Metadata Front-end for Shore-MT Storage Manager

“A dynamic relational application layer for Shore-MT with metadata management capability”

- EPFL I&C Semester Project, Sept 2012 – Jan 2013
- Student: Bao Duy TRAN (210215)
- Supervisor: Prof Anastasia Ailamaki
- Advisers: Pınar Tözün & Danica Porobic
History

Application layer

$\text{SHORE}$

Storage Manager
Easing Shore-MT’s usage

- Shore-MT: low-level API
  - No restrictions:
    - Interaction patterns
    - Data models
  → Flexibility

- For relational model:
  - Hard-coding
  - Repetition
  - Prototyping by scripting?

Shore-MT metadata **front-end** with scripting support
Project objectives

- Familiarise with Shore-MT
- Investigate SWIG (a cross-language interfacing library)
- Design & develop Shore-MT front-end
- Implement interactive console applications (demos)
- Ensure transferability, maintainability and reusability (Document all features & design decisions)
Project scope

- **Python** as target scripting environment
- Limited set of **operations**
- No **parsers**!
- Sole focus: **Functionality**
Preliminary

Metadata Front-end for Shore-MT Storage Manager

Front-end

Interface

Shore-MT

(application)

Python

C++

SWIG

(application)

Python

C++

Interface

SWIG

Front-end

Shore-MT

(application)

C++

C++
Architecture

Demo application

Relational Metadata Manager

Relational Data Manager

Shore-MT Tasks

Shore-MT Executor Service

Data structures for Relational Data Manager

depends on / uses
Executor Service: A generic utility

Demo application

Demo application

Relational Metadata Manager

Relational Data Manager

Shore-MT Tasks

Shore-MT Executor Service

Data structures for Relational Data Manager

Shore-MT
Functional design

- Synchronous / Asynchronous execution modes
- Proper signalling & synchronisation schemes
Object design

```
ShoreExecutorService
+start(): bool
+stop(): bool
+isRunning(): bool
+syncExec(ShoreTask): bool
+asyncExec(ShoreTask): bool
+setCompletionListener(listener)

std::queue<ShoreTask>

ShoreWorkerThread
+run()
+requestStop()
+setCompletionListener(listener)

<<abstract>>
ShoreTask
+run()
+waitForCompletion()
+notifyCompletion()

<<Shore-NT>>
smthread_t
+run()

<<interface>>
ShoreTaskCompletionListener
+taskCompleted(ShoreTask)
```
void myShoreMtOperation() {
    ss_m::doSomething();
    ss_m::doSomethingElse();
}

void myShoreMtOperation(ShoreExecutorService* myExecSvc) {
    DoTwoThingsTask myTask;
    myExecSvc->syncExec(&myTask);
}

class DoTwoThingsTask : public ShoreTask { … }

void DoTwoThingsTask::run() {
    ss_m::doSomething();
    ss_m::doSomethingElse();
}
Front-end tasks

Diagram:
- Demo application
- Interface
- Relational Metadata Manager
- Relational Data Manager
- Shore-MT Tasks
- Shore-MT Executor Service
- Data structures for Relational Data Manager

Legend:
- Shore-MT Tasks
- Shore-MT
- Demo application
- Interface
Front-end tasks

- **ShoreTask**
  - **BaseTask**
    - **SmTask**
      - CreateSmTask
      - DestroySmTask
      - BeginTransactionTask
      - EndTransactionTