

# Automation of Metadata Repository Development with XML Schema

Wassilios Kazakos<sup>1</sup>, Alexei Valikov<sup>1</sup>, Andreas Schmidt<sup>1</sup>, Rainer Lehfeldt<sup>2</sup>

## Abstract

Metadata standards have been discussed since years in the environmental domain. Although the community has agreed that metadata are important for information finding, an globally accepted standard is still missing. Different requirements in diverse environmental application domains, countries and organisations have led to a huge variety on metadata vocabularies and technologies. Even projects using existing standards tend to adjust them slightly to their own requirements. A consequence is that new metadata repositories are developed or adapted to the new meta data element sets over and over again. In this paper we present a novel approach for the automation of metadata repository development. As input the system requires only an XML Schema describing the metadata vocabulary. Out of this schema the user interface, database schema and the search, retrieval and upload mechanism are generated during a batch process. The user interface layout can be further controlled by templates. The system is developed for the NOKIS project and evaluated by generating a metadata repository for an adjusted ISO 19115 metadata element set.

## 1 Introduction

The development of metadata repositories has a long history, starting with electronic library catalogues many years ago and is ongoing. Especially the environmental community has put a lot of effort in establishing standards for metadata and tools. One of the most successful standardisation examples in Europe for a specific interest group is the Umweltdatenkatalog (UDK)<sup>3</sup>, containing metadata about environmental information in Germany and Austria (Swoboda et.al. 1999, Swoboda et.al. 2000). But neither this initiative nor other promising initiatives, like the Environmental Markup Language (Arendt H.-K., Günther 1999) or EEA's WebCDS (Kazakos et.al.

---

<sup>1</sup> FZI Forschungszentrum Informatik, Database Systems Department, Haid-und-Neu-Str. 10-14, D-76137 Karlsruhe, Germany, email: {kazakos|valikov|aschmidt}@fzi.de, <http://www.fzi.de/dbs.html>

<sup>2</sup> Bundesanstalt für Wasserbau DS Hamburg, Wedelerstrasse 157, D-22559 Hamburg, email: [lehfeldt@hamburg.baw.de](mailto:lehfeldt@hamburg.baw.de), <http://www.hamburg.baw.de>

<sup>3</sup> [www.umweltdatenkatalog.de](http://www.umweltdatenkatalog.de)

1999) have led to a European application domain independent standard. The main reasons are the heterogeneity in requirements from local to international level as well as the different requirements in distinct environmental application domains, not to mention the lack of awareness of other standards and the lack of willingness to accept a standard developed in other countries and organisations.

In this paper we focus on the development aspect of metadata repositories, and leave the standardisation discussion aside. Since the core functional requirements for such repositories are well defined, we believe that more effort should be put in automating the development. This would lead in faster development cycles and easier adaptation to adjusted metadata element sets.

## 1.1 Project Background NOKIS

NOKIS<sup>4</sup> (North Sea Baltic Sea Coastal Information System) is a joint project of the KFKI, BfG and BAW<sup>5</sup> with the main objectives to establish a prototype meta data information system for the German North Sea and Baltic Sea coastal regions according to the concept of an open system which permits participation of additional partners at any time. To achieve this, within the project a local prototype meta database will be installed in every participating institution and will be integrated via the Internet. One of the key requirements to the NOKIS system is the possibility to adjust slightly the metadata element set on every local site following regional requirements, by adding, renaming or removing some of the elements. FZI develops the system for the NOKIS consortium reusing parts of the CoastBase<sup>6</sup> concept (Kazakos/Kramer/Schmidt 2000).

