Novi Sad J. Math.
Vol. 34, No. 1, 2004, 107-130

QUALITY CONTROL SYSTEM OF XML BIBLIOGRAPHIC RECORDS

Gordana Budimir[1], **Dušan Surla**[2]

Abstract. The aim of this paper was not only to define the XML bibliographic records format, but also to describe the use of different XML standards and off-the-shelf tools in library automation. The quality control system of bibliographic records within the BISIS library software system has been implemented using XML Schema, XSLT and XPath languages, according to UNIMARC bibliographic format. The paper describes the possibility of using XML in bibliographic record control. The categorization of UNIMARC bibliographic records verifications, that could help to determine record quality, is discussed. Additionally, it describes XML schema for XML format used in the BISIS system. The XML schema and XSLT expressions for additional control of bibliographic records present the basis of XML bibliographic record quality control system. A prototype of this system is implemented in the Java programming environment, independently from bibliographic format, using XML validation, parsing and transformation tools. Thus, it may be used for performing XML bibliographic record quality control, which have different bibliographic formats, using XML schema and XSLT expressions specified in the paper.

AMS Mathematics Subject Classification (2000):

Key words and phrases: XML Schema, XSLT/XPath, UNIMARC bibliographic records, validation, transformation

1. Introduction

Nowadays, XML (*eXtensible Markup Language*) is not only a 'new language on the Web', but is often referred to as a technology in the maturity phase that includes many recommendations and their implementations. In addition, a variety of XML tools are developed that are based on the already standardized APIs. In library information systems, XML has been used since the introduction of XML 1.0 specification [1]. At present, however, the use of XML does not take advantage of all XML possibilities in these systems. An issue that has been discussed most is the relationship between different bibliographic formats and XML formats of bibliographic records, in the sense of defining them most completely by XML language. Although there are intentions of using XML

[1] Institute of Information Science, Maribor, Slovenia
[2] Department of Mathematics and Informatics, Faculty of Science, University of Novi Sad, Trg D. Obradovica 4, Novi Sad 21000, Serbia and Montenegro, E-mail: surla@uns.ns.ac.yu

only as a new bibliographic record format, some recent far-reaching initiatives intend to use the XML in library information systems as an entirely new technology, with all its standards, tools and methodologies. The development of XML-based library information systems includes implementation of different library functions, such as receiving, saving, searching, presenting or exchanging bibliographic data. Hence, a variety of XML specifications are used; for example, XML Schema [2-4] or XSLT (*XSL Transformation*) [5], and XPath (*XML Path Language*) [6] specifications that are parts of XSL (*eXtensible Stylesheet Language*) specification [7].

The BISIS library software system [8] has been developed at the University of Novi Sad since 1993 and several versions have been introduced so far. For version 3.0, a text server has been developed for indexing and searching bibliographic records according to the UNIMARC (*Universal Machine Readable Cataloguing*) [9] bibliographic format. This server has some characteristics that result in better performance: three-tier architecture, the use of the Java platform and independence from relational database system. Support for Unicode (*The Unicode Standard*) [10] is consistently implemented in the whole system of the BISIS Version 3.0. Detailed instructions for the use of this system are given in [8], including 140 references to BISIS system development. Part of the system that refers to user's search is available from [11]. Some of the references [12-15] refer to the future development based on XML.

This paper describes a quality control system of XML bibliographic records according to UNIMARC bibliographic format, that will be a part of the next version of the BISIS system. The prototype of this system is implemented in the Java programming environment and may be incorporated into some other library information systems and library system networks.

2. Bibliographic record control

Quality control of bibliographic records is one of the most important functions of library information systems, as the quality of bibliographic record processing in different library functions depends on it. For example, results of bibliographic data retrieval depend on the data included in bibliographic records (title, author, year of publication, publisher, etc.).

Bibliographic data can be entered into a library system through a user interface, by bibliographic record exchange or by the conversion of bibliographic data from some existing bibliographic database. The quality of received bibliographic records varies; they may contain deviations from the rules of the bibliographic format implemented in the system. Thus, errors occurring in records have to be corrected, to prevent saving syntactically and semantically incorrect records to local library systems. In this way, error correcting leads to increased record quality.

With this in view, it is required to accomplish verifications of records, concerning syntax and semantic rules of a bibliographic format. These verifications make it possible to detect errors in cataloguing of library materials, and warn cataloguers about the missing data or inadequate contents in parts of some records. However, cataloguers will still be responsible for bibliographic record quality, as only experienced cataloguers are able to find all the mistakes in a record. Without their knowledge, errors in a record may be found only if an algorithm for adequate verification is defined.

The first problem that appears is how to define verifications of bibliographic records that can discover most of errors in records and enable the best record control. The second problem is the implementation of a system for bibliographic record control that should capture all of these verifications. In library systems, the problem of bibliographic record control is solved in different ways, depending on the bibliographic format and the system implementation environment. Usually, the implementation of a particular library function includes some programming code for verification of some bibliographic record constraints.

In the BISIS system, the problem of bibliographic record control is generally solved by the quality control system of XML bibliographic records according to the UNIMARC bibliographic format. This control system is implemented as a prototype in the Java programming language, based on XML schema validation and XSLT expressions that perform additional verification of XML bibliographic records. The prototype is implemented independently from the bibliographic format. Thus, it may be used to perform quality control of XML bibliographic records for some other bibliographic format that has the same XML schema validation and XSLT expressions defined.

The following sections describe categorization of verifications of UNIMARC bibliographic records. Also, XML Schema, XSLT and XPath expressions that have been used for modeling record verifications are given. We believe that these concepts are applicable to modeling verifications of bibliographic records for other bibliographic formats, too [16].

3. Systematization of verifications

A bibliographic format is a standard for the representation and exchange of bibliographic and related information in a machine-readable form. The bibliographic record contains three elements: the record structure, the content designation, and the data content of the record. From the aspect of record structure, the data in the bibliographic record are organized into fields, each being identified by a three-character tag, and containing up to two indicators and set of subfields. Content designation represents the codes and conventions established to identify and characterize the data elements that comprise a bibliographic record with sufficient precision to support manipulation of the data for a variety of functions. The content of most data elements is defined by standards outside the formats, e.g., cataloguing rules, like ISBD (*International Standard Bibliographic Description*) [17] or AACR (*Anglo-American Cataloguing Rules*) [18].

