Managing Collections of XML Schemas in Microsoft SQL Server 2005

Shankar Pal, Dragan Tomic, Brandon Berg, and Joe Xavier

Microsoft Corporation, Redmond, WA 98052, USA
{shankarp, dragant, branber, joexav}@microsoft.com

Abstract. Schema evolution is of two kinds: (a) those requiring instance transformation because the application is simpler to develop when it works only with one version of the schema, and (b) those in which the old data must be preserved and instance transformation must be avoided. The latter is important in practice but has received scant attention in the literature. Data conforming to multiple versions of the XML schema must be maintained, indexed, and manipulated using the same query. Microsoft's SQL Server 2005 introduces XML schema collections to address both types of schema evolution.

1 Introduction

Many XML schemas have become standard in vertical industry segments such as ACORD [1] and SportsML [5]. Microsoft's Office products have made their XML schemas openly available [3]. These efforts have lead to better interoperability among loosely connected systems and the development of new applications, such as rich search and data sharing among applications. Standards bodies evolve their schemas, which may require reworking the applications since the XML schema evolution may require database schema changes. The scenarios for schema evolution are:

1. The new schema extends an existing one with new schema components. For example, the new schema can add a top-level element called "language" if a publisher enters a new, foreign market. The application considers existing data to have a default value for the new schema components (e.g. US English).
2. The schema is modified in an incompatible way. This often occurs with change in business needs and merger of systems. The existing data must be mapped to the new schema, using, for example, XSL transformations [8], if it does not conform to the new schema. The data transformation can be avoided in some cases (e.g. maxOccurs of <phone> changes from 7 to 5 but no instance exceeds 5 <phone>s).
3. The schema undergoes modification as government or business rules change. New data must conform to the new schema while old data must be retained in its old form for archival purposes. Examples are tax filing and securities trading. Tax laws change from one year to the next, but old tax returns should not be transformed to conform to the latest version of the XML schema.

While the first two kinds of schema evolution have been studied extensively in the literature, the third kind is a practical problem with a few ad hoc solutions. SQL Server 2005 addresses it using a meta-data notion called an *XML schema collection* –

Y. Ioannidis et al. (Eds.): EDBT 2006, LNCS 3896, pp. 1102 – 1105, 2006.

a container of XML schemas that may be related (e.g. through <xs:import>) or unrelated to one another. Each schema in an XML schema collection C is identified using its target namespace. A new version of an XML schema with a new target namespace can be added to C and is treated like a new schema. This framework is powerful enough to support the other types of schema evolution too.

The XML data is stored in a column of a rich data type called XML, as opposed to decomposing the data into tables and columns. This avoids database schema modification when XML schemas in C are added, dropped or modified. The database engine validates each XML instance according to XML schema specified in the XML instance during data assignment and modification.

2 XML Schema Collection

An XML schema collection C is created using the following statement and registering one or more XML schemas:

```
CREATE XML SCHEMA COLLECTION C AS '<xs:schema> …
</xs:schema>'
```

The schema processor identifies and stores in C the schema components supplied in the schemas. An *XML schema component* is anything defined at the top level of an XML schema, such as an element, attribute, type, or group definition. Schema components from multiple XML schemas with the same target namespace are grouped together by the target namespace.

A user can type an XML column using C. The constraint imposed by C is the collective set of the schema constraints imposed by the individual XML schemas in C. The post-schema validation Infoset (PSVI), which adds type information to the Infoset [6], is encoded in the internal representation of the XML data for faster parsing during XQuery [7] processing.

2.1 Schema Evolution

Suppose you add an XML schema with target namespace BOOK-V1 to an XML schema collection C. An XML column XDOC typed using C can store XML data conforming to BOOK-V1 schema. To extend the XML schema, the schema designer adds the new schema components to the BOOK-V1 namespace. Adding optional elements and attributes, and top-level elements and type definitions do not require re-validation of the existing XML data in column XDOC. Suppose later the application wants to provide a new version of the XML schema, for which it chooses the target namespace BOOK-V2. This XML schema is added to C without transforming the existing XML instances in XDOC. The XML column can store instances of both BOOK-V1 and BOOK-V2 schemas. This yields significant simplification in data management when C contains a large number of XML schemas. The user can insert and modify XML instances conforming to the latest version of the schema as well as those conforming to the older versions of the schema.

The XML schema collection framework can also support applications that require instance transformation. The application has to supply the modified XML schema and