Novi Sad J. Math.
Vol. 37, No. 2, 2007, 59-72

IMPLEMENTATION OF AN XML-BASED SYSTEM FOR CONTENT GENERATION AND LIBRARY CATALOGUE CARDS DISPLAY

Jovana Vidaković, Miloš Racković[1]

Abstract. This paper presents the use of the XML technology in modelling library documents, i.e. catalogue cards as a kind of reports found in a library information system, as well as the way of forming schemas for content of various types of library catalogue cards. Display of catalogue cards has been done based on the described schemas. The complete specification of the system for content generation and library catalogue cards display is given in the Unified Modelling Language (UML). Implementation is done in the Java programming language. The result of this program is an HTML document that represents a catalogue card that can be shown in a browser.

AMS Mathematics Subject Classification (2000): 68P15, 68P05

Key words and phrases: XML, XML Schema Language, catalogue cards, display

1. Introduction

One kind of reports in librarianship comprises reports about bibliographical material relating to reports on processed publications in the local or remote bibliographical material databases. They include catalogue cards, among other things. Catalogue cards are reports describing objects in a library collection. A catalogue card contains concepts. A concept contains fields, fields contain subfields and subfields can contain subsubfields or values.

The ISBD standard (International Standard Bibliographic Description) is used for defining the content and appearance of catalogue cards. The ISBD standard [1] is an international standard for bibliographical description and it states exactly the order of bibliographical data, the application of a particular punctuation system and data retrieval from predetermined sources. According to the ISBD, standard elements of a bibliographical description are grouped in the areas whose order, content, language and alphabet are exactly defined. A catalogue card is produced using the ISBD principles.

The XML technology is suitable for using in library information systems since library documents are structured and XML documents can be used to model them. Library data are converted into XML documents, as described in several projects. The conversion of MARC records to XML documents has

[1]Department of Mathematics and Informatics, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia, e-mail: {jovana, rackovic}@im.ns.ac.yu

been done in the Java programming language, and it has been described in the Medline Project of Stanford University Medical Center [2].

Validation of XML documents is done with the help of Document Type Definition (DTD). The USA Library of Congress has developed the MARC XML project which features manipulation of MARC data in the XML environment [3].

Conversion between MARC and MARC XML, published by the Library of Congress, is done using MARC4J. MARC4J is the library for working with MARC records in Java [4]. It is easy to write any kind of Java application or servlet including MARC or MARC XML data using MARC4J.

At the University of Buffalo, State University of New York, MARC records were converted to XML catalogue pages containing bibliographical information, holdings information, and some others [5]. Using an XSL stylesheet with properly tagged XML files, all of the data in the MARC records were preserved and control of the appearance of the page was gained. Citation elements of interest to the general user, such as author, title, call number and subject, were displayed and labelled.

In [7] the schema of the UNIMARC format [6] and the bibliographical record schema have been described, as well as bibliographical records validation.

Currently available references do not give an insight into the way of modelling library reports. If reports are accessible, only the final result of the database search of a library information system is visible, i.e. the display of data about the publication searched for.

The development of the library information system BISIS started at the University of Novi Sad [8] in 1993. Bibliographical records in the BISIS are saved in the YUMARC format, which represents the modification of the international UNIMARC format. The YUMARC format extends UNIMARC format with the block of fields for the national usage. According to YUMARC, a bibliographical record contains fields, fields contain subfields and subfields can contain subsubfields or values. Subsubfields have the similar structure to subfields, and can be found in the block for the national usage – block 9. Subfields in this block can contain subsubfields instead of data.

A user searches the database by querying a particular field or fields, for example by the author or book title, and gets a record, in the YUMARC format, as the result. In the current version of the BISIS library reports are generated using FreeMarker templates. In the thesis [12] generation of library cataloguing cards by creating templates in the FreeMarker software package is given. FreeMarker is an open-source template engine. It generates output text based on the templates and appropriate data model. The data model can be a Java object and output text can be HTML. Templates are written in FTL (FreeMarker template language) which is an XML-like template language. The formed templates are used for adding new cards and concepts.

A detailed description of the system for generating and displaying library catalogue cards is given in [9, 11]. This solution has been verified using the library information system BISIS. This paper represents a solution for modelling library catalogue cards using the XML Schema language, as well as the

implementation of the system for content generation and library catalogue cards display. Some concepts of different types of catalogue cards are described by the schema (Section 2), and then the particular types of catalogue cards are described by separate schemas (Section 3). In view of the described procedure, it is possible to extend the document by new concepts, and it is also possible to model other types of library documents.