Bibliographic record control has to verify the record structure and content, regarding the syntax and semantic rules of a particular bibliographic format and certain cataloguing rules. Thus, verifications of bibliographic records may be divided into two groups (1) structure verifications and (2) content verifications. Both depend on the library material type (monographs, serials, articles, etc.) in the same way as bibliographic record structure and content depend on them. Additionally, these verifications could be divided according to the number of fields, subfields and indicators, into:

- single-element verifications that check the structure or content of one single field, subfield or indicator,
- cross-verifications that check the structure or contents of semantically connected fields, subfields or indicators.

3.1 Single-element verifications

The verification of a single field, subfield or indicator checks its structure or content, independently from the appearance, structure and content of other fields, subfields and indicators. These verifications check the following levels of content designation defined by a bibliographic format and check data content in regard to the following cataloguing rules:

- existence of fields, subfields and indicators in a record,
- appearance of mandatory fields in a record and subfields in a field,
- appearance of repeatable fields in a record and subfields in a field,
- existence of indicator types,
- appearance of secondary fields (fields that appear in subfield `$1` of fields `421, 423` or `469`),
- appearance of coded-data subfields,
- length of subfields, and
- additional controls of subfield content (for example, correctness of ISBN or ISSN numbers).

The single-element verifications for records of particular bibliographic format can be shown in a table that contains descriptions of all fields, subfields and indicators. For example, Table I shows some single- element verifications for UNIMARC bibliographic records.

The existence of a field/subfield is indicated in the column *field/sbf* of Table I with a field/subfield identifier. The column T contains marks of library material types (`M` - monographs, `S` - serials, `C` - collections, `A` - articles, or `N` non-book materials) in which this field/subfield may appear. If a field/subfield may appear in records of all types, column T remains empty.

The existence of an indicator is shown in columns *ind1* and *ind2* of Table I with the types of the first and second indicator (for example, `T01` for the first indicator in the field `500`). Indicator types are marked with `Txy`, where `xy` represents limits of the interval that contains possible values of indicator (for example, type `T01` includes only `0` or `1`). This describes typical indicators. If an indicator has a fixed value, the column contains its value (for example, the fixed value of the second indicator in the field `500` is `0`). If an indicator is not defined in the bibliographic format, the column contains `x`.

Whether a field/subfield is mandatory or not is indicated in the column *M/N* of Table I, where `M` stands for mandatory, or `N` for non-mandatory fields/subfields. Replication of a field/subfield is indicated in the column *R/N,* where `R` stands for field/subfield that may be repeated, while N stands for those that may not. The letters in parentheses mark library material types a characteristic relates to. For example, column *M/N* contains `N(MSN), M(A)` for the field `102`, which means that the field `102` is non-mandatory for monographs, serials and non-book materials, and mandatory for articles.

Subfield length is indicated in the column *F/V* of Table I, where `F` stands for fixed-length subfields and V for variable-length subfields. The fixed length or maximal variable length of subfield content is added in brackets alongside these marks. For example, the column *F/V* contains `F (3)` for the subfield `102$a`, which means that the subfield has fixed-length content of 3 characters.

Content of subfield coded I (Table I, column *C*) belongs to the list of internal codes, whereas the other codes stand for the list of external codes (e.g. 2 denotes country code). The internal list of codes contains coded values for one subfield, while the external list of codes contains coded values for varieties of subfields. For example, for all subfields from field `101`, the column *C* contains number `2`, which represents the external list with language codes.

Additional control of subfield content is indicated in the column *Ctrl* of Table I, with the identification number of the control that checks corresponding correctness of subfield content. Additional controls are numbered and described as in Table II. For example, for the subfield `010$a`, the column *Ctrl* contains number `3`, which represents additional control of ISBN numbers. Secondary field is indicated with number `12` in the column *Ctrl*, which represents additional control for verifying secondary fields.

Table I. Examples of verifications for UNIMARC bibliographic records.

field	*ind1*	*ind2*	*sbf*	*M/N*	*R/N*	*F/V*	*C*	*Ctrl*	*T*
010	x	x		N	R				MCAN
			a	N	N	F(13)		3	MCAN
			b	N	N				MCAN
			d	N	N				M
			z	N	R				MN
101	T02	x		M	N				
			a	M	R	F(3)	2		
			b	N	R	F(3)	2		MAN
			c	N	R	F(3)	2		
			d	N	R	F(3)	2		MSAN
			e	N	R	F(3)	2		MSN
			f	N	R	F(3)	2		MSN

			g	N	N	F(3)	2		MSAN
			h	N	R	F(3)	2		MN
			i	N	R	F(3)	2		MAN
			j	N	R	F(3)	2		MN
102	x	x		N(MSN),M(A)	N				MSAN
			a	N(MSN),M(A)	R	F(3)	3		MSAN
105	x	x		N	R				
			a	N	N				
			c	N	N				
			d	N	N				
			e	N	R				MSCN
500	T01	0		N	R				MN
			a	N	R				MN
			b	N	R				MN
			h	N	R				MN
			i	N	R				MN
			l	N	N				MN
			m	N	N	F(3)	2		MN

Table II. Examples of additional controls for UNIMARC bibliographic records.

control	*description*	
1	control of date format - length 8, form YYYYMMDD, leap-year	
3	control of ISBN - length 13, modulus 11	
7	control of year - length 4, not less then 1000, if not 9999 or ????	
12	`field`	`secondary fields in subfield $1`
	421	`200abcdefghiz, 205abdfg, 206?, 210abcdefgh, 215acde, 225adefhivx, 300?, 500abhilm`
	`423, 469`	`200abhi, 500abhi, 503aj, 700abcdef4, 701abcdef4, 710abcdef4, 711abcdef4`

3.2 Cross-verifications

Cross-verifications check the structure and contents of fields, subfields and indicators that are related to structures or contents of other fields, subfields, indicators, or library material type. For example, cross- verifications may verify if the year of publication in the subfield `100$d` is greater than or equal to the year in the subfield `100$c`. All such constraints are defined by the bibliographic format and certain cataloguing rules. Cross-verifications of bibliographic records may be divided into four groups (Table III).

