Towards Federated Referatories

Erik Wilde
ETH Zürich (Swiss Federal Institute of Technology)

Available online at http://dret.net/netdret/publications#wil03j

Abstract

Metadata usage often depends on schemas for metadata, which are important to convey the meaning of the metadata. We propose an architecture where users can extend the schema used by a system for managing referential metadata. Users can plugin new schemas and install custom filters for exporting metadata, so that users are not forced to limit their metadata to a fixed schema. The goal of this architecture is to provide users with a system that helps them managing their referatory, enables them with powerful tools to adapt the tool to their metadata, and still makes it possible to collect the metadata of several users in a central storage and exploit the common facets of the metadata. Our system is based on a specialized schema language, which has been built on top of the XML schema languages XML Schema and Schematron.

1 Introduction

In an academic institution such as the ETH Zürich, many scientific employees maintain their own bibliographies. For many of them, this is simply a matter of convenience, because centralized systems usually force them to conform to formats and limitations that they are not comfortable with. The downside of this situation is that there is no single and central way to manage all bibliographic information. This is a disadvantage for the employees, because they have to develop and maintain their own "systems" (which all too often are strategies how to handle the information with rather poor tools), as well as a disadvantage for the institution, because there is no way to combine and exploit the wealth of information maintained by the employees. Since we do not want to limit our view to bibliographic information only, we generalize this towards the notion of metadata describing referenceable resources, and the rest of the paper will use the term "referatory" instead of "bibliography".

It is important to notice that our system is not designed for resource storage or retrieval, it is exclusively designed for handling well-structured metadata, which probably has been generated manually.

A typical example for per-user referatories are BibTeX [11] files. BibTeX defines a number of standard fields, but also accepts additional fields, which may be processed by specialized BibTeX styles [12], or may never be used as part BibTeX processing, but are still relevant to capture metadata about the resource. Figure 1 shows an example of a BibTeX entry, which uses a standard entry type (`misc`) and a number of standard fields, as well two non-standard fields (`uri` and `topic`).

In this example, the `uri` field simply contains the resource's URI, while the `topic` field contains structured information. It contains a list of weighted references to a topic map of Web technologies[1], thus categorizing the resource in relation to these topics. Fields like these, which have been invented by a single

[1] Available online at http://wildesweb.com/glossary/.

```
@misc{xmlns10,
    author =        "Tim Bray and Dave Hollander and Andrew Layman",
    title =         "Namespaces in XML",
    howpublished =  "W3C, REC-xml-names-19990114",
    month =         "January",
    year =          1999,
    uri =           "http://www.w3.org/TR/1999/REC-xml-names-19990114",
    topic =         "xml[0.8] xmlns[1]" }
```

Figure 1: BibTeX Example

user or a small user group, often contain very valuable information for these users, but are unlikely to be of any relevance for the vast majority of BibTeX users. Thus, if the goal is to build a system supporting users in their maintenance of their references, it is essential to provide support for this kind of specialized resource metadata.

This is what is referred to in the title with the term "federated". The goal is to build a system that enables users to keep their personal style of creating and maintaining a referatory, while on the other hand provides features that exploits the joining of personal referatories. For example, a user might benefit from storing the entry shown in Figure 1 because the system will provide him with a Web-based interface, automatic backup, and (as will be shown later) very flexible ways of exporting the data. On the other hand, other users might benefit from the availability of this entry because they might have simply searched for documents of one of the authors, or because they find the resource's URI and recognize it as a simple way to get the resource.

To summarize, the goal is to design a system that on the one hand implements a central referatory, while on the other hand providing users with the opportunity to still handle their references as "their property", by granting them access rights, giving them the opportunity to use proprietary extensions, and providing them with a powerful concept for exporting (and thus reusing) references. The main motivation for this is the users' well-known hesitation to give up established and well working habits, as long as there is no perceived personal benefit.

2 Related Work

Not much work has been done so far in the area of federated referatories. However, some projects have covered parts of what our design goals were. The MyView project [18] is similar in some ways in that it also uses XML (in fact, it is based on SGML, but it is XML-compliant) and allows users to store custom metadata. However, the system does not support any kind of schema definition for metadata extensions, and it also does not support pluggable export filters.

