Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

Diploma Thesis DA-2001.02

BIBTEXML

Brenno Lurati, Luca Previtali

Tutors: Dr. Erik Wilde, Dr. Bernhard Plattner
Supervisor: Dr. Bernhard Plattner

Winter Term 2000/2001

Abstract

BibTeXML is an XML representation of BibTeX data. It can be used to represent bibliographic data in XML. The advantage of BibTeXML over BibTeX's native syntax is that it can be easily managed using standard XML tools (in particular, XSLT style sheets), while native BibTeX data can only be manipulated using specialized tools.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

BibTeX is probably the most widely used format for bibliographies. There are many online BibTeX collections and this format is not used by LaTeX only: For example the publishing program *Adobe FrameMaker*[1] supports it.

The major advantage using BibTeX is the possibility to store bibliographic data separated from the text referencing to. This fact allows an easy data reuse and offers, for example, the possibility to apply different stylistic templates.

One drawback of BibTeX is that errors in the bibliography file are hard to locate. For example in the case of an unmatched quotation marks around a value string, the current version of the program raises an error at the end of the scan when tables overflow after several character of input. Another problem is that there is no tools for editing BibTeX bibliography: such files are usually manually typed. This fact introduces formatting variation and inconsistencies. For example the `author` field format is very abstruse and few users know the right typesetting of this information.

The goal of this master thesis is the design and the implementation of a system, that allows the use of BibTeX bibliographic database in a more effective and comfortable way.

This project focuses on the BibTeXML representation of bibliographic entries which is an XML environment for structuring the BibTeX data. XML provides an efficient way to organize data using an easy to edit and readable format. With XSLT it is possible to easily convert XML data into various formats. The basic idea of BibTeXML is that the whole bibliography should be managed using XML. BibTeX files are no longer edited: they are generated for BibTeX use only. This fact allows the users to take advantage of the full BibTeXML capabilities as the powerful macro environment or the

[1]`http://www.adobe.com/products/framemaker/main.html`

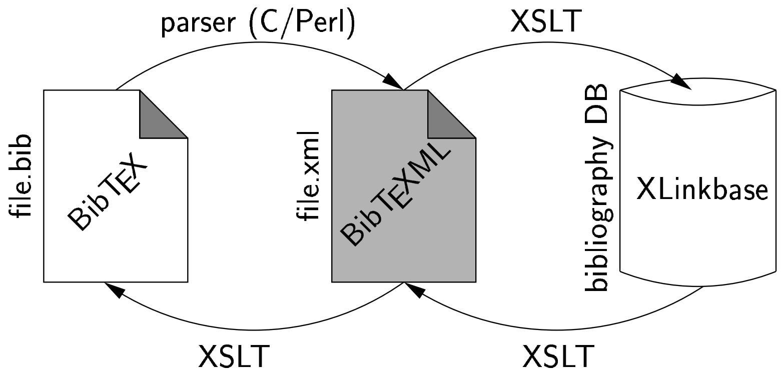

Figure 1.1: BibTeXML Architecture

possibility to deal with non–standard BibTeX fields.

To enable a reuse of existing BibTeX collections, we need a conversion tool for transforming BibTeX bibliography into BibTeXML format: It is not suitable that all existing bibliographies are manually rewritten in the new format or simply ignored.

The main task of this conversion is to parse the original BibTeX format into the new BibTeXML structure. Unfortunately the majority of the existing bibliography collections contain syntax errors which does not allow a clean parsing process. Thus, first of all, the input is checked and, if some errors are found, must be corrected. After the input BibTeX data have passed the syntax check they are parsed into the BibTeXML format. This conversion is done in several different steps which are not visible by the user. The output of the whole process is a BibTeXML file. As explained in Section 4.2 all LaTeX special symbols are converted either in Unicode or in a special XML element. This translation is done using a *conversion table* which contains the list of recognized symbols which the Unicode/XML correspondence. If a symbols is not stored in the *conversion table* it is not recognized and thus it is not translated. If this happens, a warning message is generated and the user can manually convert the symbol editing the XML file.

As already explained the whole parsing process seems a little bit complex. The absence of a well defined BibTeX grammar, the huge amount of special symbols allowed in TeX (it is also possible to define own symbols) do not simplify the conversion task and does not allow a full automated parsing process. Our goal is to reach the maximum degree of translation with the parser and to generates a series of errors and warnings messages which should help the user obtaining a consistent and correct BibTeXML bibliography. The conversion process is not “harmless”, but we believe that the time spent correcting and porting the bibliography data into the new format is an useful investment: Since the BibTeXML format is obtained the user has a consistent bibliography and benefits of all XML advantages.

Once the BibTeXML representation is obtained, it is also possible to store the BibTeXML entries in an online database which database provides complex queries and data navigation helping the user to fetch the required references. The final product of the project is the website `http://bibtexml.org` which offers all the services described before.

Chapter 2

Environment

This Chapter explains shortly the basic technologies used during the development of the thesis.

2.1 LaTeX

TeX [1] is a program created by Donald E. Knuth in 1977. At that time there was anything suitable for high quality text publishing. So Knuth started writing the TeX typesetting engine, which was an attempt to enhance the typographical quality of the scientific books and articles. TeX is today one of the best program for scientific reports and technical documents. TeX is extremely stable and runs on almost every kind of computer. The current version of TeX is 3.14159: version number is converging to π.

LaTeX [2, 3] is a macro package written by Leslie Lamport. This package uses the TeX as its formatting engine. LaTeX is an user support to create high typographical quality text, using predefined, professional layout.

Most people today use program called WYSIWYG ("What you see is what you get."). Classical examples of such programs are *MS Word* and *Corel WordPerfect*. With these applications the author specifies the appearance of the document at the same time as he is typing the text. The advantage is that you can see immediately the final result.

LaTeX has many advantages versus WYSIWYG programs. First of all documents generated with WYSIWYG programs are aesthetical pleasing but usually have a poor and inconsistent structure. LaTeX forces the user to write structured and well-organized documents. The typesetting of mathematical expression is supported in a convenient way. Complex structure such as table, footnotes, references and bibliographies can be easily generated.

2.2 BibTeX

BibTeX [4, 2] is a program and file format designed by Oren Patashnik and Leslie Lamport in 1985 for the LaTeX document preparation system. The format is entirely character based, so it can be used by any program (although the standard character set for accents is TeX). It is field (tag) based and the BibTeX program will ignore unknown fields, so it is expandable. It is probably the most common format for bibliographies on the Internet.

BibTeX reads the top-level auxiliary `.aux` file, that was output during the running of LaTeX, and creates a bibliography `.bbl` file that will be incorporated into the document on subsequent runs of LaTeX (See Figure 2.1).

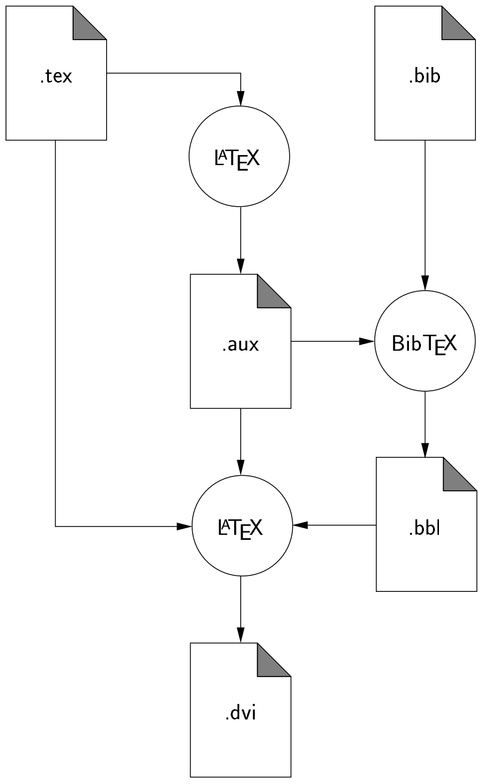

Figure 2.1: LaTeX–BibTeX Interaction

BibTeX automatically constructs a bibliography for a LaTeX document by searching one or more databases. To this end, the LaTeX file must contain the command

```
\bibliography{database1,database2,...}
```

at the point in the text where the bibliography is to appear. The standard extension for database file is `.bib`. To make a reference to a publication in the database, we place in the text the command

```
\cite{key}
```

where *key* is the database identifiers for that publication.

The entries in a bibliographic database are of the form

```
@Book{tex,
      author =      "Donald E. Knuth",
      title =       "The {\TeX}book",
      edition =     "third"
      publisher =   "Addison--Wesley",
      year =        "1986",
      address =     "Reading, Massachusetts",
}
```

The first word, prefixed with '`@`', determines the *entry type* (in the previous example it is `Book`) which is followed by the reference information for that entry enclosed in curly braces `{ }`. The very first entry is the *key* reference by which it is referred to in the `\cite` command. In the above example this is `tex`. The actual reference information is then entered in various *fields*, separated by commas. Each *field* consists of a *field name*, an `=` sign, and a *field text*. The *field names* shown in the example are `author`, `title`, `edition`, `publisher`, `year`, `address`. The *field text* must be enclosed either in curly braces or in double quotation marks. For each *entry type*, certain fields are required, others are optional, and the rest are ignored.

The general syntax for entries in the bibliographic database has the following form:

```
@entrytype{key,
      fieldname =  "fieldtext",
      ....
      fieldname =  "fieldtext",
}
```

The names of the *entry types* as well as the *field names* may be written in capitals or lower case letters, or in a combination of both. Thus `@BOOK`, `@book`, `@bOOk`, and `@BoOk` are all possible variations. The outermost pair of braces for the entire entry may be either curly braces `{ }` or parentheses `( )`. In this case, the general syntax is

```
@entrytype(key, ... ...)
```

Standard entry types

The following list describes all BibTeX standard *entry types*.

@article An article from a journal or magazine.

@book A book with an explicit publisher.

@booklet A work that is printed and bound, but without a named publisher or sponsoring institution.

@conference The same as "inproceedings".

@inbook A part of a book, which may be a chapter (or section or whatever) and/or a range of pages.

@incollection A part of a book having its own title.

@inproceedings An article in a conference proceedings.

@manual Technical documentation.

@mastersthesis A Master's thesis.

@misc Use this type when nothing else fits.

@phdthesis A PhD thesis.

@proceedings The proceedings of a conference.

@techreport A report published by a school or other institution, usually numbered within a series.

@unpublished A document having an author and title, but not formally published.

Other entry types

Using these *entry types* is not recommended, but they might occur in some bibliographies.

@collection A collection of works. The same as proceedings

@patent A patent.

Standard fields names

The following list describes all standard BibTeX *fields names.*

address Usually the address of the publisher or other type of institution.

annote An annotation. It is not used by the standard bibliography styles, but may be used by others that produce an annotated bibliography.

author The name(s) of the author(s).

booktitle Title of a book, part of which is being cited. For book entries, the title field should be uded instead.

chapter A chapter (or section or whatever) number.

crossref The database key of the entry being cross referenced. Any fields that are missing from the current record are inherited from the field being cross referenced.

edition The edition of a book (for example, "Second"). This should be an ordinal, and should have the first letter capitalized, as shown here; the standard styles convert to lower case when necessary.

editor Name(s) of editor(s). If there is also an author field, then the editor field gives the editor of the book or collection in which the reference appears.

howpublished How something strange has been published. The first word should be capitalized.

institution The sponsoring institution of a technical report.

journal A journal name. Abbreviations are provided for many journals.

key Used for alphabetizing, cross referencing, and creating a label when the "author" information is missing. This field should not be confused with the key that appears in the cite command and at the beginning of the database entry.

month The month in which the work was published or, for an unpublished work, in which it was written.

note Any additional information that can help the reader. The first word should be capitalized.

number The number of a journal, magazine, technical report, or of a work in a series. An issue of a journal or magazine is usually identified by its volume and number; the organization that issues a technical report usually gives it a number; and sometimes books are given numbers in a named series.

organization The organization that sponsors a conference or that publishes a manual.

pages One or more page numbers or range of numbers, such as 42–111 or 7,41,73–97 or 43+ (the '+' in this last example indicates pages following that don't form a simple range). To make it easier to maintain Scribe-compatible databases, the standard styles convert a single dash (as in 7-33) to the double dash used in TEX to denote number ranges (as in 7–33).

publisher The publisher's name.

school The name of the school where a thesis was written.

series The name of a series or set of books. When citing an entire book, the the title field gives its title and an optional series field gives the name of a series or multi-volume set in which the book is published.

title The work's title, typed as explained in the LaTeX book.

type The type of a technical report (for example, "Research Note").

volume The volume of a journal or multi-volume book.

year The year of publication or, for an unpublished work, the year it was written. Generally it should consist of four numerals, such as 1984, although the standard styles can handle any year whose last four non-punctuation characters are numerals, such as '(about 1984)'.

