thought of as connected with a logical *AND*.

The screenshot shows the 'Advanced Search' interface. At the top, there is a navigation bar with links: 'Logged in as 'demo_user'', 'Logout', 'Login', 'Overview', 'Search' (highlighted), 'Selection', 'Data Import', 'Schema Overview', and 'Macro Management'. Below the navigation bar is the title 'Advanced Search' and a link 'Switch to simple search'. The main search area is divided into two sections: 'Entry:' and 'Entry has Field:'. The 'Entry:' section contains fields for 'Key', 'Owner', 'Modifier', 'Group', 'Type', and 'Namespace', each with a dropdown menu and a radio button for search options. The 'Entry has Field:' section contains fields for 'Value', 'Type', 'Namespace', and 'IsPerson', each with a dropdown menu and a radio button for search options. At the bottom, there are 'Search' and 'Reset' buttons.

Entry:		
Key:	<input type="text"/>	<input checked="" type="radio"/> any <input type="radio"/> like <input type="radio"/> exact
Owner:	<input type="text" value="demo_admin"/>	'CTRL' for multiple selection
Modifier:	<input type="text" value="demo_admin"/>	'CTRL' for multiple selection
Group:	<input type="text" value="NULL"/>	'CTRL' for multiple selection
Type:	<input type="text" value="article"/>	'CTRL' for multiple selection
Namespace:	<input type="text" value="http://bibtexml.org/standard"/>	'CTRL' for multiple selection

AND

Entry has Field:		
Value:	<input type="text"/>	<input checked="" type="radio"/> any <input type="radio"/> like
Type:	<input type="text" value="address"/>	'CTRL' for multiple selection
Namespace:	<input type="text" value="http://bibtexml.org/standard"/>	'CTRL' for multiple selection
IsPerson:	<input type="text" value="ANY"/>	

Search Reset

Figure 22: Advanced Search Interface

The first form field, *Key*, is used to search for the key of an entry. If you select the radio button *any*, the system will search for any key. If you choose

like, the system will interpret your search expression as regular expression and if you choose *exact*, the system will only search for entries that exactly match the specified key. The following five form fields (*Owner*, *Modifier*, *Group*, *Type*, *Namespace*) are implemented as drop-down menus. The options of each drop-down menu are dynamically created from the database and do only contain the values that are actually present in the database. The additional wildcard option —*ANY*— in each menu can be selected if the search should not be restricted by the corresponding form field. You can also select multiple options of the drop-down menu by pressing “CTRL” on the keyboard while selecting all options.

The form field *Value* can be used to search for the value of a field. Again, the option *any* serves as a wildcard and the option *like* instructs the system to treat your search expression as a regular expression. Of course, you can also enter a simple word and the system will find all fields containing this word.

A.3.3 Regular Expressions

As mentioned above, several search fields support regular expressions. Please use the examples in Table 25 as a guideline for regular search expressions or visit the respective MySQL page¹⁸ for more information. You can use a backslash in front of the characters used for regular expressions (e.g., +?*) to escape them and their special meaning.

Expression	Meaning
<code>^a</code>	Match fields beginning with a .
<code>a\$</code>	Match fields ending with a .
<code>.</code>	Match any character.
<code>a*</code>	Match any sequence of zero or more a characters.
<code>a+</code>	Match any sequence of one or more a characters.
<code>a?</code>	Match either zero or one a character.
<code>abc xyz</code>	Match either of the sequences abc or xyz .
<code>(abc)</code>	Group the sequence abc (see the combined example below).
<code>(abc)*</code>	Match zero or more instances of the sequence abc .

Table 25: Regular Expression Syntax

¹⁸MySQL Regular Expressions: <http://www.mysql.com/doc/en/Regexp.html>

A.3.4 Search Results

Search results are presented in a table showing the matching entries (see Figure 23). Each entry can be expanded to show all its fields by clicking on the blue + on the left of the corresponding row. To expand all entries, click on the blue + in the top row. Please use the corresponding blue -- symbols to collapse entries. Entries and fields can be sorted by column values by clicking on the header of the corresponding column.

Logged in as 'demo_user'. [Logout](#) [Login](#) [Overview](#) [Search](#) [Selection](#) [Data Import](#) [Schema Overview](#) [Macro Management](#)

Advanced Search

[Switch to simple search](#)

Search results

+ --	entry_key	entry_type	entry_namespace	entry_owner	entry_modifier	entry_group	<input type="checkbox"/>
--	wil03a	article	http://bibtexml.org/standard	demo_user	demo_user		<input type="checkbox"/>
	thru_ref	field_type	field_namespace	field_textvalue	field_xmlvalue	person_info	
		author *	http://bibtexml.org/standard			Erik Wilde	
		journal	http://bibtexml.org/standard	iX	iX		
		month	http://bibtexml.org/standard	February	February		
		number	http://bibtexml.org/standard	2	2		
		pages	http://bibtexml.org/standard	12	12		
		title	http://bibtexml.org/standard	Bericht von der XML 2002 in Baltimore	Bericht von der XML 2002 in Baltimore		
		volume	http://bibtexml.org/standard	16	16		
		year	http://bibtexml.org/standard	2003	2003		
+	wil03b	inproceedings	http://bibtexml.org/standard	demo_admin	demo_admin		<input type="checkbox"/>
+	wil03c	techreport	http://bibtexml.org/standard	demo_user	demo_user		<input type="checkbox"/>

What would you like to do with the selected entries?

Include referenced entries in export
 Include referenced macros in export

Figure 23: Search Results

To select entries for processing, use the checkboxes on the right of the table. You can use the checkbox in the top row to toggle all checkboxes at once. Once selected, entries can be added to the selection, deleted from the database, or exported, by pressing the appropriate button. The details of these operations will be explained in the following section.

A.4 Selection

The *Selection* works like a shopping cart: entries from various search results can be collected for later reuse or processing by adding them to the selection. The current content of the *Selection* is displayed in the same type of table as used for search results (see Figure 24).

Logged in as 'demo_user'. [Logout](#) [Login](#) [Overview](#) [Search](#) **Selection** [Data Import](#) [Schema Overview](#) [Macro Management](#)

Selection

Current content

+ --	entry_key	entry_type	entry_namespace	entry_owner	entry_modifier	entry_group	<input type="checkbox"/>
+	and01	inproceedings	http://bibtextml.org/standard	demo_admin	demo_admin		<input type="checkbox"/>
+	bau97	phdthesis	http://bibtextml.org/standard	demo_admin	demo_admin		<input type="checkbox"/>
--	wil03a	article	http://bibtextml.org/standard	demo_user	demo_user		<input type="checkbox"/>

thru_ref	field_type	field_namespace	field_textvalue	field_xmlvalue	person_info
	author *	http://bibtextml.org/standard			Erik Wilde
	journal	http://bibtextml.org/standard	iX	iX	
	month	http://bibtextml.org/standard	February	February	
	number	http://bibtextml.org/standard	2	2	
	pages	http://bibtextml.org/standard	12	12	
	title	http://bibtextml.org/standard	Bericht von der XML 2002 in Baltimore	Bericht von der XML 2002 in Baltimore	
	volume	http://bibtextml.org/standard	16	16	
	year	http://bibtextml.org/standard	2003	2003	

What would you like to do with the selected entries?