The *Greenstone* architecture described by Witten et al. [17] is similar to our approach in that it is built around the idea of processing pipelines for importing documents and metadata. However, the system does not use the same approach for the export side, and also does not provide users with the ability to create their own schemas. In the Greenstone system, a wide variety of data can be imported due to the flexible plugin approach, but since there is no fixed digital library system behind it, it is impossible to make any statements about the schema language for metadata.

The *Federated Libraries on the Web (FLOW)* project by Gold et al. [10] is still in an early stage, but has the same focus as our system. It aims at creating an infrastructure which can be used to gather,

share, and discover metadata in research environments. It is based on an existing system, the *CERN Document Server (CDS)*, and will add support for the different layers of the FLOW architecture.

What we believe to be unique in our system is the definition of a dedicated schema language for schema extensions (described in Section 3), which is provided at the user level, so that every user may install schema extensions and then store data according to this schema. Another unique feature is the ability to install pluggable export filters (described in Section 4.2), which also are provided at the user level and make it easy to adapt the export format to the very different needs of heterogeneous user communities.

3 BibSchema Design

The core part of the system is the schema design for the referatory data that the system accepts and is able to handle. A first attempt towards such a schema has been based on DTDs and had some serious limitations in its support for datatypes and the extensibility of the schema [13]. To support a more flexible approach, DTDs are not the appropriate schema language foundation.

In order to overcome the limitations with regards to datatype support, the decision was made to build the next version of the schema language on top of XML Schema [14, 3] instead of DTDs. This solution solves the problem of datatype support, but does not help much in the area of "co-constraints", i.e. dependencies between different parts of the XML document, which are a central part of the way constraints are defined for entries. For this kind of schema information, the Schematron [8] language is ideally suited, which is a rather simple language for defining rule-based schemas, where the rules' central part are XPath [6] expressions.

The open question remained how to support the extensibility of the schema, so that users may easily define schema extensions. It seemed awkward to have users define an XML Schema and a Schematron part for each extension, and it also would have been very complex to check and guarantee the consistency of these distinct parts. Consequently, we designed a specialized schema language, which is used to define the core schema of our system as well as any extension to it. The schema language has been based on two major design considerations:

- *Extensibility:* The main goal of the system design is that users should be allowed to extend the schema and use these schema extensions to import any data they are interested in. Since users should not be restricted in their choice of extensions, the extension mechanism must support arbitrary extensions while maintaining the structural integrity of the referatory.

- *Mapping to Schema Languages:* XML schema languages[2] are evolving constantly. While XML Schema will probably have some level of success, newer approaches such as *Document Schema Definition Languages (DSDL)* [9] demonstrate that there must not be the "one size fits all" schema language. Instead, schemas may be composed out of different schema languages, specializing in different facets of schema definition facilities.

  Continuing this argument, it may make a lot of sense to define a specialized schema language for an application area, which is tailored to the needs of this application. Furthermore, to make implementation of this specialized schema language easier, one possible approach is to map it to a set of schema languages which are sufficient (or almost sufficient) to implement the specialized schema, so that only a minimal amount of code has to be produced and maintained.

[2] Whenever we refer to "XML Schema" (with a capital 'S'), we refer to the schema language defined by the W3C. When speaking of an "XML schema language" (with a lowercase 's'), we refer to the generic concept of a language for defining constraints for XML documents.

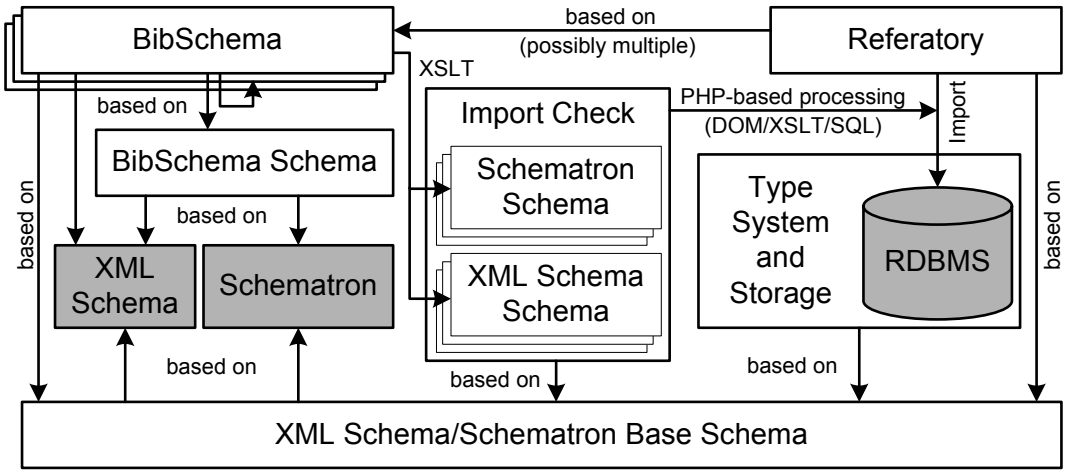

Figure 2: BibSchema Dependencies and Processing

In Figure 2, the overall design of the schema language that we developed and its dependencies are shown (shaded components are existing technologies that we used). The schema language is called *BibSchema*, and it is based on an XML Schema and a Schematron schema.

On the left side of the figure, the various layers of the schema languages can be seen. At the second layer from the bottom, it is shown that the schema for our BibSchema language, shown one layer above, is based on XML Schema and Schematron. Any BibSchema, shown at the top level, is based on our BibSchema schema. A BibSchema document defines a schema for entries in our system, using the BibTeX metaphors of *entries* which are composed using *fields*. Since fields may contain complex XML content, the BibSchema may define this content directly using XML Schema. BibSchemas may also be based on other BibSchemas, for example reusing fields from them. At the very bottom, the *base* schema is shown, which defines the basic datatypes that BibSchemas build on.

While the left side of the figure shows the schema side, the middle part shows the way of how data is checked to conform to the BibSchema(s) to which it claims to conform. The details of the implementation are described in Section 5, but here it is important to note that the schema information is used while importing the data into the system, so that the actual storage only contains BibSchema-valid data.

This is what the right side is showing: A referatory (a single reference or a large collection of references, for example an XML-encoded BibTeX document) is imported into the system by looking at the schema(s) it is using, and then checking the data when importing it. Again, details of the implementation of this process can be found in Section 5.

Within the architecture, there are two special schemas, one is the *base* schema (shown at the very bottom of Figure 2), defining the basic datatypes supported by the system, such as persons, cross-references between entries, a macro mechanism, and treatment of special text (such as text that must be treated differently depending on the output format[3]). The other special schema is the *standard* BibSchema, which defines the standard entry and field types of BibTeX. The standard schema is not shown separately in the figure and can be regarded as one of the BibSchemas shown in the upper left corner.

[3]For example, when exporting the reference to [11] for TeX processing, it would be good to have the BibTeX part of the title as "`B{\sc ib}\TeX{}`", while in other formats it would probably be sufficient to get it as "`BibTeX`".

```
<bs:schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:bs="http://dret.net/xmlns/bibtexml/bibschema"
    defaultRefAndTargetNS="http://dret.net/xmlns/bibtexml/standard">
 <bs:entry name="misc">
  <bs:minOne>
   <bs:field ref="author"/>
   <bs:field ref="title"/>
   <bs:field ref="howpublished"/>
   <bs:field ref="month"/>
   <bs:field ref="year"/>
   <bs:field ref="note"/>
  </bs:minOne>
 </bs:entry>
 <bs:field name="author" isPerson="true" repeatable="true"/>
 <bs:field name="title" isSpecialText="true"/>
 <bs:field name="month">
  <restriction base="string">
   <enumeration value="January"/> ...
```

Figure 3: BibSchema Example (Schema for `misc` Entry type)

To illustrate the dependencies among various components of the architecture, Figure 3 shows a very short excerpt from the BibSchema standard schema. In this example, there are a number of interesting things to note. One thing is that our approach is entirely based on XML Namespaces [5], identifying all BibSchemas on the basis of namespace names. The outline of the BibSchema language can be seen by the hierarchy of the element `schema`, the element `entry` (used to define an entry type), and the element `field` (used to define a field type). As shown in the example of the `month` field type, a field definition may contain normal XML Schema code.

Fields may also have qualifiers, which are either indicators for schema properties (`repeatable`), or require special treatment when processing these fields (`isPerson` and `isSpecialText`). Apart from these hard-coded properties of how fields may occur, and how fields are processed, everything in our system architecture is freely configurable (and thus could be easily modified to have a less BibTeX-based data model as the standard schema).