Other field names

BibTEX is extremely popular, and many people have used it to store various information. Here is a list of some of the more common non–standard *field names*:

affiliation The authors affiliation.

abstract An abstract of the work.

contents A Table of Contents

copyright Copyright information.

ISBN The International Standard Book Number.

ISSN The International Standard Serial Number. Used to identify a journal.

keywords Key words used for searching or possibly for annotation.

language The language the document is in.

location A location associated with the entry, such as the city in which a conference took place.

LCCN The Library of Congress Call Number. I've also seen this as lib-congress.

mrnumber The Mathematical Reviews number.

price The price of the document.

size The physical dimensions of a work.

URL The WWW Universal Resource Locator that points to the item being referenced. This often is used for technical reports to point to the ftp site where the postscript source of the report is located.

Special features

The `@STRING` command is used to define abbreviations for use by BibTeX. The command

```
@string{jgg1 = "Journal of Gnats and Gnus, Series~1"}
```

defines '`jgg1`' to be the abbreviation for the string `"Journal of Gnats and Gnus, Series~1"`. Any reference outside of quotes or braces to '`jgg1`' will be filled in with the full string.

The `@PREAMBLE` command is used to define formatter code that will be output directly to the `.bbl` file produced by the BibTeX program. This usually consists of LaTeX macros. It is unclear what one should do with the fields when converting to a format that does not use TeX.

The `@COMMENT` command lets you put any text inside it. It isn't really necessary, since BibTeX will ignore any text that isn't inside an entry. However, you can not have an '`@`' character outside of an item.

This short introduction illustrates some of the particularities of the BibTeX format. It is obvious that the different possibilities which are allowed for specifying the same information, do not facilitate the manipulation of the data.

The following example shows a typical BibTeX entry containing non–standard fields, special TeX symbols, different *field text* delimiters and abbreviations:

```
@Book{Goossens,
  author =       "Michel Goossens and Frank Mittelbach and Alexander
                 Samarin",
  title =        "The {\LaTeX} Companion",
  publisher =    "Ad{\-d}i{\-s}on--Wes{\-l}ey",
  address =      reading,
  edition =      {Second},
  pages =        "xxi + 530",
  year =         "1994",
  ISBN =         {0-201-54199-8},
  LCCN =         "Z253.4.L38 G66 1994",
  bibdate =      "Wed Nov 16 12:41:07 1994",
  price =        "US\$34.25",
  series =       "Tools and Techniques for Computer Typesetting",
  acknowledgement = ack-nhfb,
}
```

2.3 XML

XML (Extensible Markup Language) is a markup language developed by the W3C (World Wide Web Consortium) for documents containing structured information. Address books, spreadsheets, bibliographies, etc. are all examples of structured data. XML is a set of rules, guidelines and conventions for designing text formats for such data. Put succinctly, XML is a meta language that allows you to create and format your own document markups. With HTML existing markup is static. XML, on the other hand, allows you to create your own markup tags and configure each to your liking.

But one of the most important aspects of XML is that there is a growing set of optional modules that provide sets of tags and attributes, or guidelines for specific tasks. The most important are listed here:

XLink [5] describes a standard way to add hyperlinks to an XML file. These links can be multidirectional.

XPointer [6] a syntax for pointing to parts of an XML document. (An XPointer is a bit like a URL, but instead of pointing to documents on the Web, it points to pieces of data inside an XML file.).

XSL [7] an advanced language for expressing style sheets. It is based on XSLT, a transformation language that is often useful outside XSL as well, for rearranging, adding or deleting tags and attributes. By using XSL , you ensure that your documents are formatted the same no matter which application or platform they appear on.

XML Namespaces [8] a specification that describes how you can associate a URL with every single tag and attribute in an XML document. XML Namespaces are used to avoid name conflicts inside a XML document.

XML Schemas [9, 10, 11] a set of constraints that control how document may be structured. They allow you to validate documents and provide documentation for your vocabularies.

Chapter 3

BibTeXML Format

3.1 XML Schema

Usually the structure of an XML document is done using DTD (Document Type Definition). At the beginning of the project we study the advantage to define BibTeXML using XML Schema [10]. XML Schema is more powerful than DTD. Although DTDs have served developers well for many years as a mechanism for describing structured information, DTDs have severe restrictions compared to XML Schema. DTD elements can contains only one of the following options:

- Text string.
- Child elements.
- Text string with child element.

DTDs do not have an XML syntax and do not offer support for data types and namespaces. XML Schema allows the concept of namespaces to distinguish the definition and support several datatypes [11] (integer, string, url, etc...) with constrains. For these reasons we defined the BibTeXML format using XML Schema. The BibTeXML structure is described by two XML Schemas. The first Schema defines elements and attributes which correspond to standard entry types of BibTeX. The second Schema defines not standard entries of BibTeX and additionally elements that does not exist in BibTeX. This solution allows a full compatibility with the original format and makes possible further developments. The two kinds of elements belong to two different namespaces: *bibtexml* (`http://bibtexml.org/Schemas/STBIBTEXML`) and *nsbibtexml* (`http://bibtexml.org/Schemas/NSTBIBTEXML`)(see Appendix A).

3.2 BibTeXML Format Description

The BibTeXML format has almost the same structure of BibTeX. This has been done to help traditional BibTeX users. Users have not to learn another format but they use the same standard taking advantage of XML features.

The root element of a BibTeXML document is `<bibliography>`. This root element has the following optional attributes which describe the document and the BibTeXML user.

- `version`: version of the BibTeXML document.
- `author`: the name of the person who wrote the BibTeXML document.
- `created`: the date when the document was created.
- `lastmodified`: the date of the last modification of the current document.

The `<bibliography>` element has four different child elements. Three of them, `<macro>`, `<preamble>`, `<comment>`, have exactly the same meaning of the original BibTeX commands.

BibTeX standard entry types (e.g. `@article`, `@book`, etc.) are transformed in a general element called `<bibitem>` which contains two attributes:

- `type`: kind of the document. It can have the following values: article, book, conference, inbook, etc. (see Section 2.2).
- `label`: identifier of the `<bibitem>` element. (The attribute label is of type ID to keep the uniqueness of the bibitem)

The element `<bibitem>` has 29 children which are exactly the 23 standard fields of BibTeX (e.g `<author>`, `<publisher>`, `<title>`, etc.) with 6 additional not standard fields: `<URL>`, `<abstract>`, `<ISBN>`, `<ISSN>`, `<contents>`, `<notstandard>`. `<notstandard>` is used when a user want to define its own field. To make this he has just to write:

```
<nstandard name="myfieldname">myfield content</nstandard>
```

To keep a complete compatibility with the original format, we defined two special element: `<tex>` and `<abbrev>`. In Chapter 4.2 is explained the meaning of the special element `<tex>`; `<abbrev>` "simulates" the BibTeX macro mechanism. In BibTeX it is possible to define macro with the special command `@string`.

```
@string{dret = "Erik Wilde"}
...
author = dret,
```

In the same way in BibTeXML we have the following macro environment:

```
<macro alias="dret">Erik Wilde</macro>
```

The macro can be referenced inside an element with the special element `<abbrev>`:

```
<editor>
  <abbrev alias="dret"/>
</editor>
```

Finally the element `<author>` has four optional children:

- `<firstname>`: the first name of the author.
- `<middlename>`: the middle name of the author.
- `<lastname>`: the lastname of the author.
- `<suffix>`: Name suffix like: Jr., III, Sr., etc.

Chapter 4

BibTeX to BibTeXML Conversion

The main task of the BibTeX to BibTeXML conversion is to obtain an XML representation of a BibTeX file. A secondary job is to check the correctness of the input data and to generate warning or error messages if necessary. Since an XML file is easily transformed into other XML files with different structure, we decide to create an XML intermediate representation (see Figure 4.1) which is very close to the original format. The introduction of an intermediate representation makes the implementation of the front end much easier. The back end is developed using XSLT with a simple XSL stylesheet (see Section 4.3).

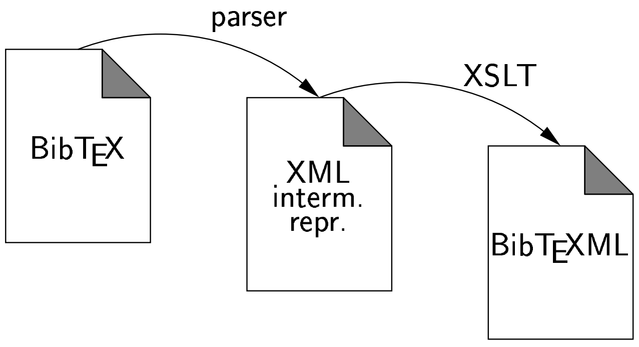

Figure 4.1: BibTeX to BibTeXML Conversion Steps

4.1 Intermediate Representation

As already explained, we introduced an intermediate representation to simplify the BIBTEX to BIBTEXML conversion. This format is very close to the original BIBTEX format but it is already structured using XML. The C parser scans the input files and transforms the BIBTEX structured data into an XML file. For example the following entry:

```
@Book{kr,
  author =       "B. W. Kernighan and D. M. Ritchie",
  title =        "The {C} Programming Language",
  publisher =    "Prentice-Hall",
  address =      "Upper Saddle River, NJ 07458, USA",
  edition =      "Second",
  year =         "1988",
}
```

is transformed into:

```
<Book>
  <label>kr</label>
  <author>B. W. Kernighan and D. M. Ritchie</author>
  <title>The {C} Programming Language</title>
  <publisher>Prentice-Hall</publisher>
  <address>Upper Saddle River, NJ 07458, USA</address>
  <edition>Second</edition>
  <year>1988</year>
</Book>
```

This XML file is really very close to the original one: the basic structure

```
@entrytype{key,
      fieldname =  "fieldtext",
      ....
      fieldname =  "fieldtext",
}
```

is transformed into

```
<entrytype>
    <label>kr</label>
    <fieldname>fieldtext</fieldname>
    ....
    <fieldname>fieldtext</fieldname>
</entrytype>
```

The macros and the references which are present into the source data, are also transformed into a XML structure:

```
...
@String{ieeecsp = "IEEE Computer Society Press"}
...
publisher =    ieeecsp
...
```

becomes

```
...
<String>
  <alias>ieeecsp</alias>
  <macrotext>IEEE Computer Society Press</macrotext>
</String>
...
<publisher><ref>ieeecsp</ref></publisher>
...
```

The parser does not make any semantic check of the input data: all *entry types* and *field names* are transformed into the XML intermediate representation, thus it is impossible to define a DTD for this format. The semantic check is performed by the XSLT which translates the bibliography into the definitive BibTeXML file.

4.2 BibTeX to Intermediate Representation Conversion

We evaluated different possibilities for the parser implementation.

The first solution we investigated, was to use BibTeX to generate the desired data. BibTeX uses a *bibliography style file* (`.bst`) as input to determine the format of the bibliography [12] and one option was to implement a special `.bst` file which describes the wanted XML bibliography format (see Figure 4.2). This solution has two major disadvantages. First of all the documentation about the *bibliography style file* is not very exhaustive: it is very difficult to find a full and clear specification of this format. The second problem is that a solution using BibTeX does not solve all drawbacks of BibTeX itself: for example no syntax check is made to proof the input bibliography data because BibTeX assumes that its input is prepared correctly.

