A Time Machine for XML: PUL Composition

Ghislain Fourny
ETH Zurich
<gfourny@inf.ethz.ch>

Daniela Florescu
Oracle
<dana.florescu@oracle.com>

Donald Kossmann
ETH Zurich
<donaldk@inf.ethz.ch>

Markos Zacharioudakis
Oracle
<markos_za@yahoo.com>

Abstract
As storage - main memory as well as disk - becomes cheaper, the amount of available information is increasing and it is a challenge to organize it. Our broader aim is to provide a unified framework for efficiently versioning and querying data, documents, as well as any kind of semi-structured information between data and documents, which can be stored as XML. In order to query this information, we started with the XQuery programming language, and extended its data model, its syntax and its processing model to make it seamlessly time-aware. More specifically, to do this, we made the assumption that the changes made by an XQuery program can always be expressed as a PUL. This is not obvious with an XQuery Scripting Extension program, as it can apply several, possibly interdependent, PULs. The contribution of this paper is to introduce a new operation on pending update lists, called PUL composition, which allows to summarize the changes made by an entire XQuery Scripting program with a single PUL instead of a sequence of PULs.

Keywords: XML, Versioning, XQuery, CVS, SVN
1. Background

A full paper describing the versioning system is available as a technical report [1]. In this section, we summarize the most important concepts.

XML nodes are organized in trees. During the execution of an XQuery program, or several XQuery programs, trees evolve.

In our versioning system, the evolution of a tree is modeled as a tree timeline. A tree timeline is a sequence of trees, which are actually "the same tree" at successive moments in time, as represented on Figure 1 (a).

![Figure 1. Tree timelines, node timelines and versions](image)

Exactly like trees, nodes are organized in node timelines (b). A node timeline is a sequence of nodes, which are actually "the same node" at successive moments in time (same reference URI, distinct identities). Within a node (tree) timeline, a node (tree) is uniquely identified by a version (c).

Deltas between several trees in a timeline are expressed as single PULs, as can be seen on Figure 2. It is this very assumption that we aim to justify.

2. PUL Composition

While in the Update Facility, it is obvious that changes made by the program can be expressed as a PUL, this is not straightforward for the Scripting Extension. In the Scripting extension, several PULs might be applied locally, and these PULs might be interdependent (e.g., the target of a PUL might be in the contents previously inserted by another PUL), so that they cannot just be merged (with *upd:mergeUpdates*, described in the XQUF specification). The Scripting Extension introduces a time
Figure 2. Deltas as single PULs

component which has to be taken into account. At first, it seems that deltas must be expressed by sequences of PULs (e.g., a and b on Figure 3).

In this section, we show how two, possibly interdependent, PULs can be composed to one single PUL (e.g., c on Figure 3). This is what allows us to express our deltas as single PULs. Note that PUL composition is different from PUL merging (\texttt{upd:mergeUpdates}), since the PULs are possibly not independent, i.e., they are not applied to the same snapshot.

We give an operational definition of PUL composition. A local PUL is maintained locally, which summarizes all local changes made since the beginning of the execution of the program, or since the last checkout. It is normalized (see Section 3), and each time a new PUL is applied by the (XQuery Scripting Extension) program, a copy of this new PUL is also composed with the local PUL (see Section 6). Note that the local PUL is not meant to be applied - rather, it is meant to be committed (checked in) to the repository in our versioning system.

The idea behind PUL composition is very simple. For each update primitive being composed with the local PUL:

- Either the target of this update primitive was already here at the beginning of the program (or at the last checkout), in which case the update primitive is accu-
mulated against the local PUL (accumulation, shown on Figure 4 and detailed in Section 4).
• Or its target is in the contents of the local PUL, in which case the extended semantics of the update primitive is made effective (aggregation, shown on Figure 3 and detailed in Section 5)

3. The normalized local PUL

In XQuery, there is a dynamic context which contains dynamic information about the execution of a program. We extend the dynamic context with a local PUL, which contains information about all changes which have been made since a certain point in time (e.g., the beginning of the program, or say the last checkout). The local PUL can be seen as the delta between this point in time and now.

The local PUL is normalized, which means that, without loss of generality, for each target, there is at most one update primitive of each kind (upd:insertBefore, upd:insertAfter, upd:insertIntoAsFirst, upd:insertIntoAsLast, upd:insertAttributes, upd:replaceNode, upd:replaceValue, upd:replaceElementContent, upd:delete and upd:rename - upd:put is out of the scope of this paper, upd:insertInto deserves a special treatment
and can always be replaced with another inserting update primitive) with this target. This was already the case for replacing and renaming update primitives in the specification, and in case there are several inserting update primitives of the same kind with the same target, the XQuery Update Facility specification says that "ordering of nodes within each group is preserved but ordering among the groups is implementation-dependent." Whenever a PUL contains several inserting update primitives of the same kind sharing the same target, this allows us to group the contents of these update primitives inside a single update primitive of that kind, in an implementation-dependent way, without altering the semantics of the PUL. The local PUL is kept normalized when aggregating or accumulating update primitives, as defined in the next sections.