This paper also presents two programs written in the Java programming language (as part of Section 4). The first one extracts content out of an XML library record following rules of the described schemas, forms a new XML document which is then displayed as HTML by the second program. The complete specification of the system is given in the Unified Modelling Language. Class and sequence diagrams of these programs are given. Implementation is done in the Java programming language.

2. Specification of the Library Catalogue Card Concepts Using XML Schema Language

During analysis of catalogue card concepts it has been noted that there are concepts recurring on various types of catalogue cards [9, 10]. Catalogue card concepts have been modelled by a single schema, i.e. content of each concept is described by citing the fields and subfields it is made up of. This schema contains description of concepts, fields and subfields. Punctuation is also given, and particular catalogue card types are modelled by schemas invoking corresponding concepts by citing their names.

The schema for catalogue card concepts description is given in detail in [9].

One of the concepts described in the schema is the headword for PhD theses. In the schema it is represented by the *HeadwordPhD* element. This concept is given in Figure 1. It is modelled by complex structure, i.e. a sequence of elements representing fields being contained by the headword for a PhD thesis. The rule for processing is *aggregation*, because the fields are processed in the given order and it is possible that the bibliographic record does not contain several subfields which are cited in the schema. The punctuation for this concept is *nextLine*, because the concept that is following the *HeadwordPhD* concept should be printed in the next line. The headword is written in bold and the attribute font has the same value. Subfields 700a and 700b contain the first and the last name of the person who has written the PhD thesis. Subfield 328f contains the field of the PhD thesis.

The remaining concepts are modelled in much the same way as described in detail in [9].

3. Specification of the XML Schema for Describing Particular Catalogue Cards

By describing appearance and content of concepts in a schema, it is possible to define all card types using these concepts. When defining this kind

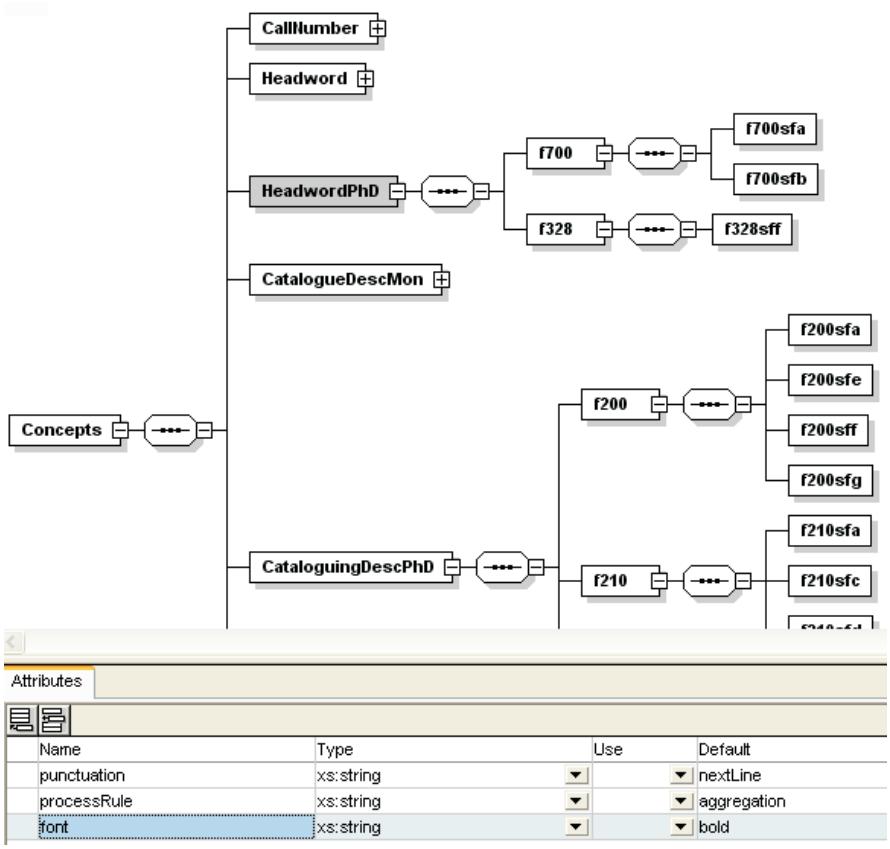

Figure 1: The *HeadwordPhD* concept

of documents, only concept names are given without providing any detail of the content, display form and punctuation of the field, subfield or subsubfield present in the mentioned concept, as was described in [9].