The second solution we investigated was to develop a stand-alone parser transforming the BibTeX input into the XML intermediate representation.

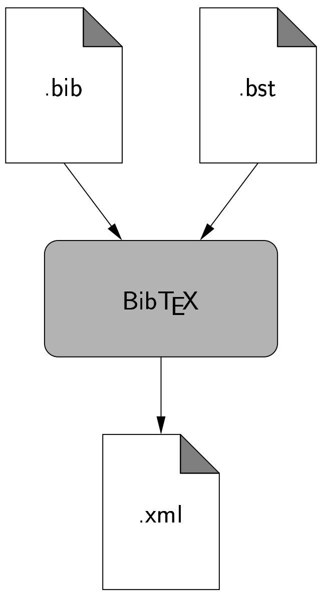

Figure 4.2: BibTEX as parser generating XML

The translation of structured languages is usually divided into three main steps. First of all the compiler performs a *lexical analysis*, grouping consecutive characters into units (called tokens) which are "words" meaningful in the given language. The *parser* processes these token sequences to verify that they are in an order permitted by the language grammar (the set of rules which defines the language). At the end the *code is generated* interpreting the the token stream (which is grammar conformant) to perform an intended task. For example, BibTEX transforms BibTEX data according to rules in a user-specified style file into formatted bibliographic data suitable for the LATEX typesetting system. BibTEX includes internal implementations of lexical analysis and parsing but these are not available to the user. Writing lexical analyzers by hand can be a tedious process, therefore software tools have been developed to ease this task. These tools are called lexical analyzer generators: they take a specially-formatted specification file containing the description of the grammar defining the different tokens and generate the scanner source code. Similarly a parser generator builds the parser source file from the language grammar.

We first evaluated the possibility to implement the parser in Java [13] to take advantage of the portability offered by the Java environment. Despite this benefit a serious handicap is the absence of good and mature scanner and parser generators for the Java language. We made a first essay using *JLex* and *CUP*. *JLex* [14] is a lexical analyzer generator for Java which is based upon the *lex* model. *CUP* [15] is a Java based system for generating parsers from simple specifications; it serves the same role as the widely used program *yacc*. *CUP* is written in Java, uses specifications including embedded Java code, and produces parsers which are implemented in Java. Unfortunately *CUP* is in an early development stage and lacks a lot of important features and is therefore unsuitable for the task.

After the evaluation of the already explained approaches, we switched to the solution which we chose, that involves C [16] as programming language and *lex* and *yacc*.

lex

lex [17] reads the given input file for the description of the scanner to generate. The specification of the regular language defining the scanner is in the form of pairs of regular expressions and C code, called rules. *lex* generates a C source file as output (`lex.yy.c`) which defines a routine `yylex()`. When the executable is run, it analyzes its input for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

yacc

yacc [17] is an LALR(1)[1] [18] parser generator. *yacc* reads the grammar specification from an input file and generates a bottom-up parser for it. The parsers consist of a set of parsing tables and a driver routine written in the C programming language. *yacc* normally writes the parse tables and the driver routine to the file `y.tab.c`.

Error detection and correction

A parser development is based on the grammar specifying the syntax of the data to manipulate. Unfortunately there is no well-defined grammar for BibTeX. Beebe proposed a grammar for BibTeX [19] which is the result of different studies and practical tests. He also developed several tools which are widely used for prettyprinting and syntax checking bibliography files. For example every bibliography in the Collection of Computer Science Bibliography (`http://liinwww.ira.uka.de/bibliography/`) has been converted to the BibTeX format in a standardized layout with the aid of the *bibclean* tool [20] developed by Beebe and any entry that does not pass the syntax

[1]Lookahead left recursive; the parsing technique that *yacc* uses. The (1) denotes that the lookahead is limited to a single token. A grammar which can be parsed by a LALR(1) parser is *context free* grammar

checker is silently omitted during the conversion process. Evaluating Beebe's work we come to the conclusion that the proposed BibTeX grammar is a robust starting point for our project.

Unlike BibTeX, our parser obeys to a well-defined grammar and available BibTeX bibliographies usually contain errors which hinder the parsing process. To solve this problem, before the transformation, the input data are checked for syntax mistake using *bibclean* and *bibparse*. Both programs generate errors or warning if a problem is found. These errors cannot be automatically fixed: the user should manually correct the input file, according to the results of the syntax test. The correction process of the input data is schematized in Figure 4.3.

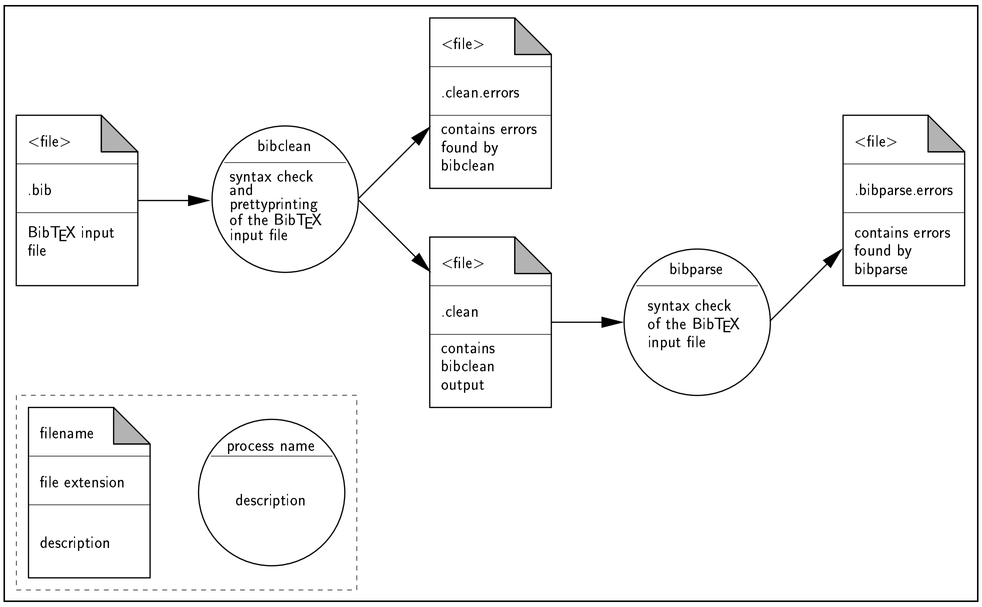

Figure 4.3: Error Correction Process

bibclean

bibclean [20] is a program which prettyprints and checks the syntax of BibTeX bibliography data files. *bibclean* prettyprints input BibTeX files to stdout, and checks the brace balance and the bibliography entry syntax as well. It detects problems in BibTeX files that sometimes confuse even BibTeX itself, and normalizes the appearance of BibTeX entries.

For example each BibTeX item is formatted into a consistent structure with one `field = "value"` pair per line, hyphen sequences in page numbers are converted to en-dashes ('–'), month values are converted to standard

BibTeX string abbreviations (`jan`, `feb`, . . . '), personal names in author and editor field values are normalized to the form "P. D. Q. Bach", from "P.D.Q. Bach" and "Bach, P.D.Q." and so on. The standardized output of bibclean facilitates the later application of the *bibtexmlparse* (see Section 4.2) parser and the *Perl* converter (see Section 4.2). *bibclean* generates also warnings if a bad ISBN or ISSN number is found

When bibclean detects an error, it issues an error message to both stderr and stdout. Error messages begin with a distinctive pair of queries "`??`" followed by the input file name and line number. Warning messages identify possible problems, and are therefore sent only to stderr, and not to stdout, so they never appear in the output file. They are identified by a distinctive pair of percents "%%". If error messages are ignored, and left in the output bibliography file, they will precipitate an error when the bibliography is next processed with BibTeX.

After issuing an error message, *bibclean* resynchronizes its input by copying it verbatim to stdout until a new bibliography entry is recognized on a line which first non-blank character is an at-sign ('`@`'). This ensures that nothing is lost from the input file(s), allowing corrections to be made in either the input or the output files.

bibparse

bibparse [21] is a program which checks a BibTeX files, verifying their conformance to a proposed grammar for BibTeX [19]. The only output normally produced by *bibparse* is on the standard error unit, stderr, only if grammatical errors are detected. Silent execution means a successful parse. This program is developed using *lex* and *yacc.*

The parser

bibxmlparse is the parsing program which we developed for transforming the input BibTeX data (which is assumed to be correct) into the intermediate representation (see Figure 4.4). This tool is based on the top of the *bibparse* version 1.04. We modified the *lex* and the *yacc* files in such a way that the desired output is generated.

The Lexer The lexer job is to feed tokens to the parser. The parser is the higher level routine, and calls the lexer `yylex()` whenever it needs a token from the input. As soon as the lexer finds a token of interest to the parser, it returns to the parser, returning the token code as the value. *Yacc* defines the token names in the parser as C preprocessor names in `y.tab.h` so the lexer can use them.

`bibxmllex.l` is the new *lex* file. We added the C code needed to store the value of the single recognized tokens in the `yylval` variable. For example if a token `TOKEN_KEY` is found we store the value with the following C expression:

```
...
case TOKEN_KEY:
  yylval.sval=strdup(yytext);
...
```

The Parser `bibxmlparse.y` is the new *yacc* file. We modified the *definition section*[2] adding the value type of the single tokens. For example the definition of the token `TOKEN_KEY` is:

```
%token <sval> TOKEN_KEY 10
```

This code specifies that the token `TOKEN_KEY` is of type `sval` which is defined in the same file as:

```
%union
{
  char *sval;
  int ival;
}
```

We adapted also the *rules section*[3] so that the desired intermediate representation is printed to the standard output. For example the following code

```
...
 printf("<label>%s</label>\n",$6);
...
```

prints the `<label>` element with its content.

Conversion of LaTeX special characters

In BibTeX entries it is possible to specify characters other than alphabetical ones, adopting the TeX character syntax. Each special character begins with

[2]The definition section of a *yacc* file specification includes declarations of the token used in the grammar and the type of values used in the parser stack.

[3]The rules section consists of a list of grammar rules. In this section is also possible to specify C code to perform rule's action.

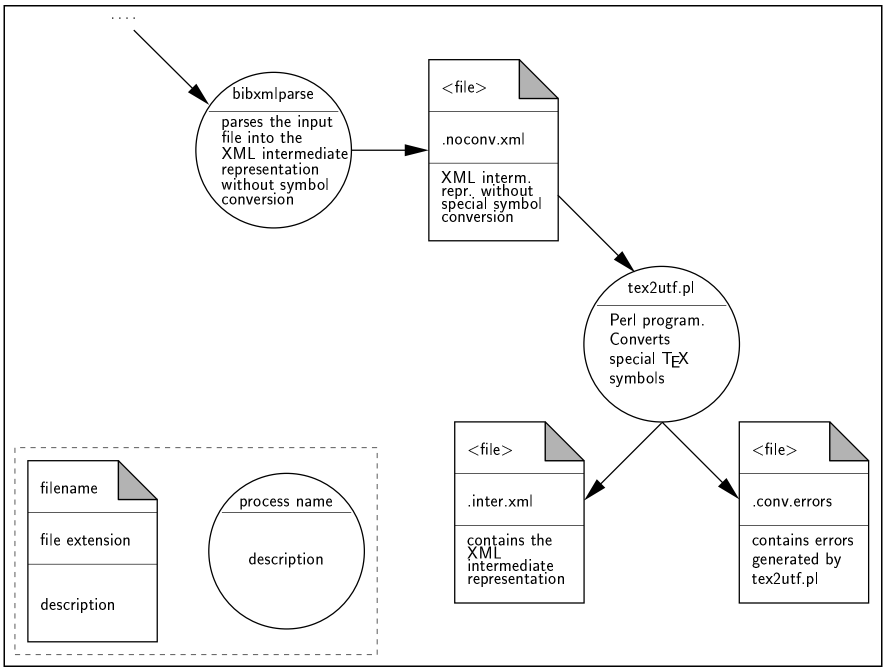
....
bibxmlparse
parses the input file into the XML intermediate representation without symbol conversion
<file>
.noconv.xml
XML interm. repr. without special symbol conversion
tex2utf.pl
Perl program. Converts special TEX symbols
<file>
.inter.xml
contains the XML intermediate representation
<file>
.conv.errors
contains errors generated by tex2utf.pl
filename
file extension
description
process name
description

Figure 4.4: BIBTEX to Intermediate Representation Conversion

a slash '\', and is followed by a special symbol and a letter. For example 'ä' and 'è' are typeset as `\"a` and `\‘e`. Since XML can not understand this character set, a conversion is needed. The best solution is to convert each TEX special character into its corresponding UTF-8 code[4]. We choose to do this conversion using a *Perl*-script which translates recognized character sequences into the corresponding UTF-8 representation (see Figure 4.4). *Perl* [22, 23] is an acronym for Practical Extraction and Report Language, a language optimized for text manipulation. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). We choose to implement this task using *Perl* because it combines (in the author's opinion, anyway) some of the best features of *C*,

[4]UTF-8 is a way of transforming all Unicode characters into a variable length encoding of bytes. It has the advantages that the Unicode characters corresponding to the familiar ASCII set have the same byte values as ASCII, and that Unicode characters transformed into UTF-8 can be used with much existing software without extensive software rewrites.

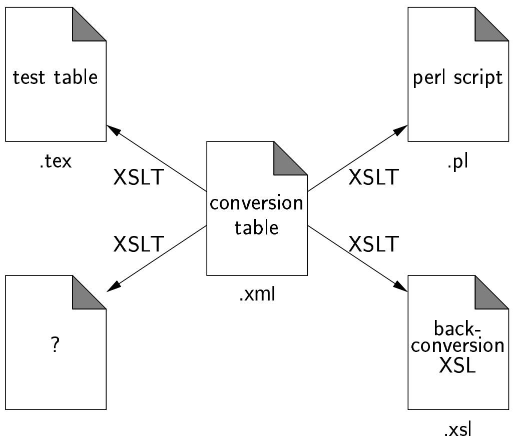

Figure 4.5: XML Conversion Table

sed[5], *awk*[6] and *sh*[7]. *Perl* expression syntax corresponds closely to C expression syntax and it can use sophisticated pattern matching techniques to scan large amounts of data quickly. *Perl* is available for most operating systems, including virtually all Unix-like platforms.

Since TEX does not define an UTF-8 code set a correspondence table is needed. We decide to build this table using XML. This solution provides many advantages: it is easy to write and to extend, and is "exportable" into various formats using XSLT (see Figure 4.5).

For example we developed a XSLT for transforming the conversion table into a LATEX file (see Section C.2). In this way it is possible to obtain a printable conversion file which enables us to check the correctness of the single TEX-UTF-8 entry.

TEX control sequences

A standard keyboard has very few keys compared to the large number of symbols that are usually needed. For example the Unicode organization[8] has coded over 50000 different symbols. In order to make a limited keyboard sufficient versatile, TEX defines an *escape character* which is reserved for

[5] *Sed* is a stream editor which is able to perform basic text transformations on an input stream

[6] *Awk* is a programming language whose basic operation is to search a set of files for pattern, and to perform specified actions upon lines or fields of line which contain instances of those patterns

[7] The *sh* utility is a command programming language that executes commands read from a terminal or a file

[8] `http://www.unicode.org`

special use [1]. Usually the “backslash” character ‘\’ is used for this purpose. Immediately after the ‘\’ follows the *control sequence*, a coded command which tell TEX what we have in mind. For example

```
\TeX
```

which causes TEX to print the TEX-logo. The string of character ‘`\TeX`’ is a control sequence. Another example is:

```
sch\"on
```

TEX converts this to ‘schön’. This control sequence (‘`\"`’) is used to place an accent over the ‘`o`’.

Control sequences come in two forms. The first one, like `\TeX`, is called a *control word*; it consists of an escape character followed by one or more letters[9] , followed by a space or by something besides a letter. The space after a control words is needed by TEX to recognize the end of the control sequence if its is followed by a letter.

A control sequence of the second kind, like `\"`, is called a *control symbol*; it consists of the escape character followed by a single *nonletter*. In this case is not needed a space to separate the control sequence from the letter that follows, control sequences of the second type always have exactly one symbol after the escape character.

When a space comes after a control word it is ignored by TEX; i.e., it is not considered to be a “real” space. But when a space comes after a control symbol, it is truly a space. To have a space which appears after a control word a “control space” is needed:

```
\␣
```

TEX will treat this as a space that is not to be ignored. To have a better control over spacing it is also possible to use the grouping mechanism. For example it is a mistake to type ‘`\TeX\`’ when the following character is not a blank space. In all cases it would be correct to type

```
{\TeX}
```

whether or not the following character is a space, because the ‘`}`’ stops TEX from absorbing an optional space. Another possibility is to type

```
\TeX{}
```

[9]for TEX a letter is one of the 52 symbols `a...z` and `A...Z`. The digits `0...9` are *not* considered letters

using an empty group for the same purpose. The '{}' does not produce output, but it does have the effect of stopping TEX from skipping spaces.

TEX understands about 900 control sequences. It is also possible to extend this built-in set. For example we defined with the `\def` command the control sequence

```
\def\BibTeX{B{\sc ib}\TeX}
```

which prints 'BIBTEX' typing `\BibTeX{}`.

Special Symbols in BIBTEX

Since version 0.99b [4], BIBTEX handles accented characters. For example using the `alpha` bibliography style, an entry with the following two fields

```
author = "Kurt G{\"o}del",
year   = 1931
```

will have the label "[Göd31]". To get this feature to work the entire accented character must be placed in braces; in this case either `{\"o}` or `{\"{o}}` will do. Furthermore these braces must not be enclosed in braces themselves (other than the field delimiting braces). Thus neither `{G{\"o}del}` nor `{G{\"{o}}del}` will work. BIBTEX considers an accented character is really a special case of a "special character", which consists of everything from a left brace at the top-most level, immediately followed by a backslash, up through the matching right brace. For example in the field

```
author = "\AA{ke} {Jos{\'{e}}} {\'{E}douard} G{\"o}del"
```

there are just two special characters, '`{\'{E}douard}`' and '`{\"o}`' (the same would be true if the pair of double quotes delimiting the field were braces instead). In general BIBTEX will do not any process of a TEX or LATEX control sequence inside a special character, but it will process other characters.

Many of the TEX control symbol (like '`\"a`' or '`\`e`') are also coded in the Unicode format. Thus it is possible to translate then into the corresponding UTF-8 code. The following BIBTEX field

```
publisher = "La Tromb{\'e}e"
```

is represented in BIBTEXML as

```
<publisher>La Tromb&#x00E9;e</publisher>
```

where '`&#x00E9;`' is the UTF-8 hexadecimal code of the symbol 'é'.

The BibTeXML file is converted back into the original BibTeX format using XSLT. An XSLT translates all Unicode compliant symbols in TeX control symbols (e.g. `&#x00E9;` to `\'e`).

TeX control words (like '`\TeX`' or '`\LaTeXe`') do not have any correspondence in the Unicode. Therefore the already explained substitution does not apply. Another mechanism, which enable full TeX compatibility is needed. The proposed solution consists in a special XML tag that is added when such a control word must be represented in BibTeXML. The following BibTeX field

```
title = "{\LaTeXe} guide"
```

is represented in BibTeXML as

```
<title><tex code="\LaTeXe">LaTeX2e</tex> guide</publisher>
```

The XML tag `<tex code="\LaTeXe">LaTeX2e</tex>` enables a full TeX compatibility (the `code` attribute value is selected and inserted in the BibTeX format) and the possibility to specify "human readable" substitution strings (for example when parsing the BibTeXML file into a HTML or plain text the sequence 'LaTeX2e' is used).

4.3 Intermediate Representation to BibTeXML Conversion

In the previous chapter we described how the *bibxmlparse* program transforms BibTeX files into the XML format. This XML files are only an intermediate representation of the bibliographies. With a simple XSLT [7] stylesheet the files are transformed into the final BibTeXML format (see Figure 4.6).

This stylesheet is called `i2bibtexml.xsl` and has to perform three main tasks:

- Recognize not standard fields: as described in the second chapter, BibTeX format has about 20 standard fields. Users can anyway define their own fields. After the transformation, not standard fields are put in the `<nstandard>` element.

- Create attributes: many BibTeX fields have practically the role of a key. For example the label at the beginning of a BibTeX item is a key which identify the item. So it is meaningful to transform BibTeX labels in XML attributes of type ID.

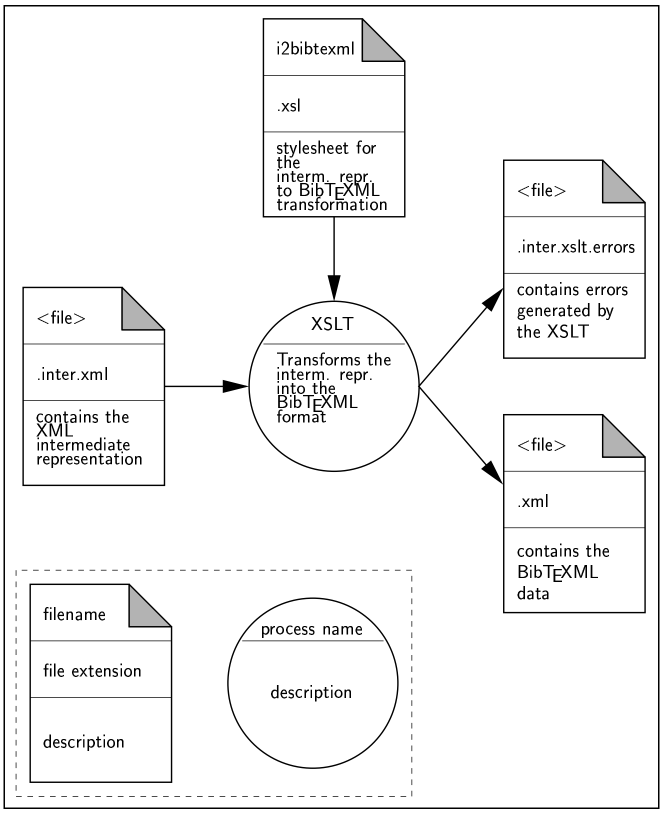
i2bibtexml
.xsl
stylesheet for the interm. repr. to BibTEXML transformation
<file>
.inter.xml
contains the XML intermediate representation
XSLT
Transforms the interm. repr. into the BibTEXML format
<file>
.inter.xslt.errors
contains errors generated by the XSLT
<file>
.xml
contains the BibTEXML data
filename
file extension
description
process name
description

Figure 4.6: Intermediate Representation to BIBTEXML Conversion

- Perform the subdivision of author names: BibTeX has an accurate syntax for author names. Analyzing this syntax the stylesheet recognize first, middle and last name of the author.

Names in BibTeX

Many people use BibTeX, but only few know the exact syntax for names. A name, in the BibTeX format, consists of a list of tokens which are divided in four parts: 'first', 'von', 'last', and 'jr'. Tokens are separated by whitespace or commas. Thus the name:

```
van Gogh, Vincent
```

has three tokens, whereas the name

```
{Previtali, Lurati, and Wilde}
```

has a single token. The name in this case is delimited by curly braces which means that it should be considered as a single token. How tokens are divided into parts depends on the form of the name. If the name has no commas, then it is assumed to be in either "first last" or "first von last" form. If there are no tokens that start with a lower-case letter, then "first last" form is assumed: the final token is the last name, and all other tokens compose the first name. Otherwise, the earliest contiguous sequence of tokens with initial lower-case letters is taken as the 'von' part.

If a name has a single comma, then it is assumed to be in "von last, first" form. A leading sequence of tokens with initial lower-case letters, if any, forms the 'von' part; tokens between the 'von' and the comma form the 'last' part; tokens following the comma form the 'first' part.

