Summa Metadata Storage

bam

May 9, 2006

The Summa Metadata Storage is accessed through the `Access` interface. There are two implementations of this interface. One is based on a postgresql database and the other is based on a Lucene index and is almost identical to the 'Netmusik' Metadata Storage. The `Access` interface has been changed quite a bit, as the delete method now changes the state of the record to deleted, but keeps the record in storage; a new method has been introduced to remove the records which were marked deleted before a given time; and the records now have a base attribute (the origin of the record).

1 Component Interface

The `Access` interface provides methods for creating, updating and deleting records in the database and for retrieving records and iterators over a collection of records.

A few important interpretation decisions:

- `deleteRecord` changes the state of the record to `DELETED`.
- `recordExists` is true if the record exists in storage (may have state `DELETED`).
- `recordActive` is true if the record exists in storage *and* does not have state = `DELETED`.
- `createNewRecord` with a given id is possible if and only if `!recordActive` for this id, i.e. we can create a new record if no record with this id exists or if the existing record has state = `DELETED`.
- `removeDeletedBefore` a given time will remove all records marked `DELETED` before the given time.

See the javadoc for further introduction to the different methods. The classes `Record` and `RecordIterator` are used for returning results. `Record` has been extended with a state and a base, which means that the record now holds an id, a content, a 'last update time stamp', a state and an original base. Only the content can be changed in a record object. The state is either standard or deleted. The `RecordIterator` is a remote iterator. It can be used like any other iterator, but it will make a remote call for each use of the `next()` method.

2 Lucene Index Storage

A few introductory notes:

- When moving or restarting the storage server, remember to update `controllucene.properties` (`directory_name` and `create_new_index`).
- On restart also remember to check (in `/tmp/`) if there is an old write lock and delete it.
- If you want to use IDEA for testing, remember to add the `config` directory to the module libraries in project settings.

The class `ControlLucene` implements the `Access` interface as well as the `ControlLuceneMBean`, which makes adjusting of the Lucene storage possible.

2.1 Internal Design

The implementation has been updated to match the Lucene 1.9.1 API, some correction have been made and some Lucene index parameters have been made available through the MBean.

`ControlLucene` has been designed to postpone the flushing of add operations and the performance of delete operations and do these in batches.

It may still be possible to optimise further. Updating the same record repeatedly is currently very expensive, but I'm guessing we will rarely do this. Updating a number of different records allows us to collect the remove and add operations into batches. Retrieving an iterator is also expensive as it requires first closing the writer (saving all add operations), then saving all remove operations and then closing the writer again to save the add operations created by the `saveRemoves` method.

2.2 Testing

The class `LuceneStorageServer` starts the Lucene Storage as an `Access` server, and the class `LuceneTestClient` should test all methods available from the `Access` interface. The methods available from the MBean interface can be tested from `jconsole` (not if the storage is started from IDEA).

Wed May 03: The small test class has revealed problems, and these have been fixed. The large scale test will probably be done by the Ingest module; unfortunately not before.

2.3 System Requirements and Installation

The Ant build file should provide us with a `lucenestorage.tgz` file in the `dist` directory. This includes the `luceneStorage.sh` script for starting the storage server. The RMIServer and ports should be set in `luceneStorage.sh`. The storage properties should be set in `config/controllucene.properties`, and the log properties should be set in `config/log4j.xml`.

3 Postgres Database Storage

TODO

4 Alternatives, Discussion and Further Work

Regardless of the chosen storage, we might want to change return type of the update methods in the `Access` interface to boolean.

We might want to change `Access` to extend the `Map` interface, in which case we get a `put` method, which will always put the given `Record` (content) into storage associated with the given id (key) and return the 'old' record associated with this id (null if there was no record with this id).