# X-Evolution: A System for XML Schema Evolution and Document Adaptation

Marco Mesiti[1], Roberto Celle[2], Matteo Sorrenti[1], Giovanna Guerrini[2]

(1) Università di Milano, Italy - mesiti@dico.unimi.it
(2) Università di Genova, Italy - guerrini@disi.unige.it

## 1 Introduction

The structure of XML documents, expressed as XML schemas [6], can evolve as well as their content. Systems must be frequently adapted to real-world changes or updated to fix design errors and thus data structures must change accordingly in order to address the new requirements. A consequence of schema evolution is that documents instance of the original schema might not be valid anymore. Currently, users have to explicitly revalidate the documents and identify the parts to be updated. Moreover, once the parts that are not valid anymore have been identified, they have to be explicitly updated. All these activities are time consuming and error prone and automatic facilities are required.

A set of primitives that can be applied on an XML schema to evolve its structure has been proposed in [4]. For each primitive we have determined the applicability conditions, that is, when its application produces a schema that is still well-formed. Moreover, we have analyzed when the primitive application alters the validity of the documents instances of the schema. Table 1 reports the evolution primitives classified relying on the object (element, simple type, and complex type) on which they are applied and the kind of operation (insertion, modification, deletion). Primitives marked with "*" do not alter the validity of the document instances. Therefore, their application do not require a revalidation process. Other primitives might only alter the validity of a single element or of a restricted set of elements depending on the schema specification. Therefore, in these cases, the entire revalidation of a document is useless. In [4] we developed an algorithm, based on an element type graph labelling, that minimizes the revalidation process to only the elements affected by the primitives thus making the process more efficient.

A related problem is how to evolve the structure of document instances in order to make them valid for the evolved schema. Suppose to introduce an element in a schema, and this is mandatory for all valid documents. This element should be introduced (maybe with a default or null value) in all the previously valid documents. Suppose also to remove an element from the schema. It should be also removed from valid documents. The problem is more complex when the schema modification refers to an operator or to the repeatability of elements, and would often require user intervention. The adaptation process thus involves subtleties related both to the kind of update performed and to the structure of

| | Insertion | Modification | Deletion |
|---|---|---|---|
| Simple Type | *insert_glob_simple_type** <br> *insert_new_member_type** | *change_restriction* <br> *change_base_type* <br> *rename_type** <br> *change_member_type* <br> *global_to_local** <br> *local_to_global** | *remove_type** <br> *remove_member_type** |
| Complex Type | *insert_glob_complex_type** <br> *insert_local_elem* <br> *insert_ref_elem* <br> *insert_operator* | *rename_local_elem* <br> *rename_global_type** <br> *change_type_local_elem* <br> *change_cardinality* <br> *change_operator* <br> *global_to_local** <br> *local_to_global** | *remove_element* <br> *remove_operator* <br> *remove_substructure* <br> *remove_type** |
| Element | *insert_glob_elem* | *rename_glob_elem** <br> *change_type_glob_elem* <br> *ref_to_local** <br> *local_to_ref** | *remove_glob_elem** |

Table **1.** Evolution primitives

the updated type. Several updates require the detection of the *minimal substructure* for an element whose insertion is required in documents to validate. Our approach to document adaptation is based on the use of *restructuring structures*, that are an extension of the labelled element type graph employed for document revalidation, in which labels can also be $\Delta_l^\epsilon$, $\Delta_\epsilon^l$, and $\Delta_{l_n}^{l_o}$, with $l, l_n$, and $l_o$ element labels. These structures allow to specify the minimal modifications to be performed on documents invalidated by a schema update and are automatically inferred from the schema update whenever possible (otherwise, user intervention is required). The adaptation process will occur during the revalidation process and the idea is that to validate the special subelement $\Delta_l^\epsilon$, element $l$ should be inserted. Similarly, to validate the special subelements $\Delta_\epsilon^l$ and $\Delta_{l_n}^{l_o}$, element $l$ should be deleted and element $l_o$ should be renamed to $l_n$, respectively.

In this demonstration paper we present X-Evolution, a .NET system developed on top of Oracle 10g that allows the specification of schema modifications in a graphical representation of an XML schema. It supports facilities for performing schema revalidation only when strictly needed and only on the minimal parts of documents affected from the modifications. Moreover, it supports the adaptation of original schema instances to the evolved schema. The adaptation process, as well as the revalidation one, can be started after a certain amount of schema updates have been performed, and is not necessarily started after each single modification. The adaptation process is semi-automatic and the required user intervention is minimized. Support is provided to the user for a convenient specification of the required updates.

Commercial tools (e.g. [1, 5]) have been developed for graphically design XML schemas. However, they are not integrated with a DBMS and do not allow the semi-automatic revalidation and adaptation of documents within contained. Schema evolution had been previously investigated for DTDs in [3], where evolution operators are proposed. Problems caused by DTD evolution and the impact on existing documents are however not addressed. Moreover, since DTDs are considerably simpler than XML Schemas [2] the proposed operators do not cover all the set of schema changes that can occur on an XML Schema.

