XML Data Transformations as Schema Evolves*

Jakub Malý, Irena Mlýnková, and Martin Nečaský

XML Research Group, Charles University in Prague, Czech Republic
{maly,mlynkova,necasky}@ksi.mff.cuni.cz

Abstract. One of the key characteristics of XML applications is their dynamic nature. When a system grows and evolves, old user requirements change and/or new requirements accumulate. Apart from changes in the interface, it is also necessary to modify the existing documents with each new version, so they are valid against the new specification. The approach presented in this paper extends an existing XML conceptual model with the support for multiple versions of the model. Thanks to this extension, it is possible to define a set of changes between two versions of a schema. This work contains an outline of an algorithm that compares two versions of a schema and produces a revalidation script in XSL.

Keywords: XML schema, conceptual modeling, evolution, revalidation.

1 Introduction

The eXtensible Markup Language (XML) [16] has become a standard for data representation and manipulation and, hence, invoked a boom of so-called XML applications that exploit a whole family of XML technologies. A typical current XML application usually consists of a set of sub-applications, each being responsible for a particular logical execution part. The life-cycle of such an XML system of applications is similar to a life-cycle of a single application, however the complexity is much higher. First of all we need to design of numerous data structures, i.e. XML schemas, that are exchanged and processed by business processes of the system. What is more, they are usually mutually related or overlayed. In other words, each application of the system utilizes several views of a common problem domain represented by XML schemas. Hence, they cannot be designed separately, but as a whole complex system. In addition, sooner or later the user requirements of the applications change and, hence, the data structures they input, process and output must be modified respectively – we usually speak about the problem of XML schema evolution.

In our previous work [15] we have proposed a five-level XML evolution framework that enables one to face the described issues. It utilizes the concepts of MDA (Model-Driven Architecture) [12] hierarchy of conceptual models that enable a user to abstract from specifics of a particular XML format, enables one to model a whole set of related XML applications concurrently and preserves the

* Supported by GAČR grant no. P202/10/0573, the grant no. 201/09/P364 and the grant no. P202/11/P455.

respective relations between them. Consequently, it naturally supports evolution management.

In this paper we focus on one particular aspect of the complex framework – propagation of changes between multiple versions of an XML schema to the respective instances, i.e. XML documents. We speak about the process of revalidation. We describe a unique approach that enables one to output so-called revalidation script, i.e. an XSLT [6] script that, when applied on the given set of XML documents valid against an old version of an XML schema, outputs a set of XML documents valid against the new version of the schema. The approach enables to reduce manual and, hence, error-prone tasks via cutting down the user interaction to the necessary minimum.

The paper is structured as follows: In Section 2 we provide an overview of the related work. In Section 3 we describe the conceptual model we utilize for the purpose of change specification. In Section 4 we define the set of changes that can be performed over the models and in Section 5 we provide the algorithm for generation of the XSLT revalidation script. In Section 6 we describe implementation of our approach and provide a complete illustrative example. Finally, in Section 7 we conclude and outline future work.

2 Related Work

For the goal of determining whether the set of documents was invalidated with the newly coming version of the schema, the system must recognize and analyze the differences between them. There are two possible ways to recognize changes – recording of the changes as they are conducted during the design process and comparing the two versions of the schema.

An evolution system that utilizes recording of the changes (e.g. [7]) usually provides some kind of a command that initiates the recording and after issuing this command all operations carried out by a user over the schema are recorded. When the desired schema is reached, the user finishes recording and the system has all the information about the changes made – the sequence of performed operations. When the recording is finished, the system can normalize the sequence for example by eliminating operations that cancel each other or by replacing groups of operations by other groups that lead to the same result but in a more straight way. These normalizing rules must be defined in the system.

An alternative approach is to base the change detection on comparison of the two versions. The user can work with both schemas independently until (s)he is satisfied with them. Before detecting changes, the mapping between the two schemas must be found, which requires some degree of user interaction ( [10] uses a visualization tool for mapping editing). The change detection algorithm then takes the two schemas as an input and compares them. The result of the comparison is a list of differences between the schemas.

On the contrary, systems X-Evolution [3] and XEM [4], built upon graphical editor for creating schemas in the XML Schema [18] or DTD [19] respectively, use an incremental validation. Each single evolution operation executed upon the