BibTeX
 Include referenced entries in export
 Include referenced macros in export

[Search entries](#)

Figure 24: Selection

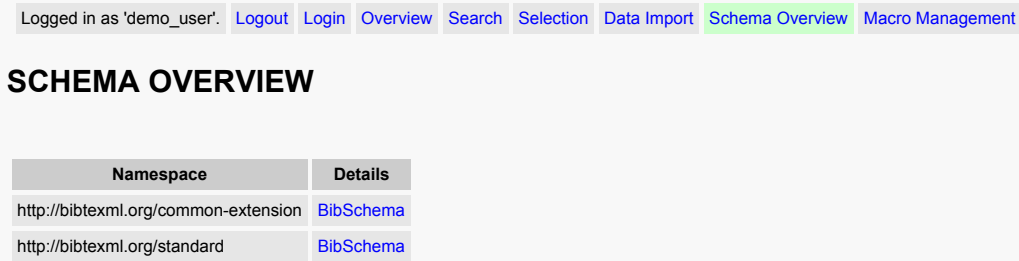
To remove entries from the *Selection*, select them using the checkboxes on the right side of the table, and press the *Remove from selection* button. These entries will be removed from your *Selection*, but not deleted from the database. To delete entries from the database, select them and press the *Delete from database* button. All selected entries will be deleted, if you have write permission for the corresponding entry.

All available export formats are listed in the drop-down menu left to the *Export* button. Please contact the system administrator if you want to install an additional export format. To export entries, select them, choose the appropriate export format in the drop-down menu, select the desired

option concerning macros and cross references, and press the *Export* button. The system will generate the desired export file and provide you with a link to access it.

A.5 Schema Overview

Before being able to import BibXML documents of a certain format, the system must have a suitable schema to validate them. An overview over all available schemas is available on the *Schema Overview* page (see Figure 20). You can click on the *Bib Schema* link for each namespace to get detailed information about the corresponding schema. Please contact the system administrator if you need to install a new schema.



Namespace	Details
http://bibtexml.org/common-extension	BibSchema
http://bibtexml.org/standard	BibSchema

Figure 25: Schema Overview

A.6 Data Import

Bibliographic data can be imported into the system through the menu point *Data Import* in the Web Interface. When you start a new import session, the import page looks as shown in Figure 26. You can use the *Browse* button to find the BibXML file you would like to import. After selecting the behavior of the system in case of a macro conflict through the radio buttons, you can start the import session by pressing the *Import File* button. The file is uploaded to the server and validated against the available schemas. Please be patient, this validation process can take a few minutes for very large documents. If the uploaded BibXML document is valid, the system begins with the actual import into the database. You are informed about the import progress by a status bar and additional information about each imported entry.

If the system encounters a problem because a certain entry key already exists in the database, you will be asked for action as shown in Figure 27.

Logged in as 'demo_user'. [Logout](#) [Login](#) [Overview](#) [Search](#) [Selection](#) **Data Import** [Schema Overview](#) [Macro Management](#)

DATA IMPORT

Please select the file you would like to import.

On macro conflict: output error use macro from database

Path:

Figure 26: Start of Data Import

Please make a choice by selecting the appropriate radio button, and submit your decision to the system to continue with the import process.

Logged in as 'demo_user'. [Logout](#) [Login](#) [Overview](#) [Search](#) [Selection](#) **Data Import** [Schema Overview](#) [Macro Management](#)

DATA IMPORT

Entry import status:

Importing entry number 1 with key 'wil03a'
 Entry already exists! Existing entry was replaced by request.
 Notice: Owner not specified. Owner has been set to 'demo_user'.
 Notice: Group name not specified. The system inserted 'NULL' instead.

Importing entry number 2 with key 'wil03c'
 An entry with key 'wil03c' already exists in the database.

What would you like to do with the existing entry?

replace
 always replace
 keep
 always keep
 keep and add new entry with key

Figure 27: Data Import Confirmation Dialogue

A.7 Macro Management

A list of all available macro names, macro values, and macro owners is available on the macro management page (if you do not know how macros can be used in this system, please ask the system administrator or consult Section 4.1 for detailed information). An example of such a macro management page is given in Figure 28. You can add a new macro by specifying a unique macro name and a macro value in the form at the bottom of the page, and submitting the form to the server. The macros that belong to you can be selected through a checkbox at the right hand side of the corresponding row. Selected macros can be deleted or ownership can be transferred to other users through the form buttons below the macro table.

Logged in as 'demo_user'. [Logout](#) [Login](#) [Overview](#) [Search](#) [Selection](#) [Data Import](#) [Schema Overview](#) [Macro Management](#)

MACRO MANAGEMENT

Macro Name	Macro Value	Macro Owner	<input type="checkbox"/>
berne	Berne, Switzerland	demo_admin	
ethz	Swiss Federal Institute of Technology	demo_admin	
london	London, UK	demo_admin	
w3c	World Wide Web Consortium	demo_user	<input type="checkbox"/>
xml	Extensible Markup Language	demo_user	<input type="checkbox"/>
zueri	Zürich, Switzerland	smith	

What would you like to do with the selected macros?

Macro Name	Macro Value
<input type="text"/>	<input type="text"/>

Figure 28: Macro Management

B Administrator Guide

The administrator guide describes the parts of the Web interface that can only be accessed by system administrators. This includes the extended *Overview* page and the management pages *Schema Management*, *Export Management*, *User Management*, and *MySQL Admin*. All remaining pages of the Web interface can also be accessed by normal users and are described in Appendix A. For details about the various file formats and technical information, please consult Chapter 4 (*Implementation*) and Appendix C (*Technical Documentation*). We will use *italics* when referring to details depicted in the various figures of this administrator guide.

B.1 Overview

After you successfully logged into the system using your administrator account, you see the overview page shown in Figure 29. You can always use the navigation bar on top of the page to access the various parts of the Web interface. In this navigation bar, the active menu point is highlighted. You have the possibility to choose among all menu points introduced in the *User Guide* as well as the additional menu points *Schema Management*, *Export Management*, *User Management*, and *MySQL Admin*.



Figure 29: Administrator Overview Page

B.2 Schema Management

As an administrator, you have the possibility to extend the BibXML format by defining new entry and field elements in BibSchema schemas. These BibSchema schemas are imported into the management system using the *Schema Import* form at the bottom of the *Schema Management* page (see Figure 30). This form also allows you to only validate but not import a BibSchema schemas by selecting the radio button *Only validate schema*. There is one important point about importing schema extensions you should be aware of: the management system makes no attempt to prevent administrators from defining multiple schemas for the same purpose. Consequently, you are expected to check the schemas manually in order to prevent the emergence of redundant schemas. The table *Schema Overview* above the import form shows an overview of the imported schema extensions. For each schema extension, the target namespace is presented and the original BibSchema schemas as well as the transformed XML Schema and Schematron documents can be viewed by following the corresponding *Bib Schema*, *XML Schema*, or *Schematron* links. A schema can be removed from the system by pressing *Delete* at the right side of the appropriate row. When deleting a schema, not only corresponding schema documents are removed from the filesystem, but also entries, fields, and special items in the target namespace of the schema in question are deleted from the database:

- An entry is deleted with all its fields from the database, if it is in the target namespace of the removed schema.
- An entry is deleted with all its fields from the database, if at least one of its fields is in the target namespace of the removed schema.
- A special item is deleted from the database, if it is in the target namespace of the removed schema. The entry and the field containing this deleted special item are not removed from the database, unless they are in the target namespace of the removed schema.

B.3 Export Management

The *Export Management* page (see Figure 31) gives you the possibility to add new export modes to the system. Available export modes are presented in a table at the top of the page. In this table, each export mode is associated with

Logged in as 'demo_admin'. [Logout](#) [Login](#) [Overview](#) [Search](#) [Selection](#) [Data Import](#) [Schema Management](#) [Export Management](#) [Macro Management](#) [User Management](#) [MySQL Admin](#)

SCHEMA MANAGEMENT

Schema Overview

Namespace	Details			
http://bibtexml.org/common-extension	BibSchema	XML Schema	Schematron	Delete
http://bibtexml.org/standard	BibSchema	XML Schema	Schematron	Delete


Schema Import

Select the Bib Schema you want to import or just validate:

Import schema
 Only validate schema

Path:

Figure 30: Schema Management

a mode name (first column of the table), additional information¹⁹ about the mode (second column), at least one XSLT stylesheet to be applied when exporting data (third column), and an optional translation table for the translation of special characters (fourth column). XSLT stylesheets of a certain mode are represented by their assigned names. They can be viewed by following the corresponding *details* links  deleted from the filesystem by pressing *remove*. Furthermore, the position of a stylesheet within the table defines its precedence among other stylesheets of the same mode. The relative precedence of a stylesheet can be changed by clicking on *up* or *down*: *up* raises the precedence of a stylesheet (and the position of the stylesheet within the table) and *down* lowers it. Translation tables can be viewed by pressing *details* and deleted by clicking on *remove* in the *Translation* column. Entire modes are removed from the system by pressing *delete mode* on the right of the table. New modes, XSLT stylesheets, and translation tables are added to the system by making use of the three forms in the lower half of the *Export Management* page.

¹⁹For example, this additional information about a mode can include the name of the person who created the corresponding stylesheets, special features of the mode or a version number.

Logged in as 'demo_admin'. [Logout](#) [Login](#) [Overview](#) [Search](#) [Selection](#) [Data Import](#) [Schema Management](#) [Export Management](#) [Macro Management](#) [User Management](#) [MySQL Admin](#)

EXPORT FORMAT MANAGEMENT

Mode	Info	XSLT Precedence*				Translation	
BibTeX	The BibTeX Mode	base	up	down	remove	details	remove details delete mode
		special	up	down	remove	details	
		person	up	down	remove	details	
		month	up	down	remove	details	
		number	up	down	remove	details	
		topic	up	down	remove	details	
		default	up	down	remove	details	
ModeX	Example Mode	base	up	down	remove	details	delete mode

* Higher position means higher import precedence.

Mode Name* **Mode Info** [Add Mode](#)

* The first 3 characters of the mode name will be the extension of the export-file for this mode.

Add XSLT with name to mode [BibTeX](#)

Path: [Browse...](#) [Add XSLT](#)

Add translation table to mode [BibTeX](#)

Path: [Browse...](#) [Add Table](#)

Figure 31: Export Management

B.4 User Management

The *User Management* page (see Figure 32) allows you to manage the users of the system. Only registered users are granted access to the bibliography management system. New users can be added to the system by specifying their *User Name*, *User Info*, and *Password* in the top form of the *User Management* page. The table at the top of the page lists all registered users of the system. Apart from the built-in user *administrator*, each user can be removed from the system by pressing *delete user* at the right of the table. Groups can be added to the system using the form at the bottom of the page. Entered groups are represented by their *Group Name* and *Group Info* in the second table of the *User Management* page. The fourth column of this table allows you to add registered users to a certain group, and all users of each group are listed in the *Members* column. The *administrator* group is a built-in group that defines the administrators of the system. Apart from the *administrator* group, each group can be removed from the system by pressing *delete group* at the right of the table.

Logged in as 'demo_admin'. [Logout](#) [Login](#) [Overview](#) [Search](#) [Selection](#) [Data Import](#) [Schema Management](#) [Export Management](#) [Macro Management](#) [User Management](#) [MySQL Admin](#)

USER MANAGEMENT

User Name	User Info	
administrator	built-in db admin	protected
anderson	Mrs. Anderson	delete user
demo_admin	Demo Admin	delete user
demo_user	Demo User	delete user
smith	Mr. Smith	delete user

User Name	User Info	Password	
<input type="text"/>	<input type="text"/>	<input type="password"/>	<input type="button" value="add user"/>

Group Name	Group Info	Members	
administrator	built-in db admin group	demo_admin remove	<input type="text" value="protected"/> <input type="button" value="Add"/> protected
demo_group	demo user group	demo_user remove	<input type="text" value="delete group"/> <input type="button" value="Add"/> delete group

Group Name	Group Info	
<input type="text"/>	<input type="text"/>	<input type="button" value="add group"/>

Figure 32: User Management

B.5 MySQL Admin

The *MySQL Admin* page (see Figure 33) lets you directly access the *biblio* database of the management system using *phpMyAdmin*²⁰. *phpMyAdmin* is a powerful database management tool that allows you to create and drop databases, create, copy, drop and alter tables, and do many more things with MySQL databases. We recommend to use *phpMyAdmin* very carefully, because you are not prevented from corrupting the management system when directly manipulating the database.

²⁰See <http://www.phpmyadmin.net/>

phpMyAdmin

Home

biblio (12)

biblio

- entry
- entrytype
- export_mode
- export_xslt
- field
- fieldtype
- grp
- macro
- namespace
- person
- usr
- usr_grp

Database *biblio* running on localhost

Structure SQL Export Search Query Drop

Table	Action	Records	Type	Size
<input type="checkbox"/> entry	Browse Select Insert Properties Drop Empty	1,642	InnoDB	544.0 KB
<input type="checkbox"/> entrytype	Browse Select Insert Properties Drop Empty	16	InnoDB	48.0 KB
<input type="checkbox"/> export_mode	Browse Select Insert Properties Drop Empty	2	InnoDB	32.0 KB
<input type="checkbox"/> export_xslt	Browse Select Insert Properties Drop Empty	8	InnoDB	64.0 KB
<input type="checkbox"/> field	Browse Select Insert Properties Drop Empty	10,298	InnoDB	2.2 MB
<input type="checkbox"/> fieldtype	Browse Select Insert Properties Drop Empty	37	InnoDB	48.0 KB
<input type="checkbox"/> grp	Browse Select Insert Properties Drop Empty	2	InnoDB	32.0 KB
<input type="checkbox"/> macro	Browse Select Insert Properties Drop Empty	451	InnoDB	128.0 KB
<input type="checkbox"/> namespace	Browse Select Insert Properties Drop Empty	2	InnoDB	32.0 KB
<input type="checkbox"/> person	Browse Select Insert Properties Drop Empty	1,762	InnoDB	112.0 KB
<input type="checkbox"/> usr	Browse Select Insert Properties Drop Empty	5	InnoDB	32.0 KB
<input type="checkbox"/> usr_grp	Browse Select Insert Properties Drop Empty	2	InnoDB	48.0 KB
12 table(s)	Sum	14,227	--	3.3 MB

Check All / Uncheck All

Figure 33: phpMyAdmin

C Technical Documentation

In this appendix, the setup and the maintenance of the management system are described, and we introduce the built-in XML documents, schema extensions for the most common entry and field types, and export filters for the BIB_TE_X format. Furthermore, the PHP functions used by the management system are formally described and their interfaces are presented. The corresponding source code can be found on the enclosed CD-ROM.

C.1 System Setup

The installation of the bibliography management system itself is very simple and is explained in Section C.1.7. The more challenging task is to install all required programs and software packages. The system relies on the following software being installed and configured correctly:

- Apache (HTTP server)
- PHP (scripting language)
- MySQL (relational database)
- Saxon (XSLT processor)
- Xerces (XML parser / schema validator)
- Java 2 runtime environment (for Saxon and Xerces)

The installation guide for each of these software packages should be consulted to install the software correctly. The following sections contain additional information concerning the installation and configuration of each package. Some information is Windows XP specific and needs to be adapted for other platforms.

All used software packages (for Windows XP) are available in the directory *Software* on the CD ROM accompanying this report (please see Appendix D). The CD ROM also contains a directory *Configuration* with example configuration files for Apache, PHP, and MySQL. All files required for the bibliography management system itself are available in the directory *SystemFiles* on the CD ROM.

The following installation directories for the software packages are assumed: *c:/apache/* for Apache, *c:/php/* for PHP, and *c:/mysql/* for MySQL. If different installation directories are used, all paths in the following sections must be changed accordingly.

C.1.1 Apache

Apache's HTTP server²¹ must be downloaded and installed. The configuration file *httpd.conf* should contain the following commands to load the PHP module and process the corresponding file types:

```
LoadModule php4_module c:/php/sapi/php4apache.dll
AddType application/x-httpd-php .php .htm .html .xml
```

To correctly load the index file of the system, the following command must be added to the *httpd.conf* file as well:

```
DirectoryIndex index.php index.htm index.html
```

The server should now be ready for the installation of PHP.

C.1.2 PHP

To enable the execution of PHP scripts, PHP²² (version 4.2.3 or higher) must be downloaded and installed. The configuration file for PHP (*php.ini*) should contain the following commands:

```
doc_root = c:/apache/htdocs/ ; specify HTTP server document root
file_uploads = 0n           ; allow HTTP file uploads
upload_max_filesize = 20M   ; maximum size for uploaded files = 20MB
extension=php_domxml.dll    ; load DOM extension
extension=php_xslt.dll      ; load XSLT extension
max_execution_time = 600    ; set maximum execution time to 10 minutes
memory_limit = 128MB        ; limit memory consumption to 128MB
```

To avoid problems, Sablotron²³ (PHP's built-in XSLT processor) and Expat²⁴ (XML parser used by Sablotron) should be upgraded to their newest versions. This step might not be necessary for PHP versions newer than 4.2.3. Instead of downloading these upgrades, all DLLs from the directory *XSLTdlls* on the CD ROM can be copied to *c:/php/dlls/*.

To complete the installation of PHP, all files from the directory *c:/php/dlls/* should be copied to a directory where they can be found by the OS (e.g., *c:/windows/system32/*), and the Apache server must be restarted.

²¹ Available at <http://httpd.apache.org/>

²² Available at <http://www.php.net/>

²³ Available at http://www.gingerall.com/charlie/ga/xml/p_sab.xml

²⁴ Available at <http://sourceforge.net/projects/expat>

C.1.3 MySQL

MySQL²⁵ (version 3.23.52 or higher) in its “max” distribution must be downloaded and installed. To enable the support for InnoDB tables, the configuration file *my.cnf* from the directory *Configuration* on the CD ROM can be used.

The directory `c:/mysql/bin/` should be included in the *PATH* variable of the system. To install MySQL as a service, the following command must be executed:

```
mysqld-max-nt --install
```

The service can then be started, and the MySQL database can be queried using the `mysql` client.

C.1.4 Saxon

Saxon can be downloaded from its project page²⁶ and installed into a directory of choice. The path to all *.jar* files in this directory must then be added to the *CLASS_PATH* variable to make Saxon available to other programs.

C.1.5 Xerces

Xerces can be downloaded from its project page²⁷ and installed into a directory of choice. The path to all *.jar* files in this directory must then be added to the *CLASS_PATH* variable to make Xerces available to other programs.

C.1.6 Java 2

Since Saxon and Xerces are implemented in Java 2, the corresponding Java 2 runtime environment (version 1.3 or higher) must be installed. It can be downloaded from Sun²⁸.

²⁵ Available at <http://www.mysql.com/>

²⁶ Available at <http://saxon.sourceforge.net/>

²⁷ Available at <http://xml.apache.org/xerces2-j/>

²⁸ Available at <http://java.sun.com/>

C.1.7 Bibliography Management System

To install the bibliography management system itself, the content of the directory *SystemFiles* needs to be copied to the Apache document root directory (e.g., *c:/apache/htdocs/*) or a subdirectory of it (e.g., *c:/apache/htdocs/bib/*).

In a next step, the bibliography database can be created using the script file *create.txt* in the root directory of the system (e.g., *c:/apache/htdocs/bib/create.txt*). The following command creates the database:

```
mysql < create.txt
```

Then, the first few lines of the file *include.php* need to be edited, so that the PHP variable `$bib_root` points to the system's root directory. Additionally, a few parameters for the database connection need to be specified (see comments in the *include.php* file itself).

As a last step, the correct host name and a few parameters for the database connection need to be specified in the phpMyAdmin²⁹ configuration file (e.g., *c:/apache/htdocs/bib/admin/config.inc.php*). Again, all necessary comments are given directly in the configuration file itself.

System setup is now complete! The system can be accessed through the Web interface with the pre-defined users specified in the file *create.txt*. For the sake of system security, this file should be moved to a safe place after installation.

The administrator can now proceed to the *Schema Management* and upload the schemas *schema_standard.xml* and *schema_extension.xml* from the extensions directory (e.g., *c:/apache/htdocs/bib/extensions/*). Additionally, the export filters for BIB_TE_X can be installed as described in section C.3.2.

C.2 System Maintenance

Apart from the regular management tasks like managing users, schemas, and export filters, the system administrator should empty the session data directory (e.g., *c:/apache/htdocs/bib/session_data/*) from time to time to clean up leftovers from unterminated sessions.

²⁹More information about phpMyAdmin: <http://www.phpmyadmin.net/>

C.3 Most Common Extensions

The presented management system is a highly extensible system in the sense that it allows users to define their own entry and field elements by extending the BibXML format accordingly. Furthermore, users have the possibility to install pluggable export filters in order to define their individual export formats. We have implemented schema extensions and export filters for the widely-used BIB_TE_X format. These extensions are presented in the following two sections. The corresponding files are located in the *extensions* subdirectory.

C.3.1 BibXML Extensions

Standard Entry and Field Types

All standard entry and field types³⁰ of the BIB_TE_X format are defined by the BibSchema document *schema_standard.xml* in the <http://bibtexml.org/standard> namespace. This includes the following 14 entry and 24 field types:

- **Standard entry types:** *article, book, booklet, conference, inbook, in-collection, inproceedings, manual, mastersthesis, misc, phdthesis, proceedings, techreport, unpublished*
- **Standard field types:** *address, annotate, author, booktitle, chapter, crossref, edition, editor, howpublished, institution, journal, key, month, note, number, organization, pages, publisher, school, series, title, type, volume, year*

Other Entry and Field Types

Besides these standard BIB_TE_X types, there are many other entry and field types defined by individual users. We defined the following types in the BibSchema document *schema_extension.xml*:

- **Other entry types:** *collection, patent*
- **Other field types:** *affiliation, abstract, contents, copyright, isbn, issn, keywords, language, location, lccn, mrnumber, price, size, url*

It is the system administrator's job to import these BibSchema documents into the system using the *Schema Management* page.

³⁰See <http://www.eeng.dcu.ie/local-docs/btxdocs/btxdoc/btxdoc/btxdoc.html>

C.3.2 Export Filters for BIB_TE_X

Pluggable export filters for the export into the BIB_TE_X format are implemented by the following six stylesheets:

base.xsl

is the master stylesheet with a template matching the root of the BibXML document to export. Its templates write the basic structure of the BIB_TE_X output and then apply templates defined by other stylesheets that match the individual field elements. Using this concept, it is very easy to write and import new stylesheets for individual field elements.

special.xsl

has one template that matches `b:special` elements and replaces them with the value of their `b:tex` children. This way, special text fields are correctly transformed into their L^AT_EX equivalents.


person.xsl

collects in each entry all person field elements of the same type and writes them in one BIB_TE_X person field (of type *author* or *editor*). Special attention had to be paid to the complex format of BIB_TE_X person field types.

month.xsl

matches *month* field elements and writes them to the output. During this process, it takes BIB_TE_X macros for months into account.

number.xsl

 matches field elements that contain a number only. The number is output omitting the usual brackets or quotation marks around field values.

default.xsl

matches all field elements that have not been matched by special templates.

It is the system administrator's job to import these stylesheets into the system using the *Export Management* page. When importing, it has to be paid attention to the chosen import precedence of the stylesheets. The listing of stylesheets above reflects the required import precedence for the presented stylesheets: *base.xsl* must have the highest and *default.xsl* the lowest import precedence. Furthermore, the translation table *translation.xml* has been im-

plemented. This table translates special characters to their corresponding L^AT_EX control sequences and can also be imported into the system using the *Export Management* page.

C.4 Built-in XML Documents

This section introduces the individual built-in XML documents grouped by their parent directories. Table 26 presents file extensions and their interpretations.

File extension	Interpretation
<i>.dtd</i>	Document Type Definition
<i>.sch</i>	Schematron document
<i>.xml</i>	Translation table or general XML instance
<i>.xsd</i>	XML Schema document
<i>.xsl</i>	XSLT stylesheet

Table 26: File Extensions

C.4.1 import

Subdirectory *import* contains two XSLT stylesheets used during the import and search process.

biblio.xsl

is applied to an already validated BibXML document during the import process. The *biblio.xsl* stylesheet transforms the BibXML document into a PHP document that contains all bibliographic data from the BibXML document, as well as instructions on how to import this data into the system. The resulting PHP document is included and executed by the main import file *import.php*.

special.xsl

is used to generate the text value out of the XML value of a field or macro. This stylesheet is used during the import process and to display search results. Please see Sections 4.2.2 and 4.8 for more information.

C.4.2 schema

Subdirectory *schema* includes all XML documents participating in the validation of imported documents (most notably imported BibSchema and BibXML documents).

datatypes.dtd³¹

is a DTD for XML Schema Datatypes. It is used to validate XML Schema documents and can not be used on its own but is intended only for incorporation in *XMLSchema.dtd*.

dummy_instance.xml

is a dummy XML document that is used to validate XML Schema documents with Xerces. Xerces does not allow the direct validation of XML Schema documents, but forces a program to create a dummy instance and validate it against the given XML Schema document, whereby also the schema gets validated.

merge_prep_exp.xsl, merge_prep_inst1.xsl, merge_prep_inst2.xsl

are used in the process of validating BibXML documents (see function *validate_instance()*). In order to merge an uploaded BibXML document with its external cross referenced entries for denormalization, these three stylesheets perform the required format transformations.

prevalidation1.sch

is a Schematron document ensuring that an imported BibXML document meets certain basic constraints before being further processed.

prevalidation2.sch

is a Schematron document ensuring that an imported BibXML document does not cross reference undefined entries or nest cross references.

resolveCrossrefs.xsl

resolves all cross references of a BibXML document by defining appropriate templates.

resolveMacros.xsl

resolves all macro references of a BibXML document by defining appropriate templates.

³¹Available online at <http://www.w3.org/2001/datatypes.dtd>

schematron1-5.xsd³²

is used together with *xml.xsd* to validate Schematron documents.

schematron1ForBibSchema.sch

ensures that namespace declarations in the imported BibSchema document are correct (recall the special conventions about declaring namespaces in BibSchema documents from Section 4.2.2).

schematron2ForBibSchema.sch

is used with *xmlSchemaForBibSchema.xsd* to validate the detailed structure of an imported BibSchema document.

schematronBase.sch

is used with *xmlSchemaBase.xsd* to validate the basic document structure that is common to all imported BibXML documents.

schematronTransform.xsl

implements the transformation of a valid, user-defined BibSchema document into an appropriate Schematron document.

skeleton1-5.xsl³³

is an XSLT implementation of Schematron 1.5 that is used in conjunction with the XSLT processor *Saxon*.

translation.xsd

is an XML Schema document that defines the format of a translation table.

xml.xsd³⁴

is used together with *schematron1-5.xsd* to validate Schematron documents.

XMLSchema.dtd³⁵

is a DTD for XML Schema Structures. It is used to validate XML Schema documents and incorporates *datatypes.dtd*.

XMLSchema.xsd³⁶

is an XML Schema schema for XML Schema Structures. It is used to validate XML Schema documents.

³²Available online at <http://www.ascc.net/xml/schematron/schematron1-5.xsd>

³³Available online at <http://www.ascc.net/xml/schematron/1.5/skeleton1-5.xsl>

³⁴Available online at <http://www.w3.org/2001/xml.xsd>

³⁵Available online at <http://www.w3.org/2001/XMLSchema.dtd>

³⁶Available online at <http://www.w3.org/2001/XMLSchema.xsd>

xmlSchemaBase.xsd

is used with *schematronBase.sch* to validate the basic document structure that is common to all imported BibXML documents.

xmlSchemaForBibSchema.xsd

is used with *schematron2ForBibSchema.xsd* to validate the detailed structure of an imported BibSchema document.

xmlSchemaTransform.xsl

implements the transformation of a valid, user-defined BibSchema document into an appropriate XML Schema document.

C.5 PHP Functions

In this section, the PHP files used by the management system are introduced, and their functions are formally described.

C.5.1 export_detail.php

When clicking on a *details* link associated with a certain stylesheet or translation table on the *Export Management* page, the PHP script *export_detail.php* creates a new window displaying the corresponding stylesheet or translation table.

C.5.2 export_management.php

This file generates the page to upload and manage the export modes, their stylesheets, and translation tables. It includes the files *include.php* and *schema/validation.php* and makes use of the functions described below:

change_priority

```
void change_priority(int xslt_id, string up_or_down)
```

is used to change the priority of the XSLT stylesheet identified by *xslt_id*. The variable *up_or_down* determines whether the priority should be raised or lowered.

clean_xslt()

```
functionType clean_xslt()
```

is used to clean up the priorities of all stylesheets by assigning the lowest possible integer priority (unique in each mode) to each stylesheet.