The catalogue card to be described in this paper is the catalogue card for a PhD thesis. This catalogue card type consists of the following concepts: call number, headword, main description for the PhD thesis, remarks, supervisor, board, UDC number and inventory number. The schema modelled for this catalogue card type is shown in Figure 2 in the Schema Design view of the XML Spy Editor.

It is possible to model the XML Schema for an arbitrary type of catalogue card consisting of concepts specified in the schema for catalogue card concepts description in a similar way.

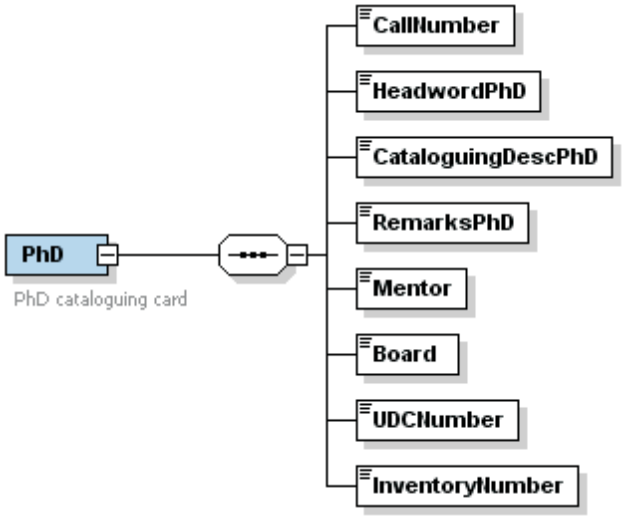

Figure 2: Catalogue card for a PhD thesis

4. Implementation of the Content Generating and Library Catalogue Cards Display

Catalogue cards are displayed with the help of two programs written in Java. The first program extracts content from an XML bibliographical record based on the schema describing catalogue card concepts and the schema describing a particular type of catalogue card. The program first traverses the structure of the schema describing the particular type of catalogue card and finds names of the concepts the card consists of. After that the program looks for these concepts in the schema for describing catalogue card concepts. It looks for the list of fields, subfields and subsubfields of the concept. Afterwards, the cited fields and subfields, i.e. subsubfields, are searched for in the record, and their content is extracted. The punctuation found in the schema describing the catalogue card concepts appends to the record content. The content with the punctuation is placed into the new XML document whose tags have names of the concepts. Beside the content, all data important for display, such as font and justification, are placed into that XML document.

The class diagram of the first program is shown in Figure 3. The *FirstPass* class is the starting class in the first program and it uses methods from the classes *XMLUtils* and *MyXMLParser*. The *XMLUtils* class is realized in the BISIS, and is used to work with XML documents. The *MyXMLParser* class transforms an input XML file into a DOM (Document Object Model) tree.

The *FirstPass* class uses the methods of the *ConceptProcess* class which processes concepts. The attributes of the concept are being found, and subsequently, so are the fields that the concept consists of. The names of the fields

are stored in a list for further processing. All concepts, except for *ArabicTracing* and *RomanTracing* concepts are processed further using the methods of the *FirstLevelProcess* class. Based on the list of field names from concepts, the appropriate fields in the record are being searched for. The subfields of the found field are also put into the special list, because it is important to know if a subfield has already appeared in the record in order to determine its punctuation. If the punctuation of a subfield depends on the appearance of another subfield in the record, the conditional punctuation is applied by processing the corresponding conditional expression. The concepts *ArabicTracing* and *RomanTracing* are enclosed in separate classes as their structure is being modelled in a different way from other concepts.