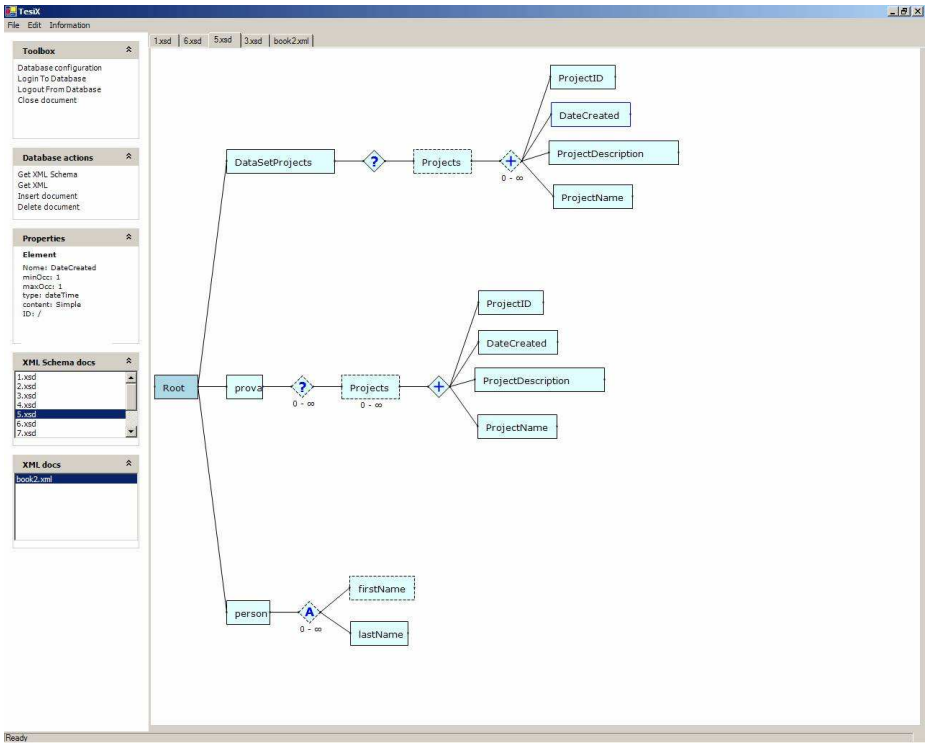


Figure **1.** X-Evolution schema modification facility

## 2 X-Evolution Facilities

X-Evolution offers different kind of facilities for handling the evolution of XML schemas, an efficient revalidation of document instances, and the adaptation of document instances to the evolved XML schema.

X-evolution connects to one or more databases in which the XML schemas and documents are stored. The left side bar of the schema modification facility (Figure 1) presents the identified documents and schemas. Whenever a user clicks on a schema, the system graphically represents the schema and identifies the documents that are valid for such a schema. Whenever a user clicks on a document, the system graphically represents the document and identifies the schemas for which the document is an instance. We remark that a document can be instances of several schemas that can present slight differences.

By graphically selecting a node of the tree representation of a schema,[1] all the possible schema evolution primitives that can be applied on such a node are visualized. When the user invokes an evolution primitive, X-Evolution checks

[1] Actually a schema should be represented as a direct graph. However, for the sake of readability, we duplicate nodes of the graph with more than one incoming edge.

whether the operation can alter the schema consistency. If it is preserved the operation is executed and the evolved schema visualized. Whenever the operation alters (or can alter) the validity of document instances of the schema, X-Evolution points out the documents that are not anymore valid. This operation is performed through an optimized revalidation algorithm detailed in [4]. The user interface helps the user in the adaptation process of non valid documents. For each non valid document the system points out the elements that should be removed or added and allows the specification of default values or structures to be inserted.

As mentioned before, document adaptation as well as document revalidation can be postponed after a sequence of evolution primitives have been applied, rather than being applied for each single schema modification. X-Evolution allows the application of such a sequence of evolution operations and, at the end, the execution of the adaptation and revalidation process.

## 3 Demonstration

The demonstration of the X-Evolution system consists in four parts.

1. We will show how to connect the graphical interface to a database in which the documents and schemas are stored and how we can easily work with the graphical representation of documents and schemas.
2. We will show how to apply the developed evolution operations on a schema. Specifically, we will show how to construct a schema from scratch and how we can remove all the components from a schema making it an empty schema. Consistency of the resulting schema is checked for each operation, and, if violated, no update is performed.
3. We will show performances of our efficient approach in the revalidation of XML documents against the evolved schema with respect to the naive solution of entirely revalidate the documents instances of the schema.
4. Finally we will show the effectiveness of our adaptation approach by specifying schema modifications and showing the result of the adaptation process.

*Acknowledge* The authors wish to thank Daniele Ghelli and Giuseppe Marchi for developing the graphical representation of XML documents and schemas.

## References

1. Altova Inc. XMLSpy, 2005. `http://www.altova.com/`
2. G.J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: A Practical Study. *WebDB*, 79–84, 2004.
3. D. K. Kramer and E. A. Rundensteiner. Xem: XML Evolution Management. *RIDE-DM*, 103–110, 2001.
4. G. Guerrini, M. Mesiti, D. Rossi. Impact of XML Schema Evolution on Valid Documents. In Proc. of WIDM, Germany 2005.
5. Stylus Studio Inc. Stylus XML Editor, 2005. `http://www.stylusstudio.com`
6. W3C. XML Schema Part 0: Primer, 2001.