Table III. Groups of cross-verifications.

group	*cross-verifications*
I	verifications that depend on library material type
II	verifications that depend on appearance order of subfields in a field, and of fields in a record
III	verifications that depend on the structure or contents of other fields, subfields or indicators in the same record
IV	verifications that depend on the structure or contents of other fields, subfields or indicators in upper- level records in the same bibliographic database

Record errors that can be found by means of these verifications may be divided into three levels depending on their importance:

- fatal errors that must be corrected before saving a record (`F`),
- warning errors that may be corrected in some circumstances (`W`),
- information errors that need not be corrected (`I`).

Examples of cross-verifications for UNIMARC bibliographic records are described in Table IV. For each verification, Table IV contains information about verification group from Table III (*group*), the importance level of found error (*level*), list of fields, subfields and indicators that have to be verified (*fields*), formal description of verification algorithm (*verification*) and textual description of verification (*verification description*). For example, third row of Table IV describes cross-verification from group II (Table III) that verifies order of appearance for subfields `$a` and `$c` of field `210`, and that finds error of importance of level W (warning).

Table IV. Examples of cross-verifications for UNIMARC bibliographic records.

group	*level*	*fields*	*verification*	*verification description*
I	F	001$d	article ⇒ t(001$d)='2'	Hierarchical level for articles is 2.
I	F	011$a, 464$1	∃(t(011$a) ∨ t(464$1)) ⇒ article	Subfields 011$a and 464$a appear in records for articles only.
II	W	210ac	∃ (t(210$a)⇒ ∃ t(210$c) before t(210$a)	If subfield $c exists in field 210, subfield a$ has to appear before subfield $c.
III	F	100bc$d	t(100$b)='e' ⇒ t(100$c)> t(100$d)	Year of publication 1 is greater than year of publication 2 in records for reproductions.
III	F	327	∀ t(327 ind1 ind2)= ∀ t(327 ind1 ind2)	All fields 327 must have the same values of indicators.
III	F	700,710	(∃t(700)⇒ ¬t(710))∨ (∃t(7 100⇒ ¬∃ t(700))	Fields 700 and 710 cannot appear in the record in the same time.
IV	F	464$1	record with ID from subfield 464$1is for monograph	Subfield 464$1 appears only as a link to records for monograph publications.

4. Modeling verifications with XML schema

If bibliographic records are represented with XML format, the definition of this XML format in XML schema can be used for bibliographic record validation. XML format definition of bibliographic records can be specified in various ways depending on their use: bibliographic record exchange, bibliographic record validation, or different services - TVS (*Transport, Validation and Services*) [19,20]. For example, MARC 21 format has two definitions for XML records; MARC XML [21] and MODS [22]. The characteristics of bibliographic records that can be checked with XML schema depend on XML element naming and their structure [23]. Bibliographic data should not be merged into single XML element. Each field, subfield and indicator should be represented with an XML element, with a different name and without attributes. That allows validating a bibliographic XML document against specified XML Schema, in accordance with specified rules. In the BISIS system, the format of XML bibliographic records is defined in such a way that the root element is `record`, elements corresponding to fields are `fxxx`, where `xxx` is a field identifier, elements corresponding to subfields are `sfx`, where `x` is a subfield identifier, and elements corresponding to the first and second indicators are `ind1` and `ind2`. Figure 1 shows so-defined XML elements for a title in a UNIMARC bibliographic record.

```
<f200>
  <ind1>1</ind1>
  <sfa>The photosynthetic bacterial reaction center</sfa>
  <sfe>structure and dynamics</sfe>
  <sfe>[proceedings of NATO Advanced Research Workshop on the Structure
    of the Photosynthetic Bacterial Reaction Centre...held September
    20-25, 1987, Cadarache, France]</sfe>
  <sff>edited by Jacques Breton and AndrVermlio</sff>
</f200>
```

Figure 1. Example of a title in the BISIS XML format.

For each library material type specific XML schema incorporating corresponding verification rules described in previous chapters has been defined:

- single-element verifications, except leap year control and control of ISBN and ISSN numbers by modulus 11 (Table II), and
- cross-verifications from I and II group (Table III).

Figure 2 shows a part of the schema definition for XML bibliographic records of monographs. Definitions of XML schema for XML bibliographic records of any library material type can be specified in a similar way.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="CommonTypes.xsd"/>
  <xsd:include schemaLocation="IndicatorTypes.xsd"/>
  <xsd:include schemaLocation="InternalCodes.xsd"/>
  <xsd:include schemaLocation="ExternalCodes.xsd"/>
  <xsd:include schemaLocation="Controls.xsd"/>
  <xsd:include schemaLocation="MonographsBlock9.xsd"/>
  <xsd:element name="record">
    <xsd:complexType>
      <xsd:sequence>
      ...
        <xsd:element name="f010" type="f010Type" minOccurs="0"
          maxOccurs="unbounded"/>
        ...
         <xsd:element name="f101" type="f101Type"/>
         <xsd:element name="f102" type="f102Type" minOccurs="0"/>
        ...
```