If a name has more than two commas, a warning is printed and the name is treated as though only the first two commas were present.

Finally, if a name has two commas, it is assumed to be in "von last, jr, first" form. (This is the only way to represent a name with a 'jr' part.) The parsing of the name is the same as for a one-comma name, except that tokens between the two commas are taken to be the 'jr' part.

The strings "John Dilbert" and "Dilbert, John" are different representations of the same name, so split into parts and tokens the same way, namely as:

```
first => ("John")
von   => ()
last  => ("Dilbert")
jr    => ()
```

If more than two tokens are included and there is no comma, they will go to the first name: thus “Donald E. Knuth” splits into

```
first => ("Donald", "E.")
last  => ("Knuth")
```

and “B. C. A. Lurati” into

```
first => ("B.", "C.", "A.")
last  => ("Lurati")
```

The ambiguous name “Henri Toulouse Lautrec” splits into

```
first => ("Henri", "Toulouse")
last  => ("Lautrec")
```

which is not the right thing. There is no way to know if the painter last name is “Toulouse Lautrec” and not only “Lautrec”, the string representation of his name must disambiguate. One possibility is “Toulouse Lautrec, Henri” which splits into

```
first => ("Henri")
last  => ("Toulouse", "Lautrec")
```

Alternately, the name can be represented as “Henri {Toulouse Lautrec}”, which becomes

```
first => ("Henri")
last  => ("{Toulouse Lautrec}")
```

Multi-token last names with lowercase components – the “von part” – work fine: both “Vincent van Gogh” and “van Gogh, Vincent” is correctly parsed into

```
first => ("Vincent")
von   => ("van")
last  => ("Gogh")
```

This allows aristocratic names to sort properly. For example “Charles Louis Xavier Joseph de la Vallée Poussin” is handled just fine, and splits into

```
first => ("Charles", "Louis", "Xavier", "Joseph")
von   => ("de", "la")
last  => ("Vall{\'e}e", "Poussin")
```

However, capitalized “von parts” don’t work so well: “Marco Van Basten” splits into

```
first => ("Marco","Van")
last  => ("Basten")
```

which is clearly wrong. This name should be represented as “Van Basten, Marco”

```
first => ("Marco")
last  => ("Van","Basten")
```

which is right.

Finally, many names include a suffix: “Jr.”, “III”, “Sons”, and so forth. These are handled, but with some limitations. If there’s a comma before the suffix (the usual U.S. convention for “Jr.”), then the name should be in “last, jr, first” form, e.g. “Bush, Jr., George W.” comes out (correctly) as

```
first => ("George","W.")
last  => ("Bush")
jr    => ("Jr.")
```

but “George W. Bush, Jr.” is ambiguous and is parsed as

```
first => ("Jr.")
last  => ("George", "W.","Bush")
```

If there is no comma before the suffix – the usual for Roman numerals, and occasionally seen with “Jr.” – then you are stuck and have to make the suffix part of the last name. Thus, “Gates III, William H.” comes out

```
first => ("William", "H.")
last  => ("Gates", "III")
```

but “William H. Gates III” is ambiguous, and becomes

```
first => ("William", "H.", "Gates")
last  => ("III")
```

– not what you want. Again, the curly-brace trick comes in handy, so “William H. Gates III” splits into

```
first => ("William", "H.")
last  => ("{Gates III}")
```

There is no way to make a comma-less suffix the jr part. Unfortunately it is also not possible to write names with title or qualification (e.g. Sir, Dr., Eng., and so on)

Finally, names that are not really names of people but rather are organization or company names should be forced into a single token by wrapping them in curly braces. For example, "Addison & Wesley" should be written "{Addison & Wesley}", which will split as

```
last  => ("{Addison \& Wesley}")
```

Names in BibTeXML

As shown before, BibTeX splits names in four token ('first', 'von', 'last', 'jr'). For BibTeXML we choose a different approach. BibTeXML still splits names into four parts but in a different way. Take for example the name "Charles Louis Xavier de la Vallée Poussin" which becomes,

```
<firstname>Charles</firstname>
<middlename>Louis</middlename>
<middlename>Xavier</middlename>
<lastname>de la Vall{\'e}e Poussin</lastname>
```

The difference is that middlenames are separated from first names and "von" parts are included in the `<lastname>`. This should be more convenient. There is no reasons to keep the "von" part separated from the "last" part. People normally does not make this separation, "von" is considered simply as a part of the last name. In addition it make no sense to have multiple first names. Actually when we look for a person we know only one first name and the last name of the person but probably we ignore the middle name and the suffix. Middle names are required to distinguish people with the same name. So for queries in a database the name structure: 'first', 'middle', 'last', 'jr' should be comfortable.

Attributes in BibTeXML

During the transformation some BibTeX fields are transformed into XML attributes. Why are these fields not simply transformed into XML elements? There are various reasons. An XML attribute describes the content of an element, it is practically a meta data, a characteristic of an element. Thus the BibTeX standard entry types, which describe the document type, are transformed in the attribute `type`. In addition there are some attribute features that simplify the compatibility with BibTeX. The key that identifies an entry in BibTeX must be unique inside a BibTeX document. This key is transformed in the BibTeXML attribute `label`. The XML Schema defines the `label` as an ID. So if two element have the same label value, the XML document can not be validated. The same mechanism is applied to other BibTeX fields, for example to `@macro`.

Not standard field

In BibTeX you can define your own fields. For example it is possible to write:

```
myfieldname = "myfieldcontent"
```

At the beginning this field was just transformed into the form

```
<myfieldname>myfieldcontent</myfieldname>
```

which was not so good because the element `<myfieldname>` was not defined in the BibTeXML Schema. So the transformed file could not be validated. At the end we chose a different approach. Every not standard field must be transformed as follows

```
<notstandard name="myfieldname">myfieldcontent</notstandard>
```

This solution assures a better compatibility with BibTeX, which is the most important goal of the project.

Chapter 5

BibTeXML to BibTeX Conversion

5.1 `xml2bib` Stylesheet

If you want to generate references for a LaTeX document, you have first to convert your BibTeXML file into a BibTeX file. To do this we wrote a XSLT stylesheet (`http://bibtexml.org/stylesheet/xml2bibt.xsl`), which converts BibTeXML files into BibTeX files. The process is performed in two steps (see Figure 5.1).

In the first step of the conversion the XSLT processor analyses the structure of the stylesheet and controls the validity of the BibTeXML document according to the BibTeXML Schema. If the document is valid the document will be transformed, otherwise the processor generates an error. Normally BibTeXML documents do not contain errors because XML editors help the user writing valid XML document. XML Spy, for example, loads the DTD or the Schema and suggests which elements are permitted in the BibTeXML document and which not. In the second step of the transformation, special UTF-8 characters are converted into LaTeX expressions. Originally this task was performed directly by the XSLT processor. Unfortunately tests demonstrated that this solution had performance problems so we implemented this part using *Perl*.

5.2 XPath Advantages and Weakness

XPath [24] is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer. In addition to its use for addressing, XPath is also used for matching elements, expressions, conditions, etc. It also

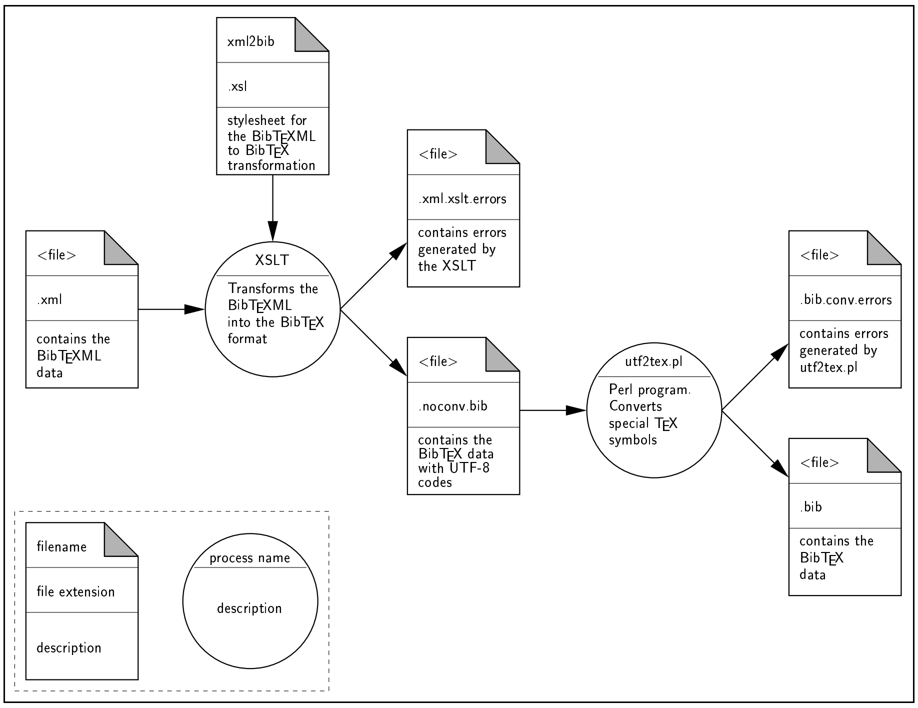
xml2bib
.xsl
stylesheet for the BibTEXML to BibTEX transformation
<file>
.xml
contains the BibTEXML data
XSLT
Transforms the BibTEXML into the BibTEX format
<file>
.xml.xslt.errors
contains errors generated by the XSLT
<file>
.noconv.bib
contains the BibTEX data with UTF-8 codes
utf2tex.pl
Perl program. Converts special TEX symbols
<file>
.bib.conv.errors
contains errors generated by utf2tex.pl
<file>
.bib
contains the BibTEX data
filename
file extension
description
process name
description

Figure 5.1: BIBTEXML to BIBTEX Conversion

provides basic facilities for manipulation of strings, numbers and boolean. The main advantage of XPath is the simplicity. Even programmers with few experience learn XPath quickly without problems. In addition, XPath allows the user to do complex query on XML documents. This makes interesting the representation of data in XML. But XPath has also some weaknesses. First XPath is focused on XML element matching, for this reason it has only the following few string functions:

- `string(object)`: converts an object to a string
- `concat(string, string, string)`: returns the concatenation of its arguments.
- `starts-with(string, string)`: the starts-with function returns true if the first argument string starts with the second argument string, and otherwise returns false.
- `contains(string, string)`: returns true if the first argument string contains the second argument string, and otherwise returns false.
- `substring-before(string, string)`: returns the substring of the

first argument string that precedes the first occurrence of the second argument string in the first argument string.

- `substring-after(string, string)`: returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string.
- `substring(string, number, number)`: returns the substring of the first argument starting at the position specified in the second argument with length specified in the third argument.
- `string-length(string)`: returns the number of characters in the string.
- `normalize-space(string)`: returns the argument string with whitespace normalized.
- `translate(string, string, string)`: returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string. For example, translate("bar","abc","ABC") returns the string BAr. (Note: It is not possible to transform string but only single characters.)

With only this function even the simplest task become complicated. If for example we want to transform the character & into a LaTeX expression `\&{}` we have to write the following code:

```
<xsl:if test="contains(string(element),'&')">
  <xsl:value-of select="substring-before(string(element),'&')"/>
  \&{}
  <xsl:value-of select="substring-after(string(element),'&')"/>
</xsl:if>
```

This substitution should be performed for every element in a document. In this example we have six XPath functions and three XSL elements and this could become a performance problem. Actually tests shows that today XSLT processor are quite slow. In particular loops slow down strongly the transformation. (The transformation with UTF-8 characters conversion with XSLT required 10 times more time than a normal transformation without the character conversion) These arguments convinced us to use a *Perl* program to perform the character conversion. This program (`utf2tex.pl`) is described in Section C.2. In the near future the user will have a lot of additional function so it would be easier to write compact and efficient XSLT stylesheets.