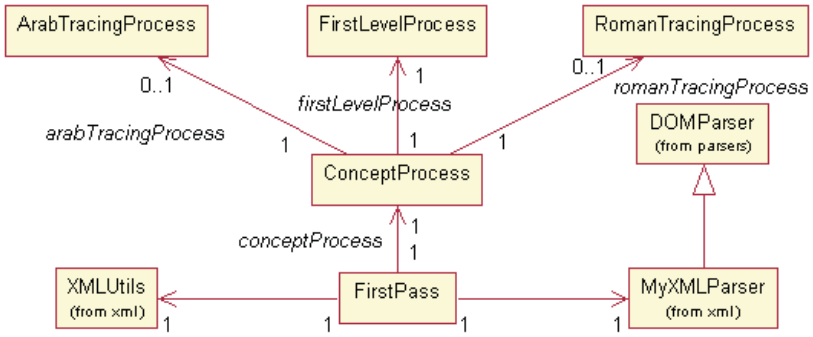

Figure 3: Class diagram of the first program

There are three documents in the program input. They are the catalogue cards concepts description schema, schema for describing particular catalogue cards and XML bibliographical record. In the constructor of the *FirstPass* class all these schema documents are transformed into DOM (Document Object Model) structure using the parser from the XML package (the method *doParse*). In this constructor, an instance of the *ConceptProcess* class is also created. It will be used for the catalogue cards concepts processing. The *makeFirstPass* method is then invoked. It creates the header of the XML document which is the result of the first pass of the program. The process starts from the schema for describing a particular catalogue card. The first tag is *xsd:sequence*, and its child elements are tags whose names are card concepts names. They are placed into the element list (*NodeList*). The *cardElementProcessing* method is invoked. It processes the list of elements whose names are concept names. After that, for each element in the concept schema, the element with the same name is being found. When the concept is found, its name is written in the final document. Before creating the closing tag, the *conceptProcessing* method from the *ConceptProcess* class is invoked for each element. This method finds all attributes of a certain concept and invokes an appropriate method. The sequence diagram of the first program is given in Figure 4.

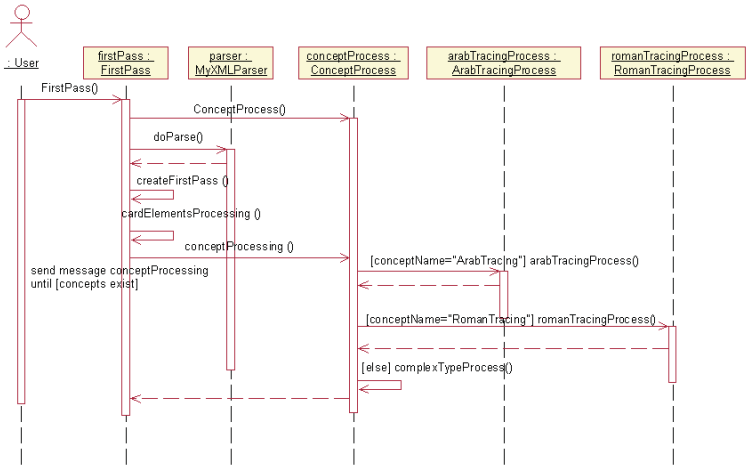

Figure 4: Sequence diagram of the first program

The *firstLevelProcess* method iterates through the field set and for each field in the set it looks for the field with the same name in the record. The sequence diagram for this part of the program is given in Figure 5. For each field in the set, its subfields are being found, and the complete list of subfields of that field is created (the *makeSubfieldList* method). If the subfield present in the concept subfields list is found, its punctuation given in the concept schema should be found too (the *getPunctuation* method). The first punctuation that is being searched for is the conditional punctuation (the *punctuationCond* attribute). If the subfield has already occurred in the concept, the repetitive punctuation is being looked for. If neither of these punctuations is found, then the regular punctuation is searched for and put in front of the subfield content (the *punctuation* attribute).

The punctuation which is placed before subfield content is retrieved from the *getPunctuation* method. The conditional punctuation is searched for first. If it is found, the parser for conditional punctuation processing (the *PunctuationParser* class) is invoked. As the result, it gives the command, i.e. conditional expression, that is processed further. Figure 6 shows the sequence diagram of the part of the program which is run if the subfield has conditional punctuation. The *parse* method checks the string and if it recognises it as one of the built-in commands which model conditional punctuation, it returns the object of one of the corresponding classes. Depending on the conditional expression in the element xsd:attribute default attribute (whose *name* attribute has the value *punctuationCond*) in the concept schema, the appropriate class to process that conditional expression (for example, *IfBefore*, *IfNotBefore*, *IfAfter*, *IfExists*, *IfSecFieldExists*) is invoked. All these classes implement the same interface *IntComm* and its execute method. The *execute* method returns the punctuation as a string. The class diagram for conditional punctuation is given in Figure 7.