C.5.3 fast_search.php

This PHP file generates the simple and the advanced search interfaces, handles search requests, and presents search results. It includes the files *include.php*, *include_fast_search.php*, and *include_export.php*.

search

`void search(string type)`

calls the search functions *simple_search()* or *advanced_search()* according to the value of *type* (“simple” or “advanced”). As soon as the search results are available, they are presented using the function *display_results()*.

C.5.4 import.php

This file implements the rather complicated import process. Please consult Section 4.4 for an abstract view of the entire procedure. The file includes *include.php* to restrict access to the import page and *schema/validation.php* to validate import files.

ask_replace_opt

`void ask_replace_opt(string key)`

asks the user to make a decision if the *key* of an import entry already exists in the database. The function makes use of the function *user_has_write_rights()* to determine the available options in the generated HTTP form.

display_progress_bar

`void display_progress_bar()`

generates the dynamic import status bar that informs the user about the import progress.

init_entry_vars

`void init_entry_vars()`

initializes the global variables used in the temporary import PHP file generated by the *import/biblio.xsl* stylesheet.

init_import_session

```
void init_import_session()
```

initializes the session variables used during an import session.

insert_entry

```
int insert_entry(int typeid)
```

is called from the temporary import PHP file generated by the *import/biblio.xml* stylesheet. It inserts a new entry into the database. The function uses the global variables set by the temporary import PHP file to generate an appropriate MySQL query. The variable *typeid* defines the type of the new entry. The function returns the ID of the inserted entry or *false* if the insertion fails.

insert_field

```
int insert_field(int entryid, int fieldtype, string fieldvalue)
```

is called from the temporary import PHP file generated by the *import/biblio.xml* stylesheet. It inserts a new field with entry ID *entryid*, value *fieldvalue*, and type *fieldtype* into the database. The function returns the ID of the inserted field or *false* if the insertion fails.

insert_macros

```
functionType insert_macros(array macros)
```

inserts all macros in the array *macros* into the database. The array *macros* contains the macro name as index and the macro value as value.

insert_person

```
int insert_person(string firstname, string middlename, string lastname, string namesuffix)
```

inserts the person characterized by *firstname*, *middlename*, *lastname*, and *namesuffix* into the database. The function returns the ID of the inserted person or *false* if the insertion fails.

process_insert_entry

```
int process_insert_entry(int typeid)
```

uses the global variable *entry_key* and checks if this entry key already exists in the database. If yes, the function asks the user for advice by calling the function *ask_replace_opt()* and processes the answer by calling *process_replace_submit()*. If no, the function inserts the entry by calling the function *insert_entry()* with parameter *typeid*. The function returns the ID of the inserted entry or *false* if the insertion fails.

process_replace_submit

`void process_replace_submit()`

processes the users decision concerning the replacement of an entry.

restart_import

`void restart_import()`

displays a link to start a new import session.

terminate_import_session

`void terminate_import_session(string message, bool restart)`

deletes all temporary files and variables associated with the current import session and displays the message string *message*. If *restart* is *true*, the function offers a link to start a new import session by calling *restart_import()*.

verify_entry

`int verify_entry(string typename, string typenamespace, string key, string &crossref)`

verifies the entry type name (*typename*) and the entry type namespace (*typenamespace*) of an entry that needs to be inserted and returns the correct entry type ID if the type exists or *false* otherwise. The string *key* is only needed to display a meaningful message to the user if problems occur. The variable *crossref* can contain the key of a cross referenced entry. If this key is correct, the value of *crossref* is changed to the ID of the cross referenced entry.

verify_field_name

`int verify_field_name(string typename, string typenamespace)`

verifies the field type name (*typename*) and the field type namespace (*typenamespace*) of a field that needs to be inserted and returns the correct field type ID if the type exists or *false* otherwise.

verify_group

`bool verify_group(string groupname)`

returns *true* if the specified *groupname* exists in the table *grp* and *false* otherwise.

verify_user

`bool verify_user(string user_name, string user_description)`

returns *true* if the specified *user_name* exists in the table *usr* and *false* otherwise. The string *user_description* is only needed to display a meaningful message to the user if problems occur.

C.5.5 include.php

This PHP file is included in almost any other PHP file. It starts or resumes a PHP session, it creates a connection to the database, and provides a collection of frequently used functions.

clean_person

`void clean_person()`

deletes all persons (from table *person*) that are no longer referenced by a field.

clean_string

`string clean_string(string string)`

replaces single and multiple occurrences of spaces, tabs, line breaks, and carriage returns in *string* by a single white space, and strips leading and trailing whitespace.

clear_namespace

`void clear_namespace(int id)`

deletes the namespace with ID *id* (and all entries and fields with this namespace) from the database.

create_dropdown

`void create_dropdown(string name, string query, string display_first, string value_first, int size, bool selected, bool multiple)`

dynamically creates a drop-down menu for HTML forms from the database. The name of the form field is specified by *name*. The MySQL query string *query* must select two attributes that are used as name and value for the options of the drop-down menu. A default option with name *display_first* and value *value_first* can be specified (NULL for no default option). The size of the drop-down menu is determined by *size*. If *selected* is *true*, the first option of the field will be selected by default. To enable the selection of multiple options, *multiple* must be set to *true*.

delete_all

`void delete_all(string file)`

deletes the file or entire directory specified by the path *file*.

delete_entry_by_id

`void delete_entry_by_id(int id)`

deletes the entry with ID *id* from the database and makes use of the function *clean_person()* to delete all persons that are no longer referenced.

delete_entry_by_key

`void delete_entry_by_key(string key)`

deletes the entry with key *key* from the database and makes use of the function *clean_person()* to delete all persons that are no longer referenced.

delete_import_session_files

`void delete_import_session_files()`

deletes all temporary session files of the current session that are used during data import.

delete_session_files

`void delete_session_files()`

deletes all temporary session files of the current session that are not used during data import.

empty_dir

`void empty_dir(string file)`

deletes all files and subdirectories in the directory specified by the path *file*. This function is used by the function *delete_all()*.

file_upload

`bool file_upload(string relative_file_save_path, string form_intro_message, string form_send_button_label, string upload_ok_message, string upload_failed_message, string mode)`

creates a HTML form for file upload or handles an uploaded file. The exact behavior of the function is determined by the selected *mode*. For more information about the several modes, please consult the documented source code of the function. After a successful file upload, the function will return *true* and *false* otherwise. The uploaded file can be found at *bibroot/relative_file_save_path*. The rest of the parameters influence the appearance of the upload form and are self-explanatory.

gatekeeper

`void gatekeeper(bool user_must_be_admin)`

is called for each page and restricts the access to the page. If *user_must_be_admin* is *true*, only administrators have access to this page. The function uses the session variables `$_SESSION["username"]` and `$_SESSION["user_is_admin"]` to restrict access and generate the navigation bar on top of each page dynamically.

get_from_db

`string get_from_db(string table, string column, string id_value)`

returns the value of attribute *column* from database table *table* where the ID attribute is equal to *id_value*.

get_textvalue

`string get_textvalue(string xmlvalue)`

returns the text value of the XML string *xmlvalue*. This text value is generated by applying the stylesheet *import/special.xsl* to the XML string.

getmicrotime

`float getmicrotime()`

returns the actual PHP system time in seconds. This function is used for performance measurements in the search routine.

my_exit

`void my_exit()`

closes the HTML tags *body* and *html* and stops program execution.

sql_escape

`string sql_escape(string string)`

is just an alias for the built-in PHP function `mysql_escape_string()` and is used to escape special characters in MySQL queries.

user_has_write_rights

`bool user_has_write_rights(string username, string key)`

determines whether the user *username* has write permission for the entry specified by *key*.

user_is_in_group

`bool user_is_in_group(string username, string groupname)`

determines whether user *username* is a member of the group *groupname*.