```
         <xsd:element name="f500" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="ind1" type="indType01"/>
              <xsd:element name="ind2" type="xsd:byte" fixed="0"/>
              <xsd:element name="sfa" type="xsd:string" minOccurs="0"
                maxOccurs="unbounded"/>
              <xsd:element name="sfb" type="xsd:string" minOccurs="0"
                maxOccurs="unbounded"/>
              <xsd:element name="sfh" type="xsd:string" minOccurs="0"
                maxOccurs="unbounded"/>
              <xsd:element name="sfi" type="xsd:string" minOccurs="0"
                maxOccurs="unbounded"/>
              <xsd:element name="sfl" type="xsd:string"
                minOccurs="0"/>
              <xsd:element name="sfm" type="languageCode"
                minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        ...
        <xsd:group ref="monographsblock9"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="f101aUnique">
      <xsd:selector xpath="f101"/>
      <xsd:field xpath="sfa"/>
    </xsd:unique>
    ...
    <xsd:key name="f327indKey">
      <xsd:selector xpath="f327[position()=1]"/>
      <xsd:field xpath="ind1"/>
      <xsd:field xpath="ind2"/>
    </xsd:key>
    <xsd:keyref name="f327indKeyref" refer="f327indKey">
      <xsd:selector xpath="f327"/>
      <xsd:field xpath="ind1"/>
      <xsd:field xpath="ind2"/>
    </xsd:keyref>
    ...
  </xsd:element>
</xsd:schema>
```

Figure 2. Part of the schema definition for records of monographs.

XML schema is divided into several schema documents for easier maintenance, access control, and readability:

- `CommonTypes.xsd` - contains definitions of common types used in declarations of XML elements, corresponding to fields that can appear in records of different library material types (e.g. `101`), or that have the same structure (e.g. some fields in block 3 contain only subfield `$a`),
- `IndicatorTypes.xsd` - contains definitions of types used in declarations of XML elements corresponding to indicators,
- `InternalCodes.xsd` - contains definitions of types used in declarations of XML elements corresponding to subfields with coded values from the internal list of codes,
- `ExternalCodes.xsd` - contains definitions of types used in declarations of XML elements corresponding to subfields with coded values from external lists of codes,
- `Controls.xsd` - contains definitions of types used in declarations of XML elements corresponding to subfields the contents of which have to be additionally checked, and
- `MonographsBlock9.xsd` - contains a definition of group monographsblock9 that contains declarations of XML elements corresponding to fields from block 9 for national use.

The root element `record` is declared using the `xsd:element` element that is defined as a complex type using the `xsd:complexType` element. This complex type definition contains the `xsd:sequence` element with declarations of `fxx` elements, corresponding to all fields in a record. The `xsd:sequence` element specifies that the elements must appear in the same sequence (order) in which they are declared. It contains the `xsd:group` element with `ref` attribute that refers to the named group `monographsblock9` defined in schema document `MonographsBlock9`. This sequence enables modeling field appearances; their identifiers always appear in the ascending order.

XML schema can indicate that some XML element value must be unique within a certain scope using the xsd:unique element. The nested xsd:select element selects a set of elements and then one or more nested `xsd:field` elements identify the element relative to each selected element that has to be unique within the scope of the set of selected elements. These elements contain xpath attributes with XPath expressions that limit the scope of uniqueness. Note that these expressions are limited to a subset of the full XPath Language specification. For example, Figure 2 shows the xsd:unique element that defines the unique `sfa` element nested in the `f101` element, and thus models uniqueness of the subfield 101$a in the bibliographic record.

The `xsd:key` and `xsd:keyref` elements enable identification of XML elements that must have the same value. The usage of these elements is similar to `xsd:unique` element syntax. It is possible to define combinations of element values that must be equal. For example, Figure 2 shows modeling equality of indicators in all fields 327 (Table IV) with identifying those combinations of `ind1` and `ind2` elements from all `f327` elements that must have the same value. The combination of `ind1` and `ind2` elements in the first `f327` element is defined by the `xsd:key` element, and in other `f327` elements, these combinations are defined by the corresponding `xsd:keyref` element.

Thereby, XML elements that are modeled as keys or as unique are required in an XML document, so the subfields and indicators that correspond to these XML elements must be mandatory in bibliographic records.

4.1 Modeling fields

Bibliographic fields are modeled with XML elements `fxxx`, where `xxx` is a field identifier. These elements are declared using the `xsd:element` element that contains an attribute `name` with an XML element name, an attribute `type` with the name of an XML element type, an attribute `minOccurs` for modeling mandatory fields and an attribute `maxOccurs` for modeling repeatable fields. `fxxx` elements are declared as complex types that may be defined as anonymous complex types (similar to `f500` element in Figure 2), or as named complex types from the included schema document `CommonTypes` (similar to `f010` element in Figure 2). Names of these complex types are in the form `fxxxType,` where `xxx` is a field identifier. For example, Figure 3 shows definitions of named complex types used in declarations for `f010` and `f102` elements.

```
<xsd:complexType name="f010Type">
  <xsd:sequence>
    <xsd:element name="sfa" type=" control3Type" minOccurs="0"/>
    <xsd:element name="sfb" type="xsd:string" minOccurs="0"/>
    <xsd:element name="sfd" type="xsd:string" minOccurs="0"/>
    <xsd:element name="sfz" type="xsd:string" minOccurs="0"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="f102Type">
  <xsd:sequence>
    <xsd:element name="sfa" type="countryCode" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="sfb" type="sf102bType" minOccurs="0
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Figure 3. Definitions of complex types for XML elements corresponding to fields.

The complex type definition contains an `xsd:sequence` element with declarations of `ind1` and `ind2` elements, corresponding to the first and second indicator in a field, and with declarations of `sfx` elements, corresponding to subfields appearing in a field in the same sequence (order) in which they are declared. This element enables modeling of indicators and subfields appearances in a certain sequence in a field.

However, in some fields (e.g. `200, 210`) order of subfield combinations is important and should be in accordance with particular cataloguing rules. Such verifications belong to group II of cross-verifications that are described in Table IV. Modeling of these verifications is performed by using `xsd:group` elements, enabling groups of elements to be defined and named. Elements, corresponding to subfields that can appear in any order in a field, are modeled by using the `xsd:all` element. The following example shows modeling of verification that checks appearance of the subfield `$a` before the subfield `$c` in the `210` field:

```
<xsd:complexType name="f210Type">
  <xsd:sequence>
    <xsd:group ref="groupf2101" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element name="sfd" type="var25Type" minOccurs="0"/>
    <xsd:group ref="groupf2102" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:group name="groupf2101">
  <xsd:all>
    <xsd:element name="sfa" type="xsd:string" minOccurs="0"/>
    <xsd:group ref="groupf210ac" minOccurs="0"/>
    <xsd:element name="sfb" type="xsd:string" minOccurs="0"/>
  </xsd:all>
</xsd:group>
<xsd:group name="groupf210ac">
  <xsd:sequence>
    <xsd:element name="sfa" type="xsd:string">
    <xsd:element name="sfc" type="xsd:string">
  </xsd:sequence>
</xsd:group>
<xsd:group name="groupf2102">
  <xsd:sequence>
    <xsd:element name="sfe" type="xsd:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="sff" type="xsd:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="sfg" type="xsd:string" minOccurs="0"
      maxOccurs="unbounded"/>
    <xsd:element name="sfh" type="xsd:string" minOccurs="0"
```

```
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

Figure 4. Definition of complex type for XML element corresponding to field `210`.

4.2 Modeling indicators

Indicators in a bibliographic field are modeled with XML elements `ind1` and `ind2`. These elements are declared using the `xsd:element` element that contains an attribute `name` with an XML element name, an attribute `type` with the name of an XML element type, an attribute `minOccurs` for modeling mandatory indicators, and an attribute `fixed` for modeling fixed values of indicators (e.g. definition in the declaration of the XML element `f500` in Figure 2). `ind1` and `ind2` elements are declared as enumeration simple types that are derived by restricting the built-in simple type `xsd:byte`. These enumeration simple types are defined in the schema document `IndicatorTypes` by using `xsd:simpeType` element definitions that contain the `xsd:restriction` element with the attribute `base="xsd:byte"`.

To identify the facets that constrain the defined type's range of values, this element contains `xsd:enumeration` elements that limit the simple type to the set of distinct values defined in `value` attributes. Names of these types are in the form `indTypexy,` where `xy` represents limits of the defined type's range of values, corresponding to all possible values of indicators. Definitions of these types enable modeling of indicator type. The following example shows definition of the `indType01` simple type used in modeling of the indicator type `T01` that has possible values `0` and `1`:

```
<xsd:simpleType name="indType01">
    <xsd:restriction base="xsd:byte">
        <xsd:enumeration value="0"/>
        <xsd:enumeration value="1"/>
    </xsd:restriction>
</xsd:simpleType>
```

Figure 5. Definition of simple type for XML element corresponding indicator.

4.3 Modeling subfields

Bibliographic subfields are modeled with XML elements `sfx`, where `x` is a subfield identifier. These elements are declared using the `xsd:element` element that contains an attribute name with an XML element name, an attribute `type` with the name of an XML element type, an attribute `minOccurs` for modeling mandatory subfields, and an attribute `maxOccurs` for modeling repetition subfields. `sfx` elements are usually declared as the built-in simple types (e.g.

sfb element nested in f010 element in Figure 3) or as named simple types from the schema documents InternalCodes, ExternalCodes and Controls (e.g. sfb element nested in f102 element in Figure 3, the type of which is declared to be sf102bType simple type defined in Figure 7). Restriction, extension and union as built-in simple types are defined by xsd:simpleType element, i.e. xsd:restriction, xsd:exstension and xsd:union elements, respectively.

Exceptions to the above are sf1 elements, corresponding to subfields $1 that contain secondary fields (Table II). These elements are declared as complex types, in the same way as sf4211Type complex type is defined in the following example, that shows the f421 element definition:

```
<xsd:element name="f421" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ind2" type="indType01"/>
      <xsd:element name="sf1" type="sf4211Type" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="sf4211Type">
  <xsd:choice>
    <xsd:element name="f200" type="f200Type" minOccurs="0"/>
    <xsd:element name="f205" type="f205Type" minOccurs="0"/>
    <xsd:element name="f206" type="f206Type" minOccurs="0"/>
    <xsd:element name="f210" type="f210Type" minOccurs="0"/>
    <xsd:element name="f215" type="f215Type" minOccurs="0"/>
    <xsd:element name="f225" type="f225Type" minOccurs="0"/>
    <xsd:element name="f300" type="f300Type" minOccurs="0"/>
    <xsd:element name="f500" type="f500Type" minOccurs="0"/>
  </xsd:choice>
</xsd:complexType>
```

Figure 6. Definition of complex type for XML element corresponding to field 421.

The sf1 element contains exactly one of the elements corresponding to the secondary field. This constraint is defined by using the element xsd:choice in a complex type definition. The xsd:choice element contains elements declaration for all possible secondary fields, which are modeled in the same way as all other bibliographic fields, but only one of its subelements is allowed to appear in an instance. Elements, corresponding to subfields with coded values from some list of codes, are declared as named simple types from the included schema documents InternalCodes and ExternalCodes. The following example shows definitions of the simple type sf102bType and the simple type countryCode used in the declaration of such elements.

```
<xsd:simpleType name="sf102bType">
    <xsd:restriction base="xsd:string">
        <xsd:pattern value="fb|rs|cr|ko|sr|vj"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="countryCode">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="abw"/>
        <xsd:enumeration value="afg"/>
            .....
    </xsd:restriction>
</xsd:simpleType>
```

Figure 7. Definitions of simple types for lists of codes.

Names of types used in declarations of elements corresponding to subfields with coded values from internal lists of codes, are in the form `sfdddxType`, where `dddx` is a field and subfield identifier (similar to `sf102bType` simple type in Figure 7). These types are usually derived by restricting the built-in simple types `xsd:string` that are defined by using the `xsd:restriction` element, which constrains the values of the element declared by using the `xsd:pattern` element in conjunction with the regular expression in the attribute `value`. This expression determines all possible values in elements, and is based on the simple type definition. For example, the definition of the type `sf102bType` in Figure 7 gives values `fb, rs, cr, ko, sr` or `vj`.

Names of types used in declarations of elements, corresponding to subfields with coded values from external lists of codes, depend on the meaning of these codes. These types are usually enumerations that are derived by restricting the built-in simple type `xsd:string`. For example, the definition of the type `countryCode` in Figure 7 specifies that all elements whose type is declared to be this simple type, must contain one of country codes defined in value attributes of `xsd:enumaration` elements.

Elements, corresponding to subfields whose contents have to be additionally checked, are declared as named simple types from the included schema document `Controls`. The controls comprise the control of subfield content length (Table I) and additional controls from Table II. The following example shows definitions of the simple type `var25Type` and the simple type `control7Type` used in the declaration of such elements.

```
<xsd:simpleType name="var25Type">
    <xsd:restriction base="xsd:string">
        <xsd:maxLength value="25"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="control7Type">
```