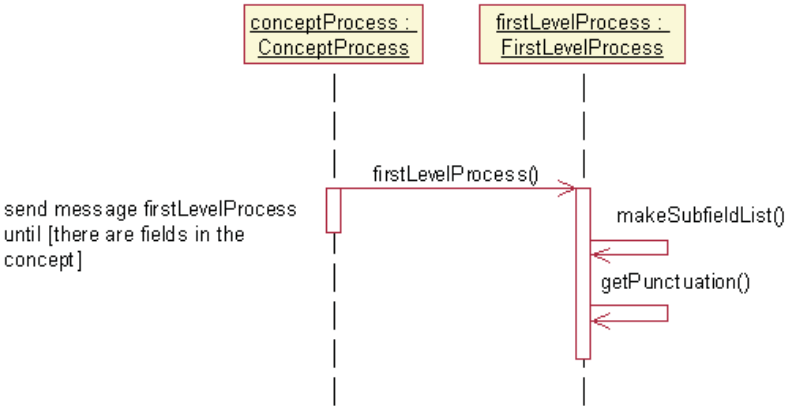

Figure 5: Sequence diagram of matching fields in the record and in the concept

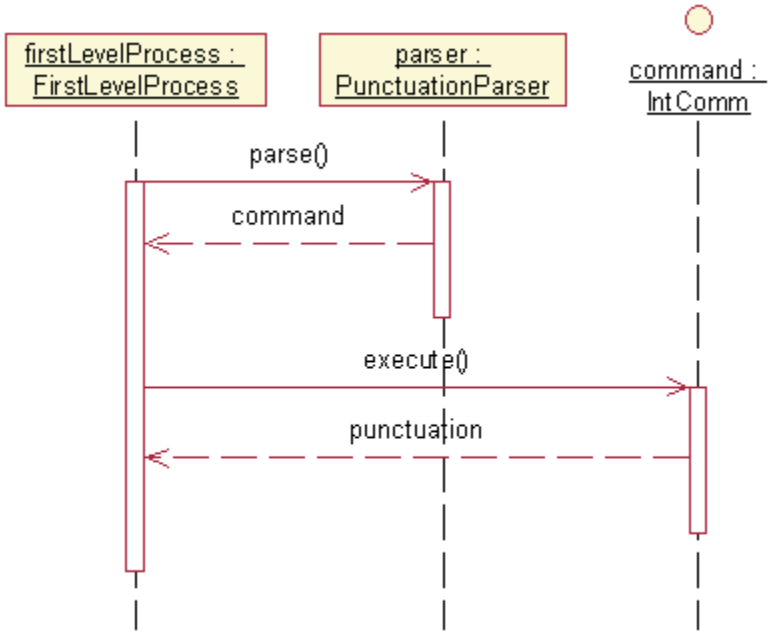

Figure 6: Sequence diagram for processing conditional punctuation in case the subfield has the conditional punctuation defined

When the repetitive punctuation is looked for, the *firstLevelProcess* method checks if the subfield, that should be processed has already occurred in the list

of subfields. If that condition is satisfied, the punctuation from the element xsd:attribute *default* attribute (whose *name* attribute has the value *punctuationRep*) is appended to the subfield content. If the subfield in the concept schema does not have the repetitive punctuation, which means that it is not repetitive or has not appeared yet, then the regular punctuation is taken.