C.5.6 include_export.php

This file contains important functions for data export. It includes the files *schema/xml_printer.php* and *include_transform.php*.

data_export

```
string data_export(array &entries, string id_or_key)
```

is the main routine to generate the export BibXML document. It calls *export_entries()* as an important subroutine. The array *entries* identifies the entries that need to be exported. The variable *id_or_key* specifies whether the array *entries* contains the IDs or the keys of the entries. The function returns the created export document as an XML string.

display_export_result

```
void display_export_result()
```

calls the export routine *data_export()* to generate the export document as a string. The function then creates the appropriate export files and displays the export result message, which contains links to the export files.

export_entries

```
void export_entries(array &entries, array &prefixes, string &xml)
```

is a subroutine of *data_export()* and used to attach the BibXML representation of all entries defined in *entries* to the XML string *xml*. The array *prefixes* is used to determine the appropriate prefix for each namespace. The function makes use of the subroutine *export_fields()* to export all fields of an entry.

export_fields

```
void export_fields(int entry_id, array &prefixes, string &xml)
```

is used to attach the BibXML representation of all fields from the entry defined by *entry_id* to the XML string *xml*. The array *prefixes* is used to determine the appropriate prefix for each namespace.

C.5.7 include_fast_search.php

This file contains important functions to search the database, present the search results in a HTML table, and resolve macros.

advanced_search

array advanced_search()

takes the various HTTP `$_POST[...]` variables coming from the submitted advanced search form, generates an appropriate MySQL query out of them, queries the database, and returns the IDs of all retrieved entries as an array.

display_results

void display_results(array entries, string search_or_selection)

presents the entries with the IDs specified in the array *entries* as an expandable HTML table, including checkboxes to select each entry for processing. The exact design of the table is determined by the variable *search_or_selection*. Please consult the documented source code of this function for more details.

export_select

void export_select()

generates the form parts needed to export selected entries into the desired export format.

init_simple_search

void init_simple_search()

initializes the various session variables used to store the actual status of the search process and the result table.

search_resolve_macros

string search_resolve_macros(string xml_string, string text_or_xml, bool add_remove_dummy_root, array &used_macro_names)

returns the string *xml_string* with all macros in it being resolved. The variable *text_or_xml* (must be set to “text” or “xml”) determines if the macro reference is replaced by the XML value, or by the text value of the macro. If *add_remove_dummy_root* is *true*, the system will add a dummy root element to the *xml_string* before processing it. All retrieved macros are added to the array *used_macro_names*.

simple_search

array simple_search(string string)

searches all entries that have a key or a field value matching the search expression *string*. The IDs of all retrieved entries are returned in an array.

C.5.8 include_transform.php

The PHP functions included in the *include_transform.php* file are used for transforming exported BibXML documents into user-defined formats like BIBTEX.

prepare_document

```
void prepare_document(string input_document_filename, string output_document_filename)
```

prepares an XML document *input_document_filename* for transformation into user-defined formats. The result is written to *output_document_filename*.

transform

```
bool transform(string source_document_filename, string dest_document_filename, int mode_id)
```

transforms a BibXML document *source_document_filename* into the the export format specified by *mode_id*. The result of the transformation is stored to *dest_document_filename*. *transform()* returns *true* if the transformation succeeded, and *false* otherwise.

translate

```
bool translate(string filename, int mode_id)
```

translates the content of file *filename* according to the translation table that is associated with the export mode with id *mode_id*. *translate()* returns *true* if the translation succeeded, and *false* otherwise.

write_importing_stylesheets

```
bool write_importing_stylesheets(string destination_filename, int mode_id)
```

creates an XSLT stylesheet that imports all user-defined stylesheets associated with export mode *mode_id*, and saves the resulting stylesheet to *destination_filename*. *write_importing_stylesheets()* returns *false* if no stylesheets can be found for the specified mode, and *true* otherwise.

C.5.9 index.php

This file is automatically loaded as index file by the Apache server. It includes the file *include.php* and provides the login screen, validates the login information, and redirects the user to the overview page (*welcome.php*) if the login is correct.

C.5.10 macro_management.php

This file generates the overview over all macros from the database. Each macro can be deleted or macro ownership can be delegated to another user by the current owner or an administrator. New macros can be added through a Web form at the bottom of the generated page.

C.5.11 schema_detail.php

When clicking on a *BibSchema*, *XML Schema*, or *Schematron* link associated with a certain schema on the *Schema Management* page, the PHP script *schema_detail.php* creates a new window displaying the corresponding schema.

C.5.12 schema_management.php

The PHP script *schema_management.php* implements the *Schema Management* page used by the administrator. For this purpose, it uses functions defined in *schema/validation.php* to import, validate, and delete individual schemas.

C.5.13 schema_overview.php

The PHP script *schema_overview.php* implements the *Schema Overview* page, which can be visited by normal users. It displays information about available schemas using the function *schema_overview()* of the *schema/validation.php* file.

C.5.14 schema/validation.php

The PHP functions included in the *schema/validation.php* file are used by the system for validating various kinds of XML documents (most notably, BibXML and BibSchema documents).

check_bib_schema

`void check_bib_schema(bool import)`

creates the input form of the *Schema Management* page, which allows administrators to upload their BibSchema documents. If the *import* argument is set to *true*, *check_bib_schema()* validates BibSchema documents and inserts valid documents into the system. If *import* is *false*, documents are only validated, but not inserted into the system.

check_external_field_refs

`bool check_external_field_refs()`

ensures that external field types referenced by the BibSchema document *session_data/session_id().bs* exist in the database. *check_external_field_refs()* returns *true* if all references are valid, and *false* otherwise.

check_xslt

`bool check_xslt(string stylesheet_filename)`

checks whether the XSLT stylesheet *stylesheet_filename* conforms to the XSLT 1.0 specification or not. If it does, *check_xslt()* returns *true* and otherwise *false*.

contains_forbidden_chars

`bool contains_forbidden_chars(string string)`

checks whether *string* contains characters that are not allowed when defining macro names in BIBTEX (illegal characters: " # % ' () , = { } space). If it does, *check_xslt()* returns *true*, and otherwise *false*.

delete_schema

`void delete_schema()`

fully removes a BibSchema schema with all its corresponding information from the system. The document to be deleted is chosen on the *Schema Management* page and an identifier for this document is passed to the *delete_schema()* function using the HTTP protocol.

get_external_refs

`mixed get_external_refs(object dom_instance)`

returns all external cross references of a BibXML document *dom_instance* (an object of class "Dom document") as an array. If cross references are external and internal (i.e., they cross reference an entry in the database as well as an entry in the imported BibXML document), *false* is returned.

get_ids_for_schemas

mixed `get_ids_for_schemas()`

returns an array of schema identifiers (an identifier for each BibSchema document in the system). If there are no BibSchema documents in the system, the value *-1* is returned in place of the array.

get_instance_namespaces

mixed `get_instance_namespaces(object dom_instance)`

returns an array of all namespaces that are used in the BibXML document *dom_instance* (an object of class “Dom document”). If the BibXML document makes use of a namespace that is not known to the management system (only the base namespace `http://bibtexml.org/base` and the target namespaces of the imported BibSchema documents are known to the system), `get_instance_namespaces()` returns *-1*.