```
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:gYear">
        <xsd:minInclusive value="1000"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="(\d|[?]){4}"/>
    </xsd:restriction>
   </xsd:simpleType>
 </xsd:union>
</xsd:simpleType>
```

Figure 8. Definitions of simple types for content control.

Names of such types depend on content length of corresponding subfields. Thus, the simple type `fixXType` defines string content with fixed length `X`, whereas the simple type `varXType`, defines string content with maximal length `X` (as the `var25Type` simple type). In the same manner, simple types whose names are in the forms `intXType`, `intmaxXType` or `decmaxXType` define intervals of integer or decimal numbers that the corresponding subfield can contain. All these types represent restriction of the built-in type `xsd:string` or `xsd:positiveInteger`, which has the subelements that determine more specifically the length of the content, defined by using the `xsd:restriction` element that constrains the values of the declared element using `xsd:pattern`, `xsd:length`, `xsd:maxLength`, `xsd:totalDigits`, or `xsd:maxInclusive`.

Names of types used in declarations of elements corresponding to the subfields whose content must be additionally checked by some control from Table II are given by `controlXType`, where `X` is the identification number of the control. Restriction or union of built-in simple types can be specified by using the `xsd:simpeType` element that contains `xsd:restriction` and `xsd:union` elements. For example, the definition of the type `control7Type` is used in declarations of elements corresponding to subfields that have to be checked according to control 7 from Table II. According to control 3 from Table II for the verification of ISBN numbers the `control3Type` simple type is defined on the basis of the simple type defined in [24], already defined and used, for example, in the declaration of the element corresponding to the subfield `010$a` (Figure 3).

5. Modeling verifications with XSLT

As it has been shown, the majority of bibliographic record characteristics can be specified with the appropriate XML schema. However, XML schema cannot capture some specific dependences that may exist in bibliographic records. For example, the constraint which determines that the year of publication in the `sfd` element, corresponding to the subfield `100$d`, must be greater than or equal to the year of publication in the `sfc` element, corresponding to the subfield `100$c` (Table IV).

For checking such constraints we can choose one of the following options [25]: to use XML schema constraint language (Schematron [26]), to develop some programming codes that check these constraints (in Java, Perl, C++, etc.), or to write a stylesheet to check the constraints with XSLT and XPath languages. Considering advantages and disadvantages of these options, we decided to use XSLT and XPath languages for modeling most of bibliographic record verifications that cannot be modeled with XML schema. For other constraints that cannot be captured by XSLT and Xpath, programming code in Java should be developed. However, for UNIMARC bibliographic records, all specific categorized verifications can be modeled with XSLT expressions:

- verification of leap year and ISBN and ISSN numbers by modulus 11 (Table II), and
- cross-verifications from group III and IV (Table III).

When an error occurs, the text with record identification (`ID`), the level of error importance (`FATAL`, `WARNING`, `INFORMATION`) and error description will be shown in output, as in the following example:

```
ID=123456
FATAL - Subfields 100bcd: for reproductions (100b='e') year of
publication 2 must be greater or equal to the year of publication 1.
```

The XSLT stylesheet for XML bibliographic records is defined with the `xsl:template` element that transforms the root element `record` into the result tree that contains text nodes with descriptions of errors in an XML bibliographic record. Figure 9 shows a part of this template definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="txt"/>
<xsl:template match="record">
  <xsl:variable name="ID" select="f000/sfx"/>
  <xsl:text>&#10;&#10;ID=</xsl:text>
  <xsl:value-of select="$ID"/>
  <xsl:variable name="sf100b" select="f100/sfb"/>
  <xsl:variable name="f436ind2" select="f436[1]/ind2"/>
  <xsl:variable name="f700" select="f700"/>
  <xsl:variable name="f710" select="f710"/>
```

```
  ...
  <xsl:if test="string-length(./f071) > 0 and not($sf001b='c' or
    $sf001b='i' or $sf001b='j')">
    <xsl:text>&#10;WARNING - Field 071: field 071 can be used for
      printed music scores and sound recordings
      (001b='c','i','j').</xsl:text>
  </xsl:if>
  ...
  <xsl:choose>
      <xsl:when test="$sf100b='e' and not(number($sf100c) &lt;
      number($sf100d))">
  <xsl:text>&#10;FATAL - Subfields 100bcd: for reproductions
    (100b='e') year of publication 2 must be greater or equal to the
     year of publication 1. </xsl:text>
    </xsl:when>
      ...
  </xsl:choose>
  ...
  <xsl:for-each select="f101/sfa">
    <xsl:variable name="sf101a" select="."/>
    <xsl:for-each select="../sfb|../sfc">
      <xsl:if test="$sf101a=.">
        <xsl:text>&#10;WARNING - Subfields 101abc: languages codes in
          101bc cannot be equal to the language code in
          101a.</xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:for-each>
  ...
  <xsl:for-each select="f436[position()>1]">
    <xsl:if test="$f436ind2 != ./ind2">
      <xsl:text>&#10;WARNING - Field 436: all 436 fields mast have
        the same value of second indicator.</xsl:text>
      </xsl:if>
  </xsl:for-each>
  ...
  <xsl:if test="string-length($f700) > 0 and string-length($f710) >
    0">
      <xsl:text>&#10;FATAL - Field 700,710: 700 and 710 fields cannot
      appear in the same time.</xsl:text>
  </xsl:if>
...
</xsl:stylesheet>
```

Figure 9. Part of XSLT template for root XML element `record`.

The `xsl:output` element outputs the result tree by producing output containing string-value of every text node in the result tree in the document order. Modeling verifications of XML elements is performed with the `xsl:if` and `xsl:when` elements, nested in the `xsl:choose` element, with the test attribute. This attribute specifies `XPath` expressions that define conditions on which a text node is to be generated by the nested `xsl:text` element; the node contains descriptions of errors, and will be inserted into the result tree.

`xsl:variable` elements are used to bind variables and have the required attribute `name`, which specifies the name of a variable. Variable values can be inserted into the result tree as text nodes with `xsl:value-of` elements. References to these variables in `select` and `test` attributes are in the form `$var`, where `var` is a variable name. For example, `$sf100b` refers to the `sf100b` variable that contains the value of the XML element corresponding to the subfield `100$b`.

Repeatable fields and subfields are modeled with repeatable XML elements. Sometimes, all of these repeatable XML elements must be checked. This can be defined with the `xsl:for-each` element that contains a template, which is instantiated for each node corresponding to repeatable XML elements, selected by the XPath expression and specified by the `select` attribute. For example, Figure 9 shows verification of values for all `sfa` elements nested in `f101` elements, corresponding to all subfields `101$a`. Or, for example, selection of all repeatable fields `436`, except for the first one, is defined with the expression `f436[position()>1]` in Figure 9.

An `xsl:template` element with a `name` attribute specifies a named template. An `xsl:call- template` element invokes a template by name; it has a required name attribute that identifies the template to be invoked. `xsl:param` elements are allowed as subelements at the beginning of an `xsl:template` element. Parameters are passed to templates using the `xsl:with-param` element. The required name attribute specifies the name of the parameter. For example, the named template `date`, with two arguments (`content` and `name`) is defined to control the leap year in some subfields. The control of ISBN and ISSN numbers by modulus 11 is modeled with specific templates based on the template [27] that has already been defined. In this definition, mathematical functions `sum`, `dev`, `mod` and `floor`, operations + and *, and recursion of some template are used to calculate verification numbers by modulus 11.