Chapter 6

bibtexml.org

http://bibtexml.org is the final product the diploma project. In this website it is possible to find information about the design and the development of the project. http://bibtexml.org is a good starting point for people who wants to learn BibTeX. Actually the website has several useful links to BibTeX and LaTeX tutorials, which could help BibTeX beginners. But the main purpose of the homepage is to give the user the opportunity to test some application for bibliographies management. In particular the user can take advantage of the services offered online. The following application are today available:

- Check the correctness of a BibTeX file with *bibclean* and *bibparse*.
- Convert a BibTeX file into BibTeXML format.
- Convert a BibTeXML file into the BibTeX format.

The homepage is developed using various technologies (SSI, CGI, PHP). The file uploading and the conversion has been realized using PHP files and are described in Appendix D. After the uploads the PHP file performs a scanning of the BibTeX file using bibclean and bibparse. If the file is "well-formed" the result appears, otherwise the website shows the detected errors and warnings. Another very important aspect of http://bibtexml.org is the user feedback. Only with the help of user it is possible to discover flaws in the implementation. If you have questions, comments and bug reports there are contact addresses (http://bibtexml.org/contact.shtml) and a BibTeXML forum (http://bibtexml.org/wwwboard.shtml).

Chapter 7

Conclusions

7.1 Test Results

We tested the BibTeXML environment using various BibTeX collections fetched browsing the Web.

The first consideration, is about the correctness of the collected bibliography: most of the files we used do not pass the *bibclean* and the *bibparse* syntax check. About the half of the tested files present one or more errors. The recognized formatting errors are almost ever the same. We can summarize these in several groups:

- Typesetting errors: BibTeX users often forget to match quotation marks or brackets or does not insert the separating coma at the end of the field. Another common typesetting problem is the field termination with a backslash. The following entry

  ```
  title = "System and \signal\"
  ```

  is invalid because the field text is not matched: the '`\"`' at the end of the line is not interpreted as a delimiting quotation mark but it as the equivalent TeX control symbol.

- Inappropriate use of the underscore '`_`' symbol: underscores are not allowed within field names. For example the following definition

  ```
  my_comment  = {hello}
  ```

  is rejected.

- Too long fields: `abstract` fields are frequently too long.

- Key duplication: often reference keys are not unique.

Another very common error is the usage of the wrong syntax for specifying names. In particular a minority respects the syntax for names which have a suffix like "John Smith Jr." and "John Smith II". Many users write `"John Smith, Jr."` or `"Smith, John, Jr."` instead of `"Smith, Jr., John"`. Unfortunately it is almost impossible to detect automatically these kind of errors. To help the user typesetting this fields, a web page on `http://bibtexml.org` explains the right BibTeX name syntax. Because of the general incorrectness of BibTeX files, it is very important for the user to have a detailed explanation of the errors, thus we provide an online FAQ explaining the most common mistakes.

The second consideration is about the quality of the collected bibliographies which have been independently created by many different authors. A consequence of this, and the fact that BibTeX is an extendible bibliography format without strict rules about the use of fields, is that the bibliographies have very diverse formats. This makes the uniforming of not standard BibTeX fields difficult: for example the field `abstract` and the field `comment` store similar informations using different names. Furthermore there are many duplicate entries in the collections. This facts is noticeable in databases which store BibTeX data: The number of duplicate entries in the collection grows rapidly, as more and more bibliographies are integrated. This overlap between different bibliographies is not easy to be avoided. As long as the spelling of titles is correct, the duplicate entries can be removed automatically to create a database without redundancy, but the complex syntax of the name fields does not facilitate the normalization of the `author` or of the `editors` information.

7.2 Technical Considerations

During the time we spent on this work, we had the chance to make a lot of experience in the various standard we used. This opportunity showed us the strengths and weakness of the current technology in this field.

BibTeX is a really widely used format, but the absence of a formally defined grammar does not allow an efficient data manipulation. In addition, the extending potential of TeX makes a lot harder the whole parsing process: we spent several evenings testing the various allowed possibility to specify TeX control sequences. The documentation available for BibTeX leaves some points about the syntax unclear [19]. We hope that the next BibTeX version (which should be version 1.0) will have a rigorous grammar.

XSLT and **XPath** are efficient modules for handling XML documents. Unfortunately they still not have enough features, especially for complicated tasks. Particularly the manipulation of strings and objects is often difficult because of the lack of appropriate functions.

7.3 Conclusions and Future Work

The testing phase showed that the chosen approach for the BibTeXML implementation is quite good and our publication to the poster track of the Tenth International World Wide Web Conference (WWW10) [25] confirms this opinion. Formatting bibliographies entries with XML enable us to obtain a clean and effective structure, thus making possible the creation of more powerful bibliographic databases and their manipulation using general-purpose XML-tools. The conversion tools that we have developed allow the translation of available BibTeX collections into the new format, and the BibTeXML database can also be translated to the original BibTeX-compatible format.

We think that the actual BibTeXML status is a good starting point for further developments. The website `http://bibtexml.org` can be improved with additional services such as XSLT translating BibTeXML into other formats like HTML or PDF. During the development of the project we study the possibility to normalize BibTeXML data. As explained in the previous Section, data redundancy is a serious problem of existing bibliography collections. The normalization should ideally be implemented using macros which eliminates data duplication. The main problem is to develop an efficient method which compares entries and selects data with the same meaning. The most important extension of the BibTeXML project is surely the development of an interface to the *XLinkbase* system, which is designed for managing large amounts of highly interlinked information, using a data model similar to *Topic Maps*. *XLinkbase* should make possible to easily browse and manipulate BibTeX entries, while BibTeXML is used as import and export format for the system.

We are also very curious about the suggestions which will come from the feedback of future users of the `http://bibtexml.org` homepage.

Appendix A

XML Schema

A.1 Standard Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema targetNamespace="http://bibtexml.org/STSCHEMA"
xmlns="http://bibteml.org/STSCHEMA xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:notbibtex="http://bibtexml.org/NSTSCHEMA">

<include SchemaLocation="http://bibtexml.org/schemas/notbibtex.xsd"/>

<xsd:element name="bibliography">
  <xsd:complexType content="elementOnly">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="macro" type="macroType"/>
    <xsd:element name="preamble" type="xsd:string"/>
    <xsd:element name="bibitem" type="bibitemType"/>
    <xsd:element name="comment" type="xsd:string"/>
    </xsd:choice>
    <xsd:attribute name="version" type="xsd:string"/>
    <xsd:attribute name="author" type="xsd:string"/>
    <xsd:attribute name="created" type="xsd:string"/>
    <xsd:attribute name="lastmodified" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="bibitemType" content="elementOnly">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="address" type="addressType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="annote" type="annoteType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="author" type="authorType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="booktitle" type="booktitleType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="chapter" type="chapterType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="crossref" type="crossrefType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="edition" type="editionType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="editor" type="editorType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="howpublished" type="howpublishedType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="institution" type="institutionType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="journal" type="journalType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="key" type="keyType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="month" type="monthType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="note" type="noteType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="number" type="numberType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="organisation" type="organisationType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="pages" type="pagesType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="publisher" type="publisherType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="school" type="schoolType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="series" type="seriesType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="title" type="titleType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="type" type="typeType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="volume" type="volumeType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="year" type="yearType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="url" type="notbibtex:urlType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="ISSN" type="notbibtex:ISSNType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="ISBN" type="notbibtex:ISBNType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="abstract" type="notbibtex:abstractType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="contents" type="notbibtex:contentsType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="notstantard" type="notbibtex:notstandardType" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string" use="required"/>
  <xsd:attribute name="label" type="xsd:ID" use="required"/>
</xsd:complexType>

<xsd:complexType name="authorType" content="elementOnly">
  <xsd:element name="firstname" type="notbibtex:firstType"/>
  <xsd:element name="middlename" type="notbibtex:middleType"/>
  <xsd:element name="lastname" type="notbibtex:lastType"/>
  <xsd:element name="suffix" type="notbibtex:suffixType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="address" content="mixed">
 <xsd:element name="tex" type="notbibtex:texType"/>
 <xsd:element name="abbrev" type="notbibtex:notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="annoteType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="booktitleType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="chapterType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="crossrefType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="editionType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="editorType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
```

```
</xsd:complexType>

<xsd:complexType name="howpublishedType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="institutionType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="journalType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="keyType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="macroType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:attribute name="alias" type="xsd:ID" use="required"/>
</xsd:complexType>

<xsd:complexType name="monthType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="noteType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="numberType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="organisationType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="pagesType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="publisherType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="schoolType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>
```

```
<xsd:complexType name="seriesType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="titleType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="typeType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="volumeType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

<xsd:complexType name="yearType" content="mixed">
  <xsd:element name="tex" type="notbibtex:texType"/>
  <xsd:element name="abbrev" type="notbibtex:abbrevType"/>
</xsd:complexType>

</xsd:schema>
```

A.2 Not Standard Schema

```
<xsd:schema targetNamespace="http://bibtexml.org/NSTSCHEMA
xmlns="http://www.w3.org/1999/XMLSchema"
xmlns:notbibtex="http://bibtexml.org/NSTSCHEMA">

<complexType name="notstandardType" content="mixed">
  <element name="tex" type="notbibtex:texType"/>
  <element name="abbrev" type="notbibtex:abbrevType"/>
  <attribute name="name" type="string"/>
</complexType>

<complexType name="abstractType" content="mixed">
  <element name="tex" type="notbibtex:texType"/>
  <element name="abbrev" type="notbibtex:abbrevType"/>
</complexType>

<complexType name="ISBNType" content="mixed">
  <element name="tex" type="notbibtex:texType"/>
  <element name="abbrev" type="notbibtex:abbrevType"/>
</complexType>

<complexType name="ISSNType" content="mixed">
  <element name="tex" type="notbibtex:texType"/>
  <element name="abbrev" type="notbibtex:abbrevType"/>
</complexType>

<complexType name="contentsType" content="mixed">
  <element name="tex" type="notbibtex:texType"/>
  <element name="abbrev" type="notbibtex:abbrevType"/>
</complexType>
```

```
<complexType name="urlType" content="mixed">
  <element name="tex" type="notbibtex:texType"/>
  <element name="abbrev" type="notbibtex:abbrevType"/>
</complexType>

<complexType name="texType" content="mixed">
  <attribute name="code" type="string"/>
</complexType>

<complexType name="abbrevType" content="empty">
  <attribute name="alias" type="IDREF"/>
</complexType>

<complexType name="lastType" content="mixed">
  <element name="tex" type="string"/>
</complexType>

<complexType name="firstType" content="mixed">
  <element name="tex" type="string"/>
</complexType>

<complexType name="middleType" content="mixed">
  <element name="tex" type="string"/>
</complexType>

<complexType name="suffixType" content="mixed">
  <element name="tex" type="string"/>
</complexType>