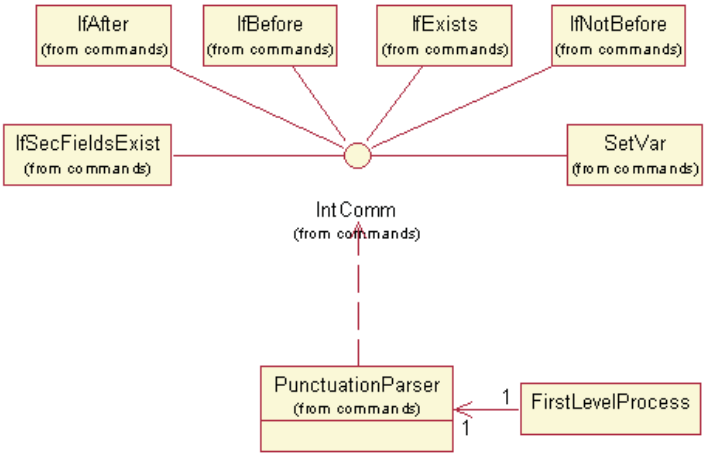

Figure 7: Class diagram for conditional punctuation

In this part of the program the processing rules are taken into account too. If the rule *or* is present, processing is finished after the first field is found, i.e. when all its subfields are processed. If the rule for processing is *and*, all fields stated in the concept should be successfully processed. As soon as any of the fields is not successfully processed, i.e. the result of searching for subfields in the record is an empty string, the processing stops, and the result is an empty string. If the rule for processing is *agreggation*, each field from the concept list, i.e. subfield content found in the record, is included in the result string. When the name of the concept is *ArabicTracing*, in the *ConceptProcess* class the *arabTracingProcess* method is invoked. Processing of this concept is implemented as a separate method, because structure of this concept in the schema document is different from other concepts. For some fields it is necessary to check only the value of indicators and write the appropriate text. Other fields are processed the same way as fields in other concepts (the *firstLevelProcess* method). There is the sequence number in front of each item of arab tracing.

If the concept is *RomanTracing*, in the *ConceptProcess* class the *romanTracingProcess* method is invoked. The class diagram for the *RomanTracing* concept process is shown in Figure 8. The *RomanTracingProcess* class uses the *TracingParser* class to match fields that are elements of the *RomanTracing* concept. The class that processes this matching is *IfContentEqual* and it implements the *TrComm* interface.

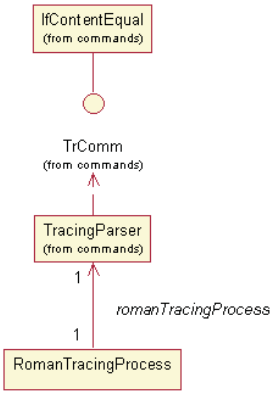

Figure 8: Class diagram of the *RomanTracing* concept

The sequence diagram of the *RomanTracing* concept process is shown in Figure 9. The *RomanTracingProcess* class uses the *TracingParser* class. This is necessary because the *RomanTracing* concept has conditional expressions, i.e. content of the subfield 6 is checked for the fields that are matched. The parse method and the conditional punctuation code check the passed string. The only conditional expression in the *RomanTracing* concept process is **if(6 equal_content)**. If this expression is recognized, an object of the *IfContentEqual* class is returned. The *execute* method that is implemented by the *IfContentEqual* class, returns the logic result *true* if the condition for matching the fields is satisfied; otherwise the result is *false*. If the fields that are matched are not repetitive (for example, field 700 and field 900, field 710 and field 910, field 720 and field 920), then it is enough to find occurrence of each of them in the record, and if they both exist, they should be written in the appropriate order with the punctuation "V." between them. But, if the fields are repetitive, matching is done using the same content of the subfield 6. The parser that processes the conditional punctuation is then invoked.

The resulting XML document is an instance of the schema modelled for checking if newly-generated XML documents are valid. The document formed in this way is ready for displaying by the second program.

In the second program the tags related to displaying are found and replaced with the appropriate HTML tags and attributes. The class diagram of the second program is shown in Figure 10.

In the *createSecondPass* method processing of the XML document starts. The *documentProcess* method is invoked and the HTML document header is created. Content of XML elements needed for visualisation is being extracted. For each tag whose name is an attribute name from the output XML document, there is a method which processes it (*justificationProcessing*, *punctuationProcessing*, *fontProcessing*, *writeProcessing*, *contentProcessing*). The content found in every tag is written into the appropriate HTML tags. For example,