Building on base and standard schemas, users can define arbitrary extension schemas. One such schema is shown in Figure 4. This example is a very simple extension, only defining one new field type (`topic`). Since we are following BibTeX's convention that fields may always appear in entries (i.e., there is no such thing as an illegal field in an entry), this new field may be used in any entry type available through other BibSchemas. The field itself is defined to contain an XML Schema complex type, in this case a sequence of `topic` elements, which must carry a `name` as well as a `weight` attribute. The content of the `field` element is regular XML Schema code, and thus enables users to define arbitrarily complex field content.

This example extension schema concludes the discussion of the BibSchema design. Figure 5 shows how an instance of this example schema looks like, and Section 5 discusses the implementation of the schema architecture in our prototype system.

```
<bs:schema xmlns="http://www.w3.org/2001/XMLSchema"
           xmlns:bs="http://dret.net/xmlns/bibtexml/bibschema"
           defaultRefAndTargetNS="http://dret.net/xmlns/bibtexml/topic">
 <bs:field name="topics">
  <sequence>
   <element name="topic" maxOccurs="unbounded">
    <complexType>
     <attribute name="name" type="NCName"/>
     <attribute name="weight">
      <simpleType>
       <restriction base="decimal">
        <minInclusive value="0"/>
        <maxInclusive value="1"/>
        <fractionDigits value="1"/>
       </restriction>
      </simpleType>
     </attribute>
    </complexType>
   </element>
  </sequence>
 </bs:field>
</bs:schema>
```

Figure 4: BibSchema Extension Example (`topic` BibSchema)

4 Handling Data

The preceding section describes how schema design is supported. It is based on the assumption that there should be a general structure of the referatory (represented by the base and standard schemas), and that users should have the opportunity to extend this structure according to their specific needs. To demonstrate this, in Figure 5 it is shown how the initial example of a BibTeX entry with proprietary fields (as shown in Figure 1) can be encoded in XML.

The most interesting part of this example is the `bibliography` element, which carries four namespace declarations. These namespace declarations refer to the base and standard schemas (provided by the system itself, the relevant part of the standard BibSchema is shown in Figure 3), the `topic` BibSchema (as presented in the previous section in Figure 4), and a hypothetical `uri` BibSchema, defining a field type for containing URIs.

The example shows that the entry (represented by the `misc` element) comes from the standard BibSchema and contains a number of fields (the `misc` element's child elements). Most of the fields come from the standard BibSchema. Here it is interesting to note that the `author` elements contain child elements from the base BibSchema. The reason is that the `author` field is defined as containing person information (as can be seen in Figure 3). The `uri` field comes from a URI BibSchema (which is not discussed in this paper), and the `topics` field comes from the topic BibSchema (as shown in Figure 4).

```
<b:bibliography xmlns:b="http://dret.net/xmlns/bibtexml/base"
   xmlns:s="http://dret.net/xmlns/bibtexml/standard"
   xmlns:t="http://dret.net/xmlns/bibtexml/topic"
   xmlns:u="http://dret.net/xmlns/bibtexml/uri">
 <b:entries>
  <s:misc key="xmlns10">
   <s:author>
    <b:bibperson>
     <b:firstname>Tim</b:firstname><b:lastname>Bray</b:lastname>
    </b:bibperson>
   </s:author>
   <s:author>
    <b:bibperson>
     <b:firstname>Dave</b:firstname><b:lastname>Hollander</b:lastname>
    </b:bibperson>
   </s:author>
   <s:author>
    <b:bibperson>
     <b:firstname>Andrew</b:firstname><b:lastname>Layman</b:lastname>
    </b:bibperson>
   </s:author>
   <s:title>Namespaces in XML</s:title>
   <s:howpublished>W3C, REC-xml-names-19990114</s:howpublished>
   <s:month>January</s:month>
   <s:year>1999</s:year>
   <u:uri>http://www.w3.org/TR/1999/REC-xml-names-19990114</u:uri>
   <t:topics>
    <t:topic name="xml" weight="0.8"/>
    <t:topic name="xmlns" weight="1"/>
   </t:topics>
  </s:misc>
 </b:entries>
</b:bibliography>
```

Figure 5: XML-encoded Example from Figure 1

4.1 Importing References

Importing references requires them to be in the appropriate XML format, and also requires that all BibSchemas referenced in the XML are known to the system. If an XML document references unknown BibSchemas, it is rejected. It is then necessary to install the BibSchema in the system (through a special management interface), after this installation the system will accept the XML document. Before actually importing the references, the XML to be imported is validated against the base schema and any BibSchema(s) it is using. This validation uses code which has been generated from the BibSchema at installation time through XSLT (as shown in Figure 2).

This design makes the system dynamically adaptive to new BibSchemas. Whenever a user wants to manage references with the system that are not supported by the installed BibSchemas, it is possible to write a new BibSchema (e.g., like the `topic` BibSchema shown in Figure 4), install it, and then import data using this BibSchema. Since BibSchema extensions can define arbitrary XML Schema structures for fields, practically all metadata that can be encoded in XML can be managed by our system.

4.2 Exporting References

Exporting references is done by first selecting them through a search interface, and then export them. Since the system's design goal was to adapt to the needs of different users, it supports a very flexible way of exporting references: they can either be referenced in the XML notation shown in Figure 5, or they can be processed by one or more XSLT programs to fit the needs of different users. The system currently supports BibTeX output (by implementing an XSLT generating BibTeX code from BibSchema XML), but it would be trivial to write XSLT code for transforming BibSchema XML to other formats, such as *Dublin Core (DC)* [16], the *Open Archives Initiative (OAI)* [15], or some RDF-based metadata format.

In the same way as BibSchemas must be installed in the system to handle XML using these BibSchemas, XSLTs can be installed to extend the system to generate new output formats. Both types of plugins (BibSchemas an XSLTs) may only be installed by users with special rights, to avoid an uncontrolled growth of BibSchemas and XSLTs in the system.

5 Implementation

The current prototype implementation of the system is based on the popular *LAMP (Linux, Apache, MySQL, PHP)* set of tools (with the exception that it is currently running under Windows, but since there are no Windows-specific parts in it, porting it to another platform such as Linux is trivial). The implementation design is based on the schema design and import and export processes described in the previous section.

Because a BibSchema is used to generate an XML Schema, it is necessary to use an XML Schema processor as part of the import process. After testing a number of XML Schema processors, Apache's Xerces was chosen because it implemented the biggest subset of XML Schema. Sadly, there doesn't seem to be a single XML Schema processor available that implements the full specification (even though there are a number who claim to do so), so finding and picking the right one took some time.

A BibSchema is also used to generate a Schematron schema, which is executed using an XSLT processor. After first using the Sablotron XSLT processor built into PHP, it became clear that it contains some errors, and the implementation now uses the Saxon XSLT processor.

The relational database we are using is MySQL, which is used to store the data after the import checking has been successful. Since MySQL does not support XML, we store fields containing XML as character data. This means that it is possible to search for XML in fields on a text basis, but it is not possible to perform any structure-oriented queries on this content.

6 Further Work

In the following list, we list some of the issues that we think would be interesting to investigate in more detail. Generally, we regard our work as a starting point to think about federated schemas and federated

referatories and ways to support users in such a scenario, but we also believe that more research should be performed in this area.

- *BibSchema Classification and Cataloguing:* Currently no attempt is made to prevent users from reinventing the wheel by defining multiple schemas for the same purpose. The problem of classifying and cataloguing existing schemas, so that users reuse existing schemas and thus increase the usefulness of the referatory, is not an easy one. Our current (admittedly simple) way to deal with this problem is to restrict schema installation to system managers, which are expected to check the schemas manually and thus prevent the emergence of redundant schemas.

- *Import Optimization:* Importing large amounts of records into our system currently is not performing very well, due to our design of the import process (as described in Section 4.1). To speed up the import process, it would be possible to perform various optimizations. In a first step, it would be possible to use compiled XSLT (Schematron validation uses XSLT, which could be pre-compiled). Going further, it would be possible to write custom validation code in a regular programming language. However, since validation must be open to extension by new schemas, this would be a rather complex task.

- *Pluggable Import Filters:* In the same way as the system now supports pluggable export filters, it would be possible to support pluggable import filters, so that users can install such an import filter, and then import any kind of import data[4], which as part of the import process is then transformed into the XML format required for internal storage.