</schema>
```

Appendix B

Parsing Process Schema

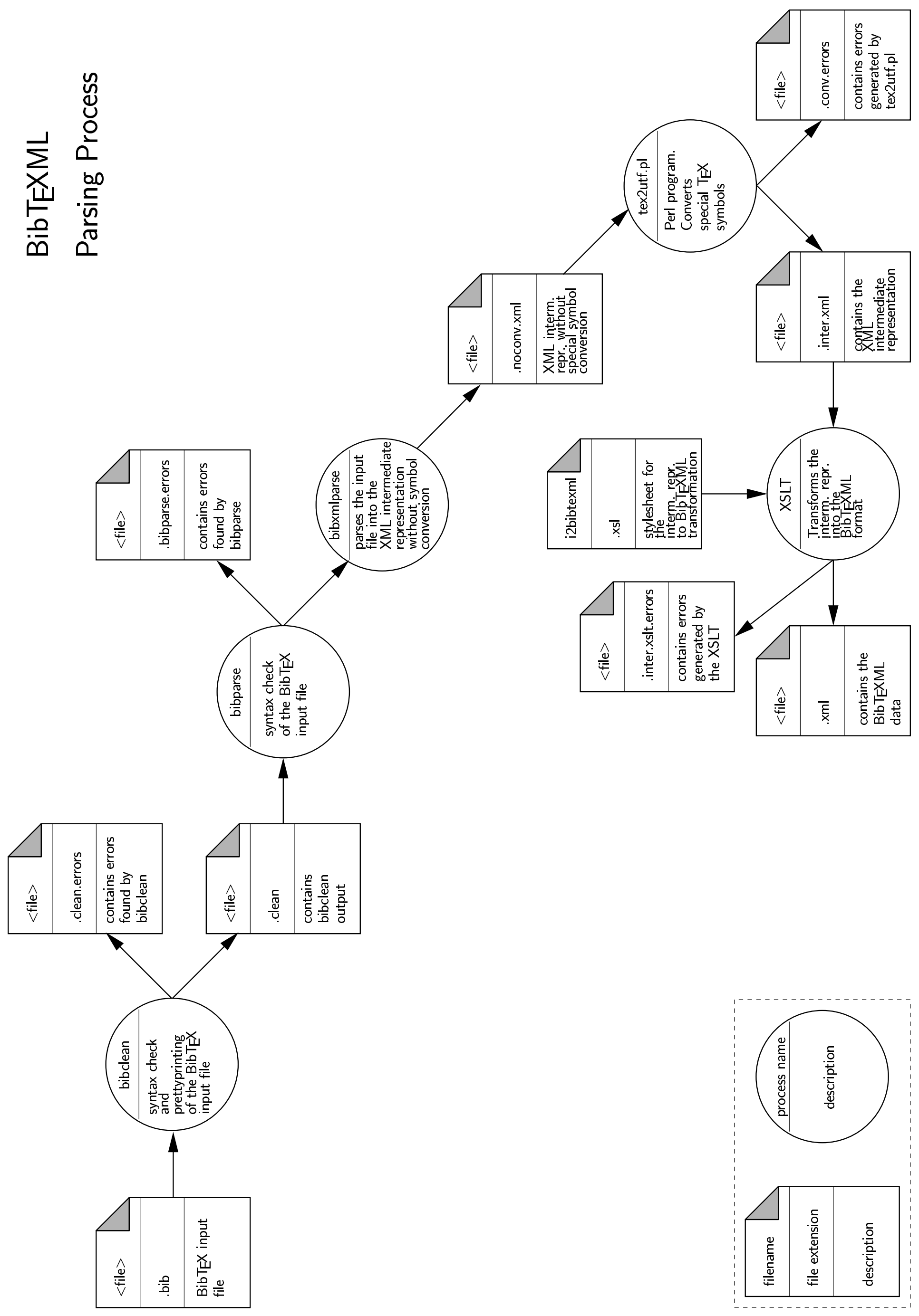
BibTEXML
Parsing Process
<file>
.bib
BibTEX input file
bibclean
syntax check and prettyprinting of the BibTEX input file
<file>
.clean.errors
contains errors found by bibclean
<file>
.clean
contains bibclean output
bibparse
syntax check of the BibTEX input file
<file>
.bibparse.errors
contains errors found by bibparse
bibxmlparse
parses the input file into the XML intermediate representation without symbol conversion
<file>
.noconv.xml
XML interm. repr. without special symbol conversion
tex2utf.pl
Perl program. Converts special TEX symbols
<file>
.conv.errors
contains errors generated by tex2utf.pl
<file>
.inter.xml
contains the XML intermediate representation
i2bibtexml
.xsl
stylesheet for the interm. repr. to BibTEXML transformation
XSLT
Transforms the interm. repr. into the BibTEXML format
<file>
.inter.xslt.errors
contains errors generated by the XSLT
<file>
.xml
contains the BibTEXML data
filename
file extension
description
process name
description

Figure B.1: Parsing Process Schema

Appendix C

Special Symbol Conversion

As explained in Section 4.2 the BibTeX format allows to typeset special symbols using the TeX character syntax. These rules are not meaningful in XML and a conversion into an XML compatible structure is needed.

The start point of the special symbols conversion mechanism is the conversion table containing the correspondences between the TeX symbol representation and the desired XML conversion. The *conversion table* is structured using XML. This solution provides many advantages: for example the *Perl* script and the XSLT for the back conversion are automatically generated applying XSLT stylesheets. It is also possible to transform the XML table into a printable format or to generate a test file. Finally, the extension of the set of recognized symbols is a simple task.

C.1 *Conversion table* Format

As already explained the *conversion table* is written using XML. This file begins with the root element

```
<convtable>
```

which is followed by all the correspondences. We divided the correspondences into two sets. The first group is used to specify all symbols which have an Unicode code (e.g. 'ö' or 'è') while the second specifies TeX symbols which does not have an Unicode representation (e.g. 'LaTeX').

The element which specifies the Unicode compliant symbols is named `unicode`. It has empty content, two mandatory attributes (`code` and `tex`) and an optional attribute `perl`. The `code` attribute specifies the UTF-8 value of the symbol which can be expressed using either the *decimal* or the *hexadecimal* form. The `tex` attribute stores the TeX representation of the symbol. While TEX defines different ways to specify these symbols (see Section 4.2) we allow the form `\*{*}` only. For example, the entry relative to 'è' is:

```
<unicode code="232" tex="\`{e}"/>
```

TEX symbol specifications may contain characters which confuse XSLT like the double quotes '`"`' used in '`\"o`': in this case the `tex` attribute must contain the Unicode representation of the offending character (e.g. `&#x0022;`). For example the declaration for 'ö' is:

```
<unicode code="246" tex="\&#x0022;{o}"/>
```

In *Perl* there are different reserved characters called *metacharacters* which don't match themselves. These metacharacters are:

```
\ | ( ) [ { ^ $ * + ? .
```

To turn a metacharacter into a literal character the backslash is used (e.g. `?` → `\?`). Thus during the *Perl* script generation a backslash is automatically inserted at the beginning of each of the `tex` attribute (e.g. `\`{e}` becomes `\\`{e}`). If this feature is not desired (e.g. for the em-dash `---`) or not sufficient (e.g. `\^o`) it is possible to use the optional `perl` attribute to specify how the symbol should be interpreted. For example the symbol 'ô' is obtained typesetting `\^{o}` and has the UTF-8 code 244. Since it is composed from two *Perl* metacharacters ('`\`' and '`^`') it cannot be simply turned into a *Perl* script adding a backslash at the beginning (the result would be '`\\^o`" but the '`^`' still remains a metacharacter). In this case the correct transformation (e.g. `\\\^{o}`) is specified with the `perl` attribute:

```
<unicode code="244" tex="\^{o}" perl="\\\^{o}"/>
```

In this case when the element is transformed into a *Perl* regular expression, the value of the `perl` attribute instead of the value of the `tex` attribute is used.

For non–Unicode symbols a similar technique is adopted: The element which specifies the non–Unicode compliant symbols is named `string`. It has an empty content and two mandatory attributes `str` and `tex`. The `tex` attribute follows the identical rules as the corresponding attribute of the `unicode` element. The `str` attribute value stores the string that should be used instead of the original TEX symbol. For example

```
<string tex="\LaTeXe" str="LaTeX2e"/>
```

tells that instead of the `\LaTeXe` sequence (which produces the LATEX 2_ε logo) the character sequence `LaTeX2e` should be used.

C.2 XSLT Stylesheet

Perl script generation

tex2utf.pl

`tex2utf_pl.xsl` is the stylesheet used to translate the conversion table into the *Perl* script. The first relevant line of this file is:

```
<xsl:template match="convtable">#! /usr/bin/perl
```

This line specifies a template and prints the first script line (`#! /usr/bin/perl`) which sets the full path of the *Perl* interpreter[1]

This is followed by the *Perl* code, that reads the single input lines:

```
LINE:
while (&#x003c;>){
...
```

The Unicode representation of the '`<`' symbol (`&#x003c;`) is necessary because this character represents the beginning of a tag in XML.

The loop body contains "hard coded" regular expressions which uniform the different ways for typesetting special symbols allowed by TEX.

```
s/\\(i|j|l|L|S|P|o|O)\ /\\$1\{\}/g;
s/\\(i|j|l|L|S|P|o|O)~/\\$1\{\}\ /g;
...
s/(\\`)\ *(([a-zA-Z])|((\\(i|j|l|L|S|P|o|O))\{\}))/
    $1\{$3$5\}/g;
s/(\\')\ *(([a-zA-Z])|((\\(i|j|l|L|S|P|o|O))\{\}))/
    $1\{$3$5\}/g;
...
s/(\\u)((\ +([a-zA-Z]))|(\ *((\\(i|j|l|L|S|P|o|O))\{\})))/
    $1\{$4$7\}/g;
s/(\\v)((\ +([a-zA-Z]))|(\ *((\\(i|j|l|L|S|P|o|O))\{\})))/
    $1\{$4$7\}/g;
...
```

After this uniformation process the "unicode" transformations are applied.

At this point the residual braces around the unicode symbols are deleted and the transformation of the "string" symbols begins.

[1] `/usr/bin/perl` is the standard path of *Perl*.If the *Perl* interpreter is installed in a non–standard location, the correct path must be inserted

After this substitution the input is checked for non-recognized TEX entries and a warning message is printed to stderr.

```
select STDERR;
while (/(code=\")?(\\\S+)(\")?/g){
  if($1 ne "code=\"" and $3 ne "\""){
    print "WARNING: found an unknown symbol in line $.: $2\n";
  }
}
select STDOUT;
```

At the end of the whole process the '"' delimiters are deleted and the result is printed to stdout.

```
s/>"/>/g;
s/"&#x003c;/&#x003c;/g;

print $_;
```

utf2tex.pl

The BibTEXML architecture defines the use of XSLT for the whole BibTEX to BibTEXML conversion. To convert the Unicode entries into the TEX representation the only way is the use of the *XPath string* functions which are unfortunately very inefficient. Tests shown that the conversion of a typical bibliography file of 300KB requires about 5 minutes on a Sun Ultra Sparc 10 equipped with 512MB of RAM and using Xalan 1.2 [26] for Java. Waiting for a better implementation or new specifications of the *XPath* environment (*XPath 2.0* ?) we developed a *Perl* script which converts Unicode entries into TEX. Since the non Unicode entries are represented in BibTEXML with a special XML element this transformation is done using XSLT.

`utf2tex_pl.xsl` is the stylesheet responsible for generating the `utf2tex.pl` script. Like the `tex2utf_pl.xsl` XSL file, this stylesheet begins with the definition of the *Perl* path which is followed from the code responsible for reading the data to manipulate: We apply then the "unicode" transformations.

At the end, warnings are generated if a non–recognized Unicode entry is found:

```
select STDERR;
while (/(&#x0026;#\w+\;)/g){
    print "WARNING: don't now how to convert the
                    following unicode symbol $1\n";
}
select STDOUT;
```

This solution is only a temporary one and when *XPath* will provide efficient string matching and substitution mechanisms the whole BibTeX to BibTeXML conversion will be implemented using XSLT only improving the system portability.

Test file

The `conv_test.xsl` stylesheet generates a test file. This file contains all *conversion table* entries and it can be used for testing the *Perl* scripts responsible of the special symbols conversion.

LaTeX printable conversion table

The `conv_test.xsl` stylesheet produces a LaTeX file with two tables: The first contains all the Unicode compliant symbols while the second contains the non-Unicode symbols. The resulting LaTeX file can be processed and printed to check the correctness of the entries. Tables C.1 and C.2 are an extract of the test file.

symb	tex	code
...	...	...
À	`` \`A ``	192
Á	`\'A`	193
Â	`\^A`	194
Ã	`\~A`	195
Ä	`\"A`	196
Å	`\r A`	197
Æ	`\AE{}`	198
Ç	`\c C`	199
È	`` \`E ``	200
É	`\'E`	201
Ê	`\^E`	202
Ë	`\"E`	203
...	...	...

Table C.1: Unicode compliant symbols

symb	tex
...	...
LaTeX	`\LaTeX`
TeX	`\TeX`
LaTeX 2_ε	`\LaTeXe`
...	...

Table C.2: Non Unicode symbols

Appendix D

Program Files Description

D.1 `bibtexml.org` Related Files

At the moment the format transformation services offered on-line are performed by two PHP files.: `uploader.php`, `downloader.php`. This is a short description of the files involved in the transformation.

uploader.php : performs the transformation from BibTeX to BibTeXML. With the command `exec` starts two scripts: `testclean.sh` and `testparse.sh`. If `testclean.sh` and `testparse.sh` do not return errors, `complet.sh` is started, otherwise the errors are displayed. If `complet.sh` is successful, the result can be downloaded.

downloader.php performs the transformation from BibTeXML to BibTeX. It starts `back.sh`. If the transformation failed, errors are displayed on a HTML page otherwise the result appear in a frame.

testclean.sh performs a *bibclean* check of the input file. It generates two files: one for errors and one for warnings.

testparse.sh performs a *bibparse* check of the input file. It generates a file with the detected errors and a file with the reported warnings.

complet.sh executes the conversion from BibTeX to BibTeXML using `bibxmlparse` and the *Xalan* Processor.

back.sh executes the conversion from BibTeXML to BibTeX. It performs the transformation of input file using the *Xalan* processor. After that it starts the `conv.sh` script which transforms UTF-8 characters in LaTeX symbols.

bibclean.html error page for *bibclean* errors.

bibparse.html error page for *bibparse* errors.

converted.html this page appears when the conversion from BibTeX to BibTeXML is successfully.

bibtex.html this page appears when the conversion from BibTeXML to BibTeX is successfully.

xslterror.html this page is loaded when the XLST processor reports some errors.

error.pl is a *Perl* scripts which filters error messages. It takes as input the errors and warnings messages generated by *bibclean* and *bibparse*. It outputs to *stderr* an HTML file containing the error messages and to *stdout* an HTML file containing the warnings.

D.2 XSLT Stylesheets

i2bibtexml.xsl is the stylesheet that describes the transformation from the intermediate representation to BibTeXML. If this file undergoes some modifications , the structure of BibTeXML will probably change, so the file `bibtexml.xsd` must be modified.

xml2bib.xsl is the XSLT stylesheet that defines the rules for the transformation from BibTeXML to BibTeX. The modification of this file has no effects on the system. The output of the stylesheet (`<xsl:output>`) must be set on 'us-ascii', otherwise the *Perl* program `utf2tex.pl` can not transform correctly the file.

D.3 XML Schemas and DTDs

bibtexml.xsd is the XML Schema where standard BibTeX fields are defined. If you change this file you have to adapt the stylesheets `i2bibtexml.xsl` and `xml2bib.xsl`.

notbibtex.xsd is the XML Schema for not standard BibTeX fields. This Schema is included in the `bibtexml.xsd` Schema with the XSL element `<include>`.

bibtexml.dtd is generated by XML Spy from `bibtexml.xsd`. It is used to validate BibTeXML documents if an XML tool does not support XML Schema.

D.4 Parser and BibTeX tools

bibxmlparse is the C parser responsible for the transformation of BibTeX files into XML intermediate representation. The modifications on the code affect the intermediate representation. The source code of this program is in the `bibtexmlparse` directory. The README file explains how to compile it.

bibclean is program which checks the syntax and prettyprints BibTeX bibliographies. The program source code is available at `ftp://ftp.math.utah.edu/pub/tex/bib/`.

bibparse verifies the grammar of BibTeX files The program source code is available at `ftp://ftp.math.utah.edu/pub/tex/bib/`.

D.5 Character Transformation

The following files are in the `perl` directory. The README file explain how to apply the transformations.

convtable.xml is the XML document that containing the conversion table for the BibTeX special characters. Every time a new characters is added, *Perl* programs have to be generated using the stylesheets `tex2utf_pl.xsl` and `tex2utf_pl.xsl`.

tex2utf_pl.xsl is the stylesheet that transforms `convtable.xml` into the `tex2utf.pl` *Perl* program.

utf2tex_pl.xsl is the stylesheet that transforms `convtable.xml` into the `utf2tex.pl` *Perl* program.

conv_tex.xsl is the stylesheet that transforms `convtable.xml` into a tex file which can be compiled and printed.

tex2utf.pl *Perl* program generated by `tex2utf_pl.xsl`. It performs the transformation of BibTeX special characters into UTF-8 characters.

utf2tex.pl *Perl* program generated by `utf2tex_pl.xsl`. It performs the transformation of UTF-8 characters into BibTeX special symbols.

Appendix E

Beebe's Mail Exchange

During our tests we noticed a difference between the grammar implemented by *bibclean* and those implemented by *bibparse*. We informed Beebe of this problem and he answered us with two interesting e-mails.

```
Date: Tue, 13 Feb 2001 11:27:09 +0100 (MET)
From: Luca Previtali <lprevita@ee.ethz.ch>
To: Nelson H. F. Beebe <beebe@math.utah.edu>

Dear Mister Beebe,
we are the two students working on the diploma thesis about an
XML representation of BibTeX.
We noticed a difference between the output of bibclean
(version 2.11.4) and biblex (version 1.04) which we can not
understand.

The input bibliography is:

@Book{myBook,
  author =       "myAuthor",
  title =        "myTitle",
  publisher =    "myPublisher",
  year =         "1999",
  comment =      "myComment",
}

The problem hangs around the field "comment":
bibclean (bibclean -no-prettyprint) recognizes it as a
7 FIELD "comment" while biblex recognizes it as a
```

```
1 ABBREV "comment".

(This difference affects also the parsing process)
...
```

The first Beebe answer was:

```
Date: Tue, 13 Feb 2001 12:04:10 -0700 (MST)
From: Nelson H. F. Beebe <beebe@math.utah.edu>
To: Luca Previtali <lprevita@ee.ethz.ch>

Thanks very much for the problem report on biblex vs bibclean:
I can reproduce it with this simple file:

% cat foo.bib
@Article{x, comment="y" }

% bibclean -no-prettyprint foo.bib | grep -i comment
7       FIELD   "comment"

% biblex foo.bib | grep -i comment
1       ABBREV  "comment"

bibclean is correct; biblex is wrong.

I've got several consecutive meetings this afternoon, but
hope to have a patch late today.
...
```

Then Beebe wrote this interesting mail:

```
Date: Wed, 14 Feb 2001 12:46:36 -0700 (MST)
From: Nelson H. F. Beebe <beebe@math.utah.edu>
To: Luca Previtali <lprevita@ee.ethz.ch>
Subject: Re: bug in biblex and/or bibclean

I didn't finish my investigation of the inconsistency in
biblex and bibclean for the case of a key named "comment".
I'm still working on it.

he problem turns out to be more serious than I thought, and
it will take some time yet (possibly a few days, maybe longer)
```

to resolve it.

The original grammar in my 1993 TUGboat article explicitly says
that "comment", "include", "preamble", and "string" should be
lexed as TOKEN_ABBREV if they do not follow an at-sign.
I don’t know why I made that decision, rather than lexing
them as TOKEN_FIELD.

When I figure it out, I’ll then be able to decide whether it is
biblex, or bibclean, or the grammar, that needs changing.

I really would like to keep the grammar frozen (like a
national constitution), rather than amend it, even if
it has what for the moment appears to be a blemish.
Any change to the grammar would require an erratum to
be published in TUGboat, and a concomitant update in all
software that builds upon the grammar.
That is not a nice thing to have to do.
...

Acknowledgments

We would like to thank the following persons.
Erik Wilde, our supervisor assistant, for supporting us during our work.
Prof. Bernhard Plattner, for having accepted to be our supervising professor.
Mr. Nelson H. Beebe, for the very helpful explanations and suggestions about the BibTeX program.

Bibliography

[1] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, second edition, 1984.

[2] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, second edition, August 1994.

[3] Helmut Kopka and Patrik W. Daly. *A Guide to LATEX 2ε*. Addison-Wesley, Harlow, England, second. edition, 1995.

[4] Oren Parashnik. BIBTEXing. BibTEX version 0.99b, February 1988.

[5] Steve DeRose, Eve Maler, and David Orchard. XML Linking Language (XLink) version 1.0. World Wide Web Consortium, Proposed Recommendation PR-xlink-20001220, December 2000.

[6] Ron Daniel, Steven J. DeRose, and Eve Maler. XML Pointer Language (XPointer) version 1.0. World Wide Web Consortium, Candidate Recommendation CR-xptr-20000607, June 2000.

[7] James Clark. XSL Transformations (XSLT) version 1.0. World Wide Web Consortium, Recommendation REC-xslt-19991116, November 1999.

[8] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. World Wide Web Consortium, Recommendation REC-xml-names-19990114, January 1999.

[9] David C. Fallside. XML Schema Part 0: Primer. World Wide Web Consortium, Proposed Recommendation PR-xmlschema-0-20010316, March 2001.

[10] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. World Wide Web Consortium, Proposed Recommendation PR-xmlschema-1-20010316, March 2001.

[11] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. World Wide Web Consortium, Proposed Recommendation PR-xmlschema-1-20010316, March 2001.

[12] Oren Parashnik. Designing BibTEX styles. BibTEX version 0.99b, February 1988.

[13] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification.* The Java Series. Addison-Wesley, second edition, 2000.

[14] Elliot Berk and Martin Dirichs. JLex: A lexical analyzer generator for Java. `http://www.cs.princeton.edu/~appel/modern/java/JLex/`, 1998.

[15] Scott E. Hudson, Frank Flannery, C. Scott Ananian, Dan Wang, and Andrew W. Appel. CUP parser generator for Java. `http://www.cs.princeton.edu/~appel/modern/java/CUP/`, March 1998.

[16] B. W. Kernighan and D. M. Ritchie. *The C Programming Language.* Prentice-Hall, Upper Saddle River, NJ 07458, USA, second edition, 1988.

[17] John R. Levine, Tony Mason, and Doug Brown. *Lex & Yacc.* O'Reilly & Associates, second edition, 1992.

[18] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[19] Nelson H. F. Beebe. Bibliography prettyprinting and syntax checking. *TUGboat*, 14(4):395–419, December 1993.

[20] Nelson H. F. Beebe. *bibclean.* University of Utah, Salt Lake City, Utah, May 1998.

[21] Nelson H. F. Beebe. *bibparse.* University of Utah, Salt Lake City, Utah, May 1999.

[22] Larry Wall, Randal L. Schwartz, Tom Christiansen, and Stephen Potter. *Programming Perl.* Nutshell Handbook. O'Reilly & Associates, second edition, 1996.

[23] Randal L. Schwartz. *Learning Perl.* O'Reilly & Associates, August 1994.

[24] James Clark and Steven J. DeRose. XML Path Language (XPath) version 1.0. World Wide Web Consortium, Recommendation REC-xpath-19991116, November 1999.

[25] Luca Previtali, Brenno Lurati, and Erik Wilde. BibTEXML: An XML representation of BibTEX. In *Proceedings of the Tenth International World Wide Web Conference (WWW10)*, Hong Kong, May 2001.

[26] Xalan-Java overview. `http://xml.apache.org/xalan/overview.html`.

[27] Jeremy Peter Bennett. *Introduction to Compiling Techniques: a first course using ANSI C, Lex and Yacc.* McGraw-Hill, second edition, 1996.

[28] Nikos Drakos. The LATEX to HTML translator. Technical report, Computer Based Learning Unit, University of Leeds, 1994.

[29] Nikos Drakos. From text to hypertext: A post-hoc rationalisation of LATEX2html. *Computer Networks and ISDN Systems*, 27(2):215–224, November 1994.

[30] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LATEX Companion.* Addison-Wesley, Reading, Massachusetts, January 1994.

[31] Tobias Oetiker. The not so short introduction to LATEX 2_ε. ETH-Zurich.

[32] Chet Ramey and Brian Fox. *Bash reference manual.* Free Software Foundation, second edition, April 1998.

[33] Erik Wilde. *Wilde's WWW – Technical Foundations of the World Wide Web.* Springer-Verlag, Berlin, Germany, November 1998.

[34] Paul DuBois. *Using csh and tcsh.* O'Reilly & Associates, August 1995.