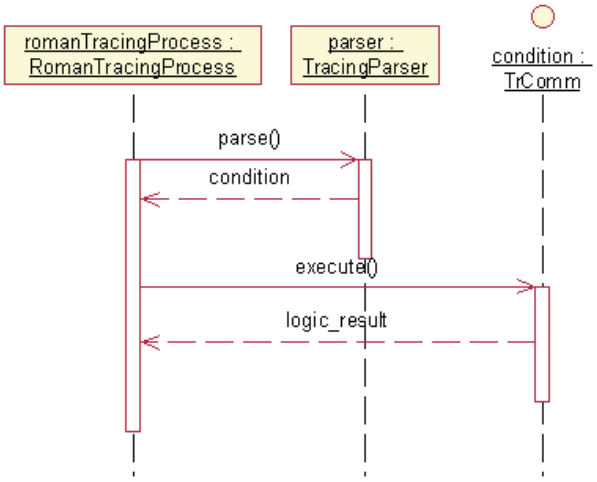

Figure 9: Sequence diagram for the *RomanTracing* concept process

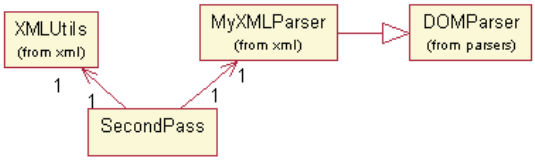

Figure 10: Class diagram of the second program

the content of the concept *CallNumber* is written right justified. The content of the attribute justification is put into the attribute ALIGN in the HTML tag TD, since it provides the justification needed. Each of the attributes is processed by a separate class, and the sequence diagram of the second program is shown in Figure 11.

The final result of these two programs is an HTML document ready for presentation by a browser. An example of the catalogue card display generated by the described application and shown by the browser can be seen in Figure 12. This is the example of the bibliographical catalogue card for the PhD thesis of the candidate Djordje Herceg. Its title is 'Convergence of simultaneous methods for determination of polynomial zeros'. This record is taken from the Library of Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad. It is in Serbian.

The detailed specification and the examples of outputs from the programs are shown in [9]. The HTML document which is the final result of these programs can be used as a means of showing existing catalogue records on the screen in the traditional catalogue card format using a web browser. This application can be extended for printing catalogue cards, so that catalogue cards for the new

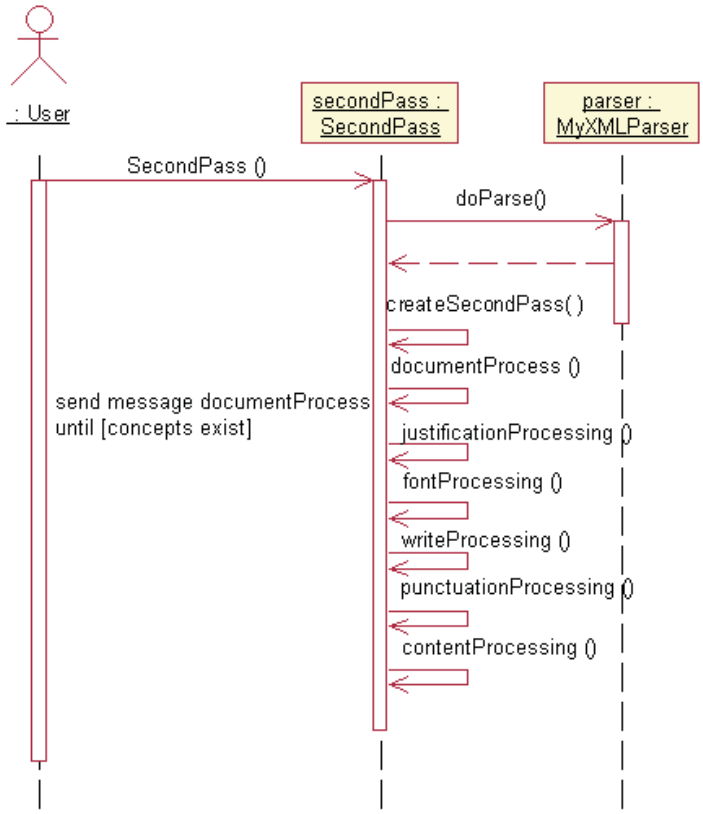

Figure 11: Sequence diagram of the second program

and existing catalogue records can be printed in the traditional format.

5. Conclusion

In this paper a way of bibliographical catalogue cards modelling using the XML Schema language is described, as well as display of these cards. The general description of catalogue cards is made by modelling the schema describing catalogue card concepts. Particular catalogue cards are modelled by individual schemas. The complete specification of the system for content generation and library catalogue cards display is given in the Unified Modelling Language. The implementation is done in the Java programming language based on the schemas.