Also, there are backpointers from the nodes of the local trees in the store (read by the XQuery program) to their copy in the local PUL whenever there is one (i.e., whenever they were added by the program), in order to know where they came from (see Figure 5). These backpointers are followed when the PUL to be applied is copied to a new PUL which is to be composed with the local PUL (see Section 6). From now on, as on Figure 5, we take the convention that the nodes encircled in black are copies which belong to contents of the local PUL or to the contents of the

Figure 4. Two PULs (a, b) can be summarized in a single PUL (c) (Accumulation Case)
update primitive being composed. The identities of these copies are distinct from those of the original nodes.

4. Accumulation

When the target of an update primitive was already here at the last checkout (i.e., it is not to be found anywhere in the contents of the local PUL - to put it simply, it is not encircled in black), then this update primitive is accumulated against the local PUL. It is mostly like merging them (upd:mergeUpdates), with the additional constraint that the local PUL remains normalized (Figure 6).

For upd:insertBefore, upd:insertIntoAsLast and upd:insertAttributes, the contents of the update primitive are inserted at the end of the contents of the update primitive of the same kind in the local PUL (or the update primitive is inserted in the local PUL if not available).

For upd:insertAfter, upd:insertIntoAsFirst, the contents of the update primitive are inserted at the beginning of the contents of the update primitive of the same kind in the local PUL (or the update primitive is inserted into the local PUL if not available).
For `upd:delete`, `upd:replaceNode`, the update primitive is inserted into the local PUL if there is not already an update primitive with the same target and the same kind in the local PUL. Otherwise, an error is raised (this should never happen during the normal execution of an XQuery program).

For `upd:replaceValue`, `upd:replaceElementContent`, the update primitive is inserted into the local PUL if there is not already an update primitive with the same target and the same kind in the local PUL. Otherwise, the contents (or string) of the update primitive in the local PUL are replaced with the contents of the update primitive being accumulated.

For `upd:rename`, the update primitive is inserted into the local PUL if there is not already an `upd:rename` update primitive with the same target in the local PUL. Otherwise, the name mentioned in the update primitive of the local PUL is replaced with the name of the update primitive being accumulated.

5. Aggregation

When the target of an update primitive was not here at the last checkout, but was put into place by a former PUL, it means that it is to be found somewhere in the contents of the local PUL (to put it simply, it is encircled in black). In this case, the update primitive is aggregated against the local PUL (Figure 7).

The semantics of aggregation is the same as the semantics of applying a PUL, with just the following modifications for `upd:insertBefore`, `upd:insertAfter` and `upd:replaceNode`.

For these three update primitives, the specification says that the parent property of the target must be non-empty. We relax this constraint. However, if the parent property is empty, the target must be in the contents of an update primitive in the local PUL (encircled in black). This corresponds to (a) on Figure 7.

In case the parent property of the target is empty, the semantics of applying these three update primitives is extended as follows: for `upd:insertBefore` (`upd:insertAfter`), the content of the update primitive is inserted right before (after) the target
in the contents of the update primitive of the local PUL to which this target belongs (like on Figure 7); for *upd:replaceNode*, the content of the update primitive replaces the target of the update primitive of the local PUL to which this target belongs.

### 6. Extension to Apply Expressions

Apply Expressions are introduced in the XQuery Scripting Extension specification. In an apply expression, after each operand is evaluated, the PUL it returns is applied with *upd:applyUpdates*.

We perform PUL composition right here. Whenever a PUL is applied, a copy of it is also composed with the local PUL. This is described on Figure 8.

For nodes which were not here at the last checkout, but were subsequently added, we introduce provenance information in order to know from which update primitive in the local PUL they come from. During the copy of the PUL, these backpointers are followed for each target, whenever possible, and new backpointers are created when copying the contents. On Figure 5, the nodes encircled in black are in the contents of the local PUL.

More precisely, we extend the semantics of Apply Expressions as follows. For each operand, before the PUL is applied:

- The PUL is copied to a new PUL (the contents are copied too), where:
  - Backpointers are put from each node in the contents of the update primitives in the original PUL to the corresponding node in the copied PUL. (a)
  - In this copied PUL, the original targets (which are never in a PUL) are replaced, whenever there is a backpointer to the local PUL, by the corresponding target in the local PUL. (b)
- The copied PUL is then composed with the local PUL

Then the original PUL is applied (with *upd:applyUpdates*, as defined in the XQUF specification).
Note that after this entire process, the original nodes, applied, will be in the store (not in a PUL), whereas the copied nodes, composed, will be in the content of the local PUL. This means that the backpointer always points from a node in the store to the corresponding node in the local PUL (see Figure 8).

7. Conclusion

With this new composition operator on PULs, it is possible to summarize the semantics of several successive PULs in a single one, without introducing any order between the update primitives. For our versioning framework, it means that it is legitimate to assume that the delta between two trees in a timeline can be expressed by a single PUL.

In a separate document, we came up with a formal proof that PUL composition is correct, i.e., the composed PUL has the same semantics on the XML tree than the original sequence of interdependent PULs. This document is available on request.
Bibliography