Articles and other component parts are connected with upper-level bibliographic records for serial or monograph publications through subfields `011$a` and `464$1`. Group IV of cross-verifications (Table IV) describes the constraints that have to be checked. After verification of all XML elements according to XML elements from the same XML document, verification of some XML elements according to an XML element from an XML document that contains upper-level records has to be performed. For example, the XML document base.xml contains the collection element with nested record elements corresponding to upper-level bibliographic records. Elements from this XML document can be accessed through the base variable that is defined as presented in Figure 10.

```
<xsl:variable name="base"
  select="document('base.xml')/collection/record"/>
<xsl:for-each select="$base">
  <xsl:if test="./f000/sfx=$sf4641">
  ...
  </xsl:if>
</xsl:for-each>
```

Figure 10. Part of XSLT template for accesing elements from XML document.

The document('base.xml') function passes all record elements from the base.xml document into the nodes of the input tree for XSLT transformations. These nodes can be transformed in a similar way as other nodes in the input tree by referring to the base variable.

Controls and listings presented in this paper can be modeled using some of the existing software tools (e.g. XMLSpy editor).

6. System implementation

The quality control system for UNIMARC bibliographic records based on specific XML schema and XSLT expressions has been developed within BISIS library software system. The control system prototype is implemented in the Java programming environment using the following tools: for validation - MSV (*Sun Multi-Schema XML Validator*) [28], for parsing - Xerces [29], and for transformation - Xalan [30]. The characteristics of these tools are:

- they are in the open-source domain,
- they implement standard interfaces: JARV (*Java API for RELAX Verifier*) [31] for validation, TrAX (*Transformation API for XML*) [32] for transformation and JAXP (*Java API for XML Processing 1.2*) [33] for parsing XML documents,
- they support specifications of XML Schema, XSLT, XPath, DOM [34] and SAX [35],
- MSV enables schema caching that makes validation of XML documents faster, as schema is parsing into the memory only once,
- MSV is fail-fast and reports all validation errors at once, unlike for example, XMLSpy editor [36] that reports only the first validation error,
- Xerces enables caching of XSLT specification,
- MSV and Xerces are thread-safe for useing saved XML schema and XSLT specification, which is important in developing Web Services for quality control of XML bibliographic records.

Quality control of XML bibliographic records performs as follows. Firstly, specified XML schema and XSLT expression should be parsed and saved. Secondly, records should be validated and transformed according to saved specifications. Validation messages and the text that is the result of XML record transformation are stored in an output file that is followed by error processing in XML records. GUI (*Graphical User Interface*) is simple, and is shown in Figure 11.

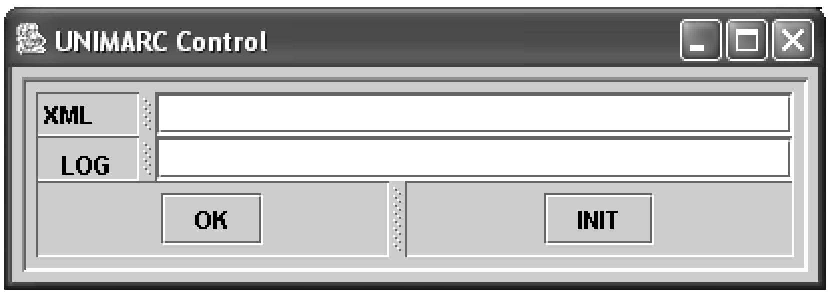

Figure 11. GUI for quality control system of XML bibliographic records.

The text field labeled with *XML* contains the name of the input file that holds XML bibliographic records. The text field labeled with *LOG* contains the name of the output file that contains error messages after quality control is performed. *OK* button starts control. If some other bibliographic format is processed, the text field labeled with *XML* contains the names of files that hold the corresponding XML schema and XSLT expressions. In this case *INIT* button submits initialization of new quality control system for bibliographic records according to this bibliographic format. Further controls of XML bibliographic records will be performed according to the new specifications. Therefore, it is not necessary to restart the application.

Complete listing of XML Schema and XSLT language, as well as the application itself can be obtained electronically from the authors on request.

7. Conclusion

Control of bibliographic record quality encompasses control of their structure and contents. The records can be mapped into XML documents. In the proposed system the control of quality of XML bibliographic records is carried out using XML schema and XSLT languages. The major part of the controls can be described using XML schema language, and the rest require the application of an appropriate algorithm (procedure), and they are described by XSLT language. Based on this, a prototype of the system for controlling quality of bibliographic records is implemented in the Java programming environment according to UNIMARC format. The controls have been proved and used in the BISIS library software system.

The advantage of so-defined quality control system is its independence from bibliographic format. Thus, implemented prototype may be used for bibliographic record control of some other bibliographic format for which equivalent XML schema and XSLT specification are defined. Concepts presented in this paper may be used in modeling these XML specifications for validation and transformation of XML bibliographic records for some other bibliographic format.

The prototype could be expanded with more sophisticated GUI or with a different organisation of error messages. Onto this prototype, additional library system functions may be developed, for example, XML editor of bibliographic records that does not require user knowledge of XML, or some Web service for bibliographic record control.

Acknowledgments. This paper is part of the research project 'XML Technologies and Cooperative Information Systems', supported by the Ministry of Science and Environmental Protection of the Republic of Serbia (Project No. 1875). The authors are indebted to one of the referees who performed the painstaking task of correcting the English of the paper.

References

[1] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Franis, Y., Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation 2004. `http://www.w3.org/TR/2004/REC-xml-20040204/` [4 February 2004].

[2] Fallside, D. C., XML Schema Part 0: Primer. W3C Recommendation 2001. `http://www.w3.org/TR/xmlschema-0/` [20 January 2002].

[3] Tomphson, H. S., Beech D., Maloney, M., Mendelsohn, N., XML Schema Part 1: Structures. W3C Recommendation 2001. `http://www.w3.org/TR/xmlschema-1/` [5 February 2002].

[4] Biron, P. V., Malhotra, A., XML Schema Part 2: Datatyps. W3C Recommendation 2001. `http://www.w3.org/TR/xmlschema-2/` [5 February 2002].

[5] Clark, J., editor, XSL Transformations (XSLT), Version 1.0. W3C Recommendation 1999. `http://www.w3c.org/TR/xslt` [2 September 2002].

[6] Clark, J., DeRose, S., editors, XML Path Language. W3C Recommendation 1999. `http://www.w3.org/TR/xpath` [2 September 2002].

[7] Adler, S., Berglund, A. Caruso, J., Deach, S., Graham, T., Grosso, P., Gutentag, E., Milowski, A., Parnell, S., Richman, J., Zilles, S., Extensible Stylesheet Language (XSL), Version 1.0. W3C Recommendation 2001.// `http://www.w3c.org/TR/xsl` [20 September 2002].

[8] Surla, D., Konjovic, Z., et al., Instruction Manual for Library Software System BISIS vesion 3. Group for Information Technologies, Novi Sad, 2003 (in Serbian).

[9] UNIMARC Manual: bibliographic.format / International Federation of Library Associations and Institutions. IFLA Universal Bibliographic Control and International MARC Programme, New Providence, London, 1994.

[10] The Unicode Consortium. The Unicode Standard, Version 4.0 . Boston, MA, Addison-Wesley, 2003. ISBN 0- 321-18578-1. http://www.unicode.org/versions/Unicode4.0.0/ [16 January 2004].

[11] Library information system BISIS, http://libsrv.im.ns.ac.yu [2 February 2004].

[12] Zeremski, M., Modeling of UNIMARC Format in XML Technology, Mater Thesis, Faculty of Sciences, Novi Sad, 2002 (in Serbian), http://diglib.ns.ac.yu/ndltd/docs/set1/ndltd64/ZeremskiMMagistarskaTeza.pdf [20 December 2002].

[13] Mijic, V., XML Editor for UNIMARC format description, Master Thesis, Faculty of Sciences, Novi Sad, 2003 (in Serbian), http://diglib.ns.ac.yu/ndltd/docs/set1/ndltd133/MijicVMagistarskaTeza.pdf [3 September 2003].

[14] Vidakovic, J., Modeling and Implementation of Bibliographical Cataloguing Cards in XML Technology, Master Thesis, Faculty of Sciences, Novi Sad, 2003 (in Serbian), http://diglib.ns.ac.yu/ndltd/docs/set2/ndltd335/JovanaMagistarski.pdf [25 November 2003].

[15] Milosavljevic, B., Extensible Multimedia Information Retrieval System, Ph.D. Thesis, Faculty of Technical Sciences, Novi Sad, 2003 (in Serbian), http://diglib.ns.ac.yu/ndltd/docs/set1/ndltd143/xmirs.pdf [30 May 2003].

[16] Budimir, G., Quality Control of XML Bibliographic Records, Master Thesis, Faculty of Sciences, Novi Sad, 2004 (in Serbian), http://diglib.ns.ac.yu/ndltd/docs/set3/ndltd409/kontrola_kvaliteta.pdf [15 June 2004]

[17] IFLA, Family of ISBDs : Publication list. International Federation of Library Associations and Institutions, http://www.ifla.org/VI/3/nd1/isbdlist.htm [30 September 2003].

[18] Delsey, T., The Logical Structure of the Anglo-American Cataloguing Rules Part I, The Joint Steering Committee for Revision of AACR 1998. http://webdoc.gwdg.de/ebook/aw/1999/jsc/aacr.pdf [8 March 2002].

[19] TVS Model, BookMARC, Servis de Informao Bibliogrica, Lda, http://www.bookmarc.pt/tvs/ [14 January 2003].

[20] Carvalho, J., Cordeiro, M. I., XML and bibliographic data: the TVS (Transport, Validation, Services) model, IFLA Council and General Conference, vol. 68, 2002. http://www.ifla.org/IV/ifla68/papers/075-095e.pdf [30 September 2002].

[21] LOC. MARC 21 XML Schema. The Library of Congress, http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd [18 May 2004].

[22] LOC. Metadata Object Description Schema (MODS), The Library of Congress, http://www.loc.gov/standards/mods/ [12 December 2003].

[23] Budimir, G., MARC records and XML. INFOTHECA, Journal of Informatics and Librarianship, 2004; 4(1-2). http://www.unilib.bg.ac.yu/bibliotekarstvo/infoteka/1_2-2004/MARCXML%20fin%20(2).pdf [18 Jun 2004].

[24] Costello, R., Sperberg, S., XML Schema simpleType Definition of an ISBN, Xfront 2004. http://www.xfront.com/isbn.xsd [30 January 2004].

[25] Costello, R., et al., XML Schemas: Best Practices, xFront 2003, http://www.xfront.com/BestPracticesHomepage.html [17 February 2003].

[26] Jelliffe, R., The Schematron: An XML Structure Validation Language using Patterns in Trees. Academia Sinica Computing Centre, Taibei, 1999-2001, http://www.ascc.net/xml/resource/schematron/schematron.html#overview [2 February 2002].

[27] Costello, R., Sperberg, S., XSLT-validation of an ISBN, Xfront, http://www.xfront.com/isbn.xsl [30 June 2004].

[28] Kawaguchi, K., Sun Multi-Schema XML Validator, Sun Microsystems, Inc. 1994-2004. http://www.sun.com/software/xml/developers/multischema/ [30 April 2003].

[29] Xerces, The Apache Software Foundation, http://xml.apache.org/xerces-j/ [11 May 2002].

[30] Xalan, The Apache Software Foundation, http://xml.apache.org/xalan-j/ [30 November 2003].

[31] Kawaguchi, K., JARV User's Guide, http://iso-relax.sourceforge.net/JARV/JARV.html [15 Jan 2003].

[32] Java API for XML Processing (JAXP), Sun Microsystems, Inc. http://java.sun.com/xml/jaxp/ [24 May 2004].

[33] Transformation API for XML, The Apache Software Foundation, http://xml.apache.org/xalan-j/trax.html [15 December 2003].

[34] Le Hors, A., Le Haret, P., Wood, L., Nicol, G., Robie, J., Champion, M., Byrne, S., editors, Document Object Model (DOM) Level 2 Core Specification, Version 1.0. W3C Recommendation, 2000. http://www.w3.org/TR/DOM-Level-2-Core/ [10 January 2002].

[35] Simple API for XML (SAX), http://www.saxproject.org/ [24 May 2004].

[36] XMLSpy, Altova, http://www.xmlspy.com [30 January 2004].

Received by the editors September 16, 2004