This way of modelling bibliographical catalogue cards provides a simple method for adding new concepts and types of catalogue cards and for modifying the existing ones. If an existing type of catalogue card should be changed or a new type should be made using existing concepts, it is enough to change

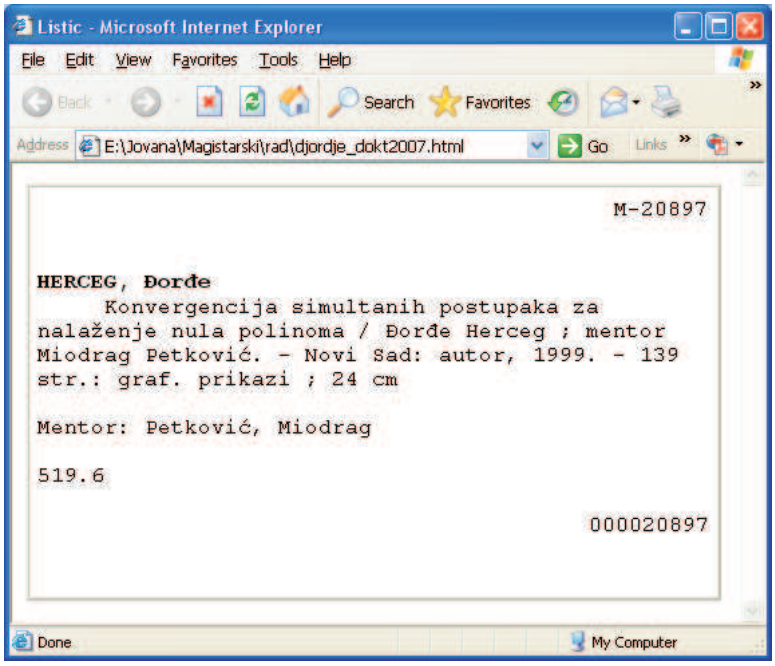

Figure 12: Example of a catalogue card

then existing schema or make a new schema for that type of catalogue card by citing only the concepts it consists of, which can be done by the end user, i.e. a librarian.

If a new concept should be created or an existing one should be modified, it is enough to change it or add it to the general schema by citing all the fields and subfields comprising the concept. It should be done by the rules which the schema has been modelled by, so as to enable the application to process the new concept as well.

Only if a new concept cannot fit into any of the existing rules, or needs some specific processing, it is necessary to change the existing programs.

The system can be expanded so that it can support other types of bibliographical reports. This way of report modelling may be applied to any other structured report.

Acknowledgement

This paper is part of the scientific research project 'Abstract Methods and Applications in Computer Science' no. 144017, supported by the Ministry of Science, Republic of Serbia.

References

[1] Family of ISBDs, http://www.ifla.org/VI/3/nd1/isbdlist.htm [February 15, 2007]

[2] Clarke, Kevin S., Medlane/XMLMARC Update: From MARC to XML Database, A Project of Lane Medical Library, Stanford University Medical Center http://elane.stanford.edu/laneauth/medlane_2001.html [February 15, 2007]

[3] MARC XML - MARC 21 XML Schema, Official Web Site http://www.loc.gov/standards/marcxml/ [February 20, 2007]

[4] Peters, B., MARC4J, http://marc4j.tigris.org/ [February 22, 2007]

[5] Ludwig, M., Breaking Through the Invisible Web - 1/15/2003, netConnect, CA266430 http://www.schoollibraryjournal.com/article/CA266430.html [March 2, 2007]

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

[7] Jakšić, M., Mapping of bibliographical standards into XML, Software: Practice and Experience, Softw. Pract. Exper. 2004; Volume 34, Issue 11, Pages 1051-1064

[8] Surla, D., Konjović, Z., Overview of the development of the library information system BISIS, Proceedings of the International Conference on Distributed LibraryInformation Systems, Ohrid, Former Yugoslav Republic of Macedonia, June 1-6, 2004, ISBN 86-7444-009-6, 13-18

[9] Vidaković, J., Modelling and implementation of bibliographical catalogue cards in XML technology, Master Thesis, Novi Sad, 2003. (in Serbian)

[10] Vidaković, J., Racković, M., Modelling the concepts of bibliographical catalogue cards using XML Schema language, Novi Sad Jour. of Mathematics, Vol. 33, No. 2, 2003, 95-102.

[11] Vidaković, J., Racković, M., Generating content and display of library catalogue cards using XML technology, Software - Practice and Experience, Volume 36, Issue 5, 25 April 2006 (513-524)

[12] Rađenović, J., Modelling and implementation of bibliographical cataloguing cards in software package FreeMarker, Master Thesis, Novi Sad, 2006. (in Serbian)

Received by the editors December 20, 2006