- *XML Database Support:* The system as it is implemented now is based on a relational database, which brings with it all the problems of an XML-based data model stored in a relational database [1]. It would be an interesting task to move the system to an XML database, which in particular would be interesting if the BibSchema extensions included complex XML structures that should be accessible via queries. Which directly leads to the next point:

- *XML Query Language (XQuery):* Currently, the system accepts queries via a Web interface and maps them to SQL queries. This is sufficient as long as queries are targeted at field level. However, if queries need to recognize field structures, then it would be very useful to have a query language specialized for XML queries, such as XQuery [4]. XQuery support would imply an underlying database supporting it, and the question whether this would be worth the effort highly depends on the kind of BibSchema extensions (and in particular, features to query that data) that should be supported. Looking at the current design of the system, this could also involve the following:

- *BibSchema Query Language:* In the same way as the system currently has its own schema language (which is mapped to XML standard technologies), the system could be extended to have its own query language, which would be specifically designed to support BibSchema with its extension mechanism, and possibly XQuery-style queries into BibSchema extensions. Which leads to the last and most complex way to extend the system architecture:

- *Distributed Query Processing:* If both schemas and queries use their own language, then it would be possible to extend the system architecture to support distributed queries. This would take the notion of a federated referatory to a new level, where distributed referatories cooperate to increase

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

the amount of metadata accessible to users. The move towards distributed query processing certainly would be challenging, but it would be possible to benefit from the wealth of knowledge gathered by the federated databases community.

This list is non-exhaustive, but it should be sufficient to illustrate the direction we are envisioning for our system and future research in this area.

7 Conclusions

The system described in this paper is a first approach towards federated referatories. It also is an exercise in using different XML technologies for shortening the development effort required for prototypes. Most of the implementation effort for our system went into choosing and integrating the XML technologies that we used (most notably, DOM, XML Schema, Schematron, and XSLT).

8 Acknowledgements

Philip Schaffhauser and Felix Hauser implemented the prototype for the system described in this paper. A detailed report of their work ins available in their diploma thesis [7].

References

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, San Francisco, California, October 1999.

[2] Maristella Agosti and Constantino Thanos, editors. *Research and Advanced Technology for Digital Technology: Proceedings of the 6th European Conference on Digital Libraries*, volume 2458 of *Lecture Notes in Computer Science*, Rome, Italy, September 2002. Springer-Verlag.

[3] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. World Wide Web Consortium, Recommendation REC-xmlschema-2-20010502, May 2001.

[4] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Working Draft WD-xquery-20030502, May 2003.

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

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

[7] Felix Hauser and Philip Schaffhauser. Database-Driven XML-Enabled Bibliography Management System. Master's thesis, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland, March 2003.

[8] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron. to be published as ISO/IEC 19757-3.

[9] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL). to be published as ISO/IEC 19757.

[10] Anna Keller Gold, Karen S. Baker, Jean-Yves LeMeur, and Kim Baldridge. Building FLOW: Federating Libraries on the Web. In Gary Marchionini, editor, *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 287–288, Portland, Oregon, July 2002. ACM Press.

[11] Oren Patashnik. BibTEXing. Technical report, February 1988.

[12] Oren Patashnik. Designing BibTEX Styles. Technical report, February 1988.

[13] Luca Previtali, Brenno Lurati, and Erik Wilde. BibTEXML: An XML Representation of BibTEX. In *Poster Proceedings of the Tenth International World Wide Web Conference*, pages 64–65, Hong Kong, May 2001. ACM Press.

[14] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. World Wide Web Consortium, Recommendation REC-xmlschema-1-20010502, May 2001.

[15] Herbert Van de Sompel and Carl Lagoze. Notes from the Interoperability Front: A Progress Report on the Open Archives Initiative. In Agosti and Thanos [2], pages 144–157.

[16] Stuart L. Weibel, John A. Kunze, Carl Lagoze, and Misha Wolf. Dublin Core Metadata for Resource Discovery. Internet informational RFC 2413, September 1998.

[17] Ian H. Witten, David Bainbridge, Gordon Paynter, and Stefan Boddie. Importing Documents and Metadata into Digital Libraries: Requirements Analysis and an Extensible Architecture. In Agosti and Thanos [2], pages 390–405.

[18] Jens E. Wolff and Armin B. Cremers. The MyView Project: A Data Warehousing Approach to Personalized Digital Libraries. In Ron Y. Pinter and Shalom Tsur, editors, *Proceedings of Fourth International Workshop on Next Generation Information Technologies and Systems*, volume 1649 of *Lecture Notes in Computer Science*, pages 277–294, Zikhron-Yaakov, Israel, July 1999. Springer-Verlag.