## 2 Overall Approach

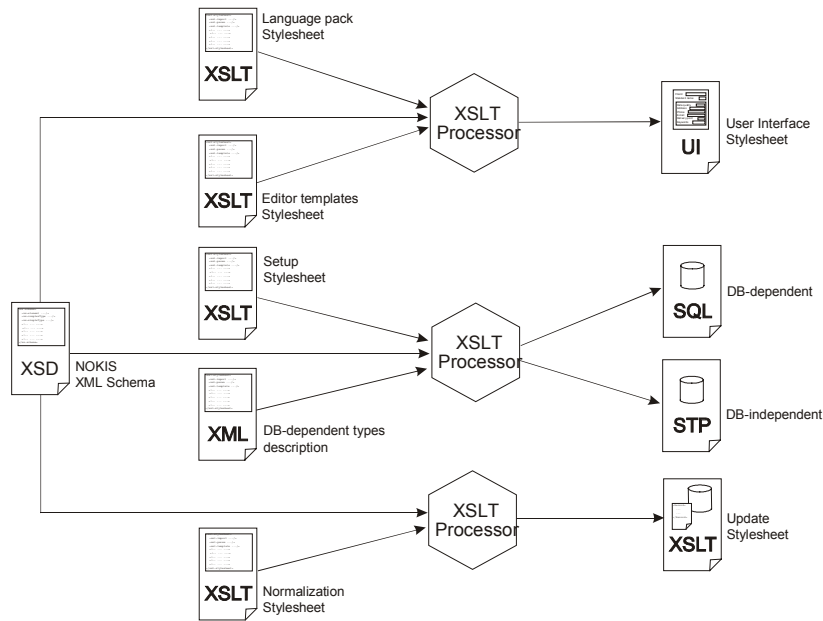
The overall system is developed by following in the main lines an XML-extreme approach as proposed in (Kazakos/Schmidt/Paoli 2001), i.e. using XML-technologies as far as possible, to achieve maximum reusability and easy configuration. We further extended our approach, and set up a system architecture that generates all components of a typical metadata repository out of an XML schema definition (XSD). The following picture illustrates the overall approach:

---

<sup>4</sup> <http://nokis.baw.de>

<sup>5</sup> KFKI - Kuratorium für Forschung im Küsteningenieurwesen, BAW - Bundesanstalt für Wasserbau, BfG - Bundesanstalt für Gewässerkunde

<sup>6</sup> <http://www.coastbase.org>



Firstly, the database schema is generated through a two step transformation process. In the first step a database system independent database setup is generated from of the metadata XSD and a setup stylesheet. In a second step the database system dependent schema is generated out of the database setup and the database system dependent types.

Secondly, the stylesheets for the user interface are generated from the metadata element set XSD by using the editor templates, describing how to present complex and simple XML Schema types. For the editor templates we provide default styles for complex and simple typed elements. In order to support elements with special layout, default templates can be extended (custom UI templates). In order to provide a multilingual user interface, the generation process is additionally controlled by a language pack stylesheet.

Thirdly, in order to be able to store XML documents in the database, we need an update stylesheet, normalizing the XML document for storage. This is generated out of the metadata XSD and the normalization templates stylesheet.

The approach creates out of a given XSD a complete Web-based repository, including automatic database schema and user interface generation. It provides additionally the software modules needed for the runtime of the system. In order to be able to automate this process, we need to put certain restrictions on initial XML Schema.

Firstly, all complex elements must have named complex types defined as sequences of other complex and simple elements. Secondly, there must not be any type cycling (for instance, when type A may have an element of type B and type B

may have an element of type A). Thirdly, all simple elements must be either primitive XML Schema types (like string or integer) or enumerations. And finally, attributes must not be used.

Although these restrictions seem to be very burdensome, the set of suitable XML Schemas is still very large. Moreover, these restrictions do not constrain the approach itself, as additional programming effort should allow overcoming most of them.

### 3 Storing Metadata Records in a Generated Database Schema

There are two important considerations in metadata storage: efficient query processing and fast document retrieval. To achieve this, we suggest storing metadata in parallel in a file system and in a database.

Storing metadata records in a file system ensures almost the fastest document retrieval possible, as the file is only read from the server's hard drive and sent to the requesting client. At the same time, file system-based storage is very inefficient in a sense of query processing. Evaluating a query against thousands of metadata files is simply unacceptable. For the sake of efficient query processing, metadata may be also saved in a relational database.

For the storage of the metadata we generate the database schema automatically out of the XML Schema definition. As long as all complex types are explicitly defined in the document XML Schema and structurally consist of sequences of other elements, there is a clear correspondence between entities of the XML document and entities of the relational database. This correspondence is described in the following table

| <i>XML Document Entities</i>                   | <i>Relational Database Entities</i> |
|--|-------------------------------------|
| Complex type                                   | Table                               |
| Element of a complex type (complex element)    | Row                                 |
| Simple single sub-element of a complex element | Column                              |

Pure relational databases do not allow usage of complex data types. Columns of tables must have atomic values; therefore complex sub-elements must be stored in separate tables. A similar situation is with simple sub-elements, which may be repeated inside its parent element. For instance, a person may have several e-mail addresses, or several phone numbers. This is a usual situation for XML documents. However, relational databases do not allow multiple columns with the same name. Storing data from several elements in one column would violate normalization rules, essential for every database schema. Obviously, repeatable elements must be stored in separate tables.

The conclusion is that two types of tables are required:

- One table for each complex type of the document schema.
- One table for each repeatable simple element inside each complex type of the document schema

Following the specifications given in the document schema, both types of table definitions may be generated automatically, as the correspondence is very clear. However, to ensure database independence, one more step should be considered. In order not to be tied to the specific database implementation, database schema for document storage must be defined in a more abstract way. A reasonable solution is to first generate it in an abstract XML-based database schema definition language (we name it the Setup), which after that may be converted into data definition language (DDL) statements for a specific database.

This effectively means several simple steps. On the first step, document schema is transformed into the database Setup. The Setup precisely describes tables, columns, keys and key references but does not go into database-specific details like data types etc. On the second step, data type mapping is defined for the specific database. Alternatively, an existing data type mapping may be modified, as differences are usually not too big. On the third step, the database Setup is transformed into DDL statements for the specific database. This last step uses data type mapping definition made on the second step; it may additionally require changing transformation stylesheets, if DDL syntax of the target database does not follow SQL standards.

The results of completing these steps are two database schema definitions: one in a form of database-unspecific Setup and another in a form of DDL statements for the target database. Running the latter in the target database will construct the full database schema capable of storing content and structure of documents in the initial document schema. Background information on how to map XML to relational databases can be found in (Kazakos/Schmidt/Tomczyk 2002).

#### **4 User Interface Generation**

The user interface of the document repository may consist of parts, which do not depend on document XML schema and parts, which depend on it. Correspondingly, if schema-independent parts may be developed manually, schema-dependent parts must be generated automatically.

A good example of a schema-dependent part of the user interface is the document editor. Document authoring should be strictly aligned with its schema, otherwise metadata quality and validity would be very difficult to ensure.

The aforementioned restrictions on the document XML schema simplifies the document structure and allows automatic generation of schema-dependent user interface parts.

The most basic elements of XML documents are simple elements, which may only contain text nodes. Complex elements are composed of sequences of other simple and complex elements. As type cycling is prohibited, any complex element is, finally, a structural composition of simple elements. It means that if interface components for presenting simple elements are defined, presentation of any complex element may be structurally composed of these components as well.

Consequently, schema-dependent user interface part may be built based on the following two prerequisites:

- simple element interface components.
- a procedure for constructing complex element interface components from interface components of its sub-elements.

This approach may be illustrated by the document editor example. In this case, simple element interface components are templates to show editors for simple types used in the document schema. These templates are either written manually for the most primitive XML Schema data types (like string, date, boolean, etc.) and simple types, which require custom editing, or created automatically based on information from the document schema (primitive type extensions, drop-down lists for enumerated types, etc.). Each of these simple type editor templates is named after the simple type that this template edits.

The image shows a screenshot of a web form titled "Metadata record information". The form contains several fields and sections:

- Record ID:** fzi74528
- Language:** German (dropdown menu)
- Charset:** UTF-8 (dropdown menu)
- Parent record ID:** (empty text input)
- Hierarchy level:** [not selected] (dropdown menu)
- Timestamp:** (date input) - [...] (dropdown menu) - Now (button)
- Standard name:** ISO 19115 (text input)
- Standard version:** Version 1.0 (text input)
- Contact information:** (expandable section)
- Maintenance information:** (text input)
- Maintenance Frequency:** [not selected] (dropdown menu)
- User-defined Frequency:** (text input)

Annotations on the form include:

- Complex type editor:** A bracket on the left side of the main form area.
- Enumerated type editors:** A bracket on the right side pointing to the Language, Charset, and Hierarchy level dropdowns.
- Simple type editors:** A bracket on the right side pointing to the Parent record ID, Timestamp, Standard name, and Standard version inputs.
- Nested complex element editors:** A bracket on the left side pointing to the Maintenance information section.
- Add element buttons:** A label pointing to the expandable section icons.
- Remove element buttons:** A label pointing to the "Remove this element" buttons at the bottom of the nested sections.

**Fig. 1. Complex element editor and its components**

The SEQARABISCHSEQARABISCHeditor template for each of the complex types is constructed from calls to the templates of its sub-elements. Visually, com-

plex type editor is a form, where first simple sub-element editors are displayed, and then complex sub-editor forms are included. Elements are added with "+" and removed with "-" button, which are displayed according to the definition of element's cardinality in document schema ("+" button will disappear, if single allowed element instance already exists). This composition strategy is defined once and then used to generate complex element editor templates for each of the complex types used in the document schema.

SEQARABISCHA After this is done, we will have editor templates for all (primitive, simple and complex) types used in a given document schema. Uniting them into one stylesheet will result in a document editor stylesheet. If we apply this stylesheet to a document we want to edit, we will receive a web page, which structurally displays the document data in input fields. Modifying values of these fields changes the document data; adding and removing elements changes the document structure. This actually means editing the document with full respect to its schema.

The generation approach illustrated above for the case of the document editor interface is used for all other schema-dependent user interface parts. As mentioned before, these tasks always consist of two parts: (a) define set of basic templates for simple types and (b) specify how these simple type templates should be combined into templates for complex types. Applying the procedure specified in part (b) to the template library defined in part (a) generates a stylesheet for a schema-dependent part of the user interface.

Compared to the manual development of user interface, this approach drastically reduces amount of work effort required for the implementation due to the following reasons:

- The number of simple types used in documents schemas is usually much lower than quantity of complex elements.
- Template libraries for simple types are easily reusable across different schemas. In particular, editor templates for XML Schema primitive data type do not require changes at all.
- If the document schema evolves, templates may be re-generated.

The automatic generation approach may seem to have a drawback that the user interface built according to the "default" rules may also have a "default" look, while manually developed interfaces allow higher level of customisation. As a matter of fact, automatic generation also allows a high level of customisation and only if customisation is not needed (and, therefore not defined), default rules are used.

## 5

## 5 Conclusions and Further Work

The combination of automatic approaches described in this paper significantly reduced working effort needed for the development of a metadata repository. Even with very complex schemas like ISO 19115, the system design is very tolerant to schema evolution, as schema changes do require almost no manpower investments. The system is database and platform-independent and built according to the latest standards based on free and open-source components. It is multilingual and currently translated to English and German. The system contains a context-sensitive help system, which upon request provides advice both on user interface and on the metadata. Documents stored in the system may be exported into pure XML, into plain text or HTML as well as into a high-quality printable PDF document, or into other metadata formats. The system is also capable of importing metadata in other formats (e.g. the ESRI XML export). The system may be integrated with external applications, like map viewers or thesauri browsers.

Our further developments in this area will be dedicated to repository generation based on the schema of an existing database. This would allow rapid creation of repository applications for already existing databases.

## 6 Bibliography

- Arendt H.-K., Günther (eds.) (1999): Environmental Markup Language (EML). Workshop 1, Berlin.
- Kazakos, W., Kramer, R., Schmidt, A. (2000): Coastbase - The Virtual European Coastal and Marine Data Warehouse. In Armin Cremers and Klaus Greve, editors, Computer Science for Environmental Protection '00. Environmental Information for Planning, Politics and the Public, volume II, pages 646-654.
- Kazakos, W., Kramer, R., Nikolai, R., Rolker, C., Bjarnason, S., Jensen, S. (1999): WebCDS - A Java-based Catalogue System for European Environment Data. In Dogac, A., Özsu, M.T., Uluzoy, O (Eds): Current Trends in Data Management Technology, pp. 234 - 249. Idea Group Publisher.
- Kazakos, W., Schmidt, A., Paoli H. (2001): XML based Virtual Catalogue Module in Coastbase. In Lorenz Hilty and Paul Gilgen, editors, Sustainability in the Information Society. 15th International Symposium Informatics for Environmental Protection, volume II, pages 513-520.
- Kazakos, W., Schmidt, A., Tomczyk, P. (2002): Datenbanken und XML – Konzepte, Anwendungen, Systeme. Xpert.press. Springer Verlag. ISBN 3-540-41956-X. <http://www.datenbanken-und-xml.de>.
- Swoboda, W., Kruse, F., Nikolai, R., Kazakos, W., Nyhuis, D., Rousselle, H. (1999): The UDK Approach: the 4th Generation of the Environmental Data Catalogue for Austrian and German Public Authorities, Proc. IEEE Meta-Data'99, <http://computer.org/proceedings/meta/1999/papers/45/wswoboda.html>

Swoboda, W., Kruse, F., Legat, R., Nikolai, R., Behrens S. (2000): Harmonisierter Zugang zu Umweltinformationen für Öffentlichkeit, Politik und Planung: Der Umweltdatenkatalog im Einsatz. In Armin Cremers and Klaus Greve, editors, Computer Science for Environmental Protection '00. Environmental Information for Planning, Politics and the Public, volume II Pp 595-607