get_new_macros

mixed `get_new_macros(object dom_instance, bool prefer_db_macros)`

returns an array that contains an entry for each macro definition of the BibXML document *dom_instance* (an object of class “Dom document”). The names of the new macros are used as the index of the array, and the values of the macros as the array values. If the macro names do not conform to the L^AT_EX specifications for macro names (i.e., spaces and the following characters are not allowed: `” # % ’ () , = { }`), *false* is returned. If *prefer_db_macros* is *true*, *dom_instance* is allowed to declare macro names that are already declared in the system. If *prefer_db_macros* is *false*, the redeclaration of macro names is not allowed and results in the return value *false*.

get_root_element_attribute

string `get_root_element_attribute(string document_filename, string attribute)`

returns the value of the *attribute* attribute of the root element of the XML document *document_filename*. If this attribute does not exist, the empty string is returned.

insert_bib_schema

bool `insert_bib_schema(string namespace)`

inserts the target namespace *namespace* of a BibSchema schema into the database and saves the BibSchema schema with this target namespace and its transformed XML Schema and Schematron schemas in a corresponding subdirectory of *schema*. `insert_bib_schema()` returns *true* if the insertion succeeds, and *false* otherwise.

insert_types_into_db

`bool insert_types_into_db(int id)`

inserts entry and field types defined by the BibSchema schema with identifier *id* into the database. *insert_types_into_db()* returns *true* if the insertion succeeds, and *false* otherwise.

is_external_field

`bool is_external_field(string ref, array uri)`

checks whether the specified field type exists in the database or not. *ref* is the prefixed name of the field type and the array *uri* has namespace prefixes as its index, and namespace URIs as its values. *is_external_field()* returns *true* if the field type exists in the database, and *false* otherwise.

is_new_namespace

`bool is_new_namespace(string bib_schema_filename)`

checks whether the target namespace of the BibSchema document *bib_schema_filename* is already taken by another schema. *is_new_namespace()* returns *true* if the namespace is still available, and *false* otherwise.

is_wellformed

`bool is_wellformed(string document_filename)`

ensures that the XML document *document_filename* is well-formed. If it is, *is_wellformed()* returns *true*, otherwise *false*.

output_saxon_error

`void output_saxon_error(array error)`

formats and outputs the error message specified by *error*, which is an array filled with every line of output from the Saxon processor.

output_xerces_error

`void output_xerces_error(array error, string code, string allowed_msg)`

formats and outputs the error message specified by *error*, which is an array filled with every line of output from the Xerces processor. *allowed_msg* defines a line of output from the Xerces processor that should be ignored in the output. *code* is a string that is appended at the front of every line of output from the Xerces processor.

remove_dir

`void remove_dir(int id)`

removes all directories that are associated with the BibSchema document with identifier *id*.

remove_ns_and_dir

`void remove_ns_and_dir(int id)`

removes all database entries and directories that are associated with the BibSchema document with identifier *id*.

resolve_macros

`bool resolve_macros(string document_filename, bool &has_entries)`

resolves the macro references of the BibXML document *document_filename* and sets *has_entries* to *true*, if the BibXML document contains at least one entry. *resolve_macros()* returns *false* if it is not able to resolve all macro references, and *true* otherwise.

schema_overview

`void schema_overview(string type)`

If *type* is set to *administrator*, *schema_overview()* creates a table for manipulating imported BibSchema schemas on the *Schema Management* page. If *type* is set to *administrator*, it creates a table showing information about imported BibSchema schemas on the *Schema Overview* page.

schematron_check_with_saxon

`bool schematron_check_with_saxon(string document_filename, string schematron_filename)`

validates an XML document *document_filename* against a Schematron schema *schematron_filename*. *schematron_check_with_saxon()* returns *true* if *document_filename* is valid, and *false* otherwise.

simple_xerces_validation

`bool simple_xerces_validation(string document_filename, string xml_schema_filename)`

validates an XML document *document_filename* against an XML Schema *xml_schema_filename*. *simple_xerces_validation()* returns *true* if *document_filename* is valid, and *false* otherwise.

validate_bib_schema

`bool validate_bib_schema(string bib_schema_filename)`

validates a BibSchema schema *bib_schema_filename* against the specifications for the BibSchema schema language. *validate_bib_schema()* returns *true* if *bib_schema_filename* is valid, and *false* otherwise.

validate_denorm_instance

`bool validate_denorm_instance(string instance_filename)`

validates a denormalized BibXML document *instance_filename* against the predefined and user-defined schemas of the system. *validate_denorm_instance()* returns *true* if *instance_filename* is valid, and *false* otherwise.

validate_instance

`bool validate_instance(string instance_filename, bool prefer_db_macros)`

validates a BibXML document *instance_filename* (compared with *validate_denorm_instance()*, *instance_filename* does not have to be denormalized) against the predefined and user-defined schemas of the system. *validate_instance()* returns *true* if *instance_filename* is valid, and *false* otherwise. If *prefer_db_macros* is *true*, *instance_filename* is allowed to declare macro names that are already declared in the system. If *prefer_db_macros* is *false*, the redeclaration of macro names is not allowed and results in the return value *false*.

xml_schema_check

`bool xml_schema_check(string document_filename)`

accepts only BibSchema, XML Schema, or Schematron schemas as parameter *document_filename*, and validates *document_filename* against the appropriate built-in schemas. *xml_schema_check()* returns *true* if *document_filename* is valid, and *false* otherwise.

C.5.15 schema/xml_printer.php

The PHP functions included in the *schema/xml_printer.php* file are used for displaying XML documents in a browser window.

schema_format

`string schema_format(string xml)`

reformats an XML string *xml* and returns a structured XML string that can be displayed in a browser window.

xml_format

`string xml_format(string xml)`

translates all characters of the XML string *xml* that have HTML character entity equivalents into these entities, and returns the resulting XML string.

C.5.16 selection.php

This file generates the selection, which can be used like a shopping cart to collect multiple search results for later processing or export. It uses the function *display_results()* from the file *include_fast_search.php* to present the collected entries in an expandable HTML table. The file includes *include_export.php* to offer all the export functions already known from the search pages.

init_select_session

```
void init_select_session()
```

initializes all session variables used to remember the status of the current selection (entries in the selection, expanded or collapsed).

C.5.17 user.php

This file generates the user management page. It uses the *gatekeeper()* function from the included file *include.php* to grant access to administrators only. The user management page provides an overview over all registered system users and groups, and is dynamically created from the database. Administrators can add, remove, and manage users and groups.

C.5.18 welcome.php

This PHP file generates the overview page for users and administrators. The file only includes *include.php* to restrict access to the system and generate the navigation bar.

D CD ROM

This CD ROM contains all necessary files to install and configure the bibliography management system. The contents of the CD ROM are described in Table 27. Instructions for the system setup are given in Section C.1.

Directory	Contents
<i>Configuration</i>	Example configuration files for Apache, MySQL, and PHP
<i>Presentation</i>	Powerpoint presentation of the diploma thesis
<i>Report</i>	This report in the pdf format
<i>Software</i>	All software necessary to install the system on a Windows XP platform
<i>SystemFiles</i>	The files of the bibliography management system

Table 27: CD ROM Contents