Zorba Store API

1. Goals

- Make it possible for Zorba to query a variety of different data sources; e.g., Web pages in the browser, a relational database, an RSS stream. Provide an interface that allows these data sources to produce input data for Zorba.

- Shield all main memory management from the Zorba runtime. Have a single place that controls (and possibly optimizes) main memory management, disk IO, and garbage collection. Furthermore, shield all security and remote data access issues from the Zorba runtime system and its clients. Again, all these issues are encapsulated in the implementation of a store.

- Implement the complete XQuery data model. At the same time, provide implementors of stores the flexibility to use the freedom that the XQuery data model provides; e.g., only store lexical values, only store binary values, or store both.

- Provide functionality for collection and document management.

- Updates to existing data (preserving identity) and creation of new data (node construction).

- Make it possible to implement simple stores as well as sophisticated, highly-tuned stores (e.g., fully streamed, sophisticated pooling of strings, URIs, qnames, clever garbage collection and main memory management, chained IO from disk, etc.).

- Support versioning of data (branching and time travel). Nevertheless, provide well-defined semantics for stores that do not support such versioning.

- Support common XML APIs in order to load XML data into a store:
    1. DOM
    2. SAX
    3. Binary and plain XML as an iostrema
    4. ItemIterator

2. Assumptions

- Zorba engine (runtime + clients consuming XQuery results) will never hold more data than what fits in virtual main memory. Stores do not necessarily provide functionality to "swap" data in and out of main memory.

- There is only one store instance. All new data (nodes and atomic values) must be create using the ItemFactory of that store. Parsing of input data is tightly integrated into the store. If this assumption is violated, it is no longer possible to compare nodes (i.e., different stores have different implementations of node-ids).

- All access to XML data must carried out using the „item“ API provided by the store. In other words, XML items are opaque to clients. In particular, comparisons between

qnames and URIs must be implemented by the implementation of items inside the store.

Todo (known problems, open questions):

- Multi-threading: Currently, the store API is not thread-safe. In particular, the „apply“ method that applies a pending update list to the store needs to be synchronized. Support for multi-threading will be defined as soon as a more global strategy for thread safety has been defined for Zorba. Furthermore, the RC_Object (garbage collection by reference counting) is not thread-safe on certain architectures (need to double-check this).

- Transactions: Currently, no support for „database-style“ locking and synchronization is provided. Such synchronization must be implemented as an additional layer on top of the store.

- Store API currently only provides support for collections and not for documents. Documents can be implemented as a particular kind of collection. It is unclear whether this should be done by the store or by the implementation of the Zorba C API (i.e., the C API that Zorba users use).

- Store API currently does not provide a good way to integrate converters for input data; e.g., CSV -> XML mappings. In order to write such a converter, clients must write an ItemIterator in order to implement the conversion and pass that ItemIterator to the store.

- Indexing: The hooks to implement indexes at compile-time and run-time are not yet implemented.

- Flesh out the details of Requester and TimeTravel in order to support versioned stores.

3. Details

The code of the store API can be found in the following directory:

- .../zorba/xquery/store/api

Example implemenations of the API can be found in the following directories:

- .../zorba/xquery/store/dom
- .../zorba/xquery/store/native

The following (fully abstract) classes are important:

- Store (in store.h)
- Collection (in collection.h)
- Item (in item.h)
- ItemFactory (in item_factory.h)

From the user's perspective, a store provides the functionality to manage collections. That is, the store provides the methods to put a new collection into the store (using various APIs such as DOM, SAX, iostream, and ItemIterator), associate a URI with to a collection, and remove a collection from a store. All this functionality is provided in the „Store" API (see „store.h"). The „Collection" API allows to read (via an ItemIterator) and update a collection (e.g., add and remove items from the collection. The „Item" API allows to read XDM properties from specific items (e.g., the items returned by an ItemIterator as part of a query result). The „ItemFactory" API allows to create new data (i.e., items).

In essence, thus, a „Store" is kind of „Map" from URI to collections and a collection is a „Vector" of items. Both can live in main memory, disk, or on some network device – all this is implementation defined and encapsulated by the store API. For implementers of a store, the most challenging part is the implementation of the „Item" and „ItemFactory" APIs. There are two scenarios:

a.) Implement a new store from scratch: In this case, „items" are implemented according to the requirements of the application and scenario for which the store is designed. For instance, it would be possible to design a non-updateable store that is highly tuned to process (transient) messages. Examples of stores that are built from scratch are the DOM and „native" stores which a shipped with Zorba.

b.) Integrate an existing data source (e.g., an RDBMS): In this case, there is already an existing implementation of „items" as part of the existing data source. In this case, „glue" (or wrapper) items must be implemented which implement the „Item" and „ItemFactory" on top of the existing instances of the data source.

4. Example Stores

The following lists a number of examples that the designers of the store APIs kept in mind and that demonstrate the wide range of stores that are supposed to be pluggable into the Zorba architecture.

- Native, virtual main memory store: (shipped with Zorba)
- DOM store implementation (shipped with Zorba)
- A relational database (e.g., MySQL, SQLite, Oracle, ...)
- Client-server / networked access to remote data (relational database fall into this class)
- DOM in the Web browser (shipped as part of the XBrowser project)
- An XML message stream
- The file system

Contact:

- David Graf: david.graf@28msec.com
- Donald Kossmann: donaldk@ethz.ch
- Tim Kraska: tim.kraska@inf.ethz.ch

First Version: September 9, 2007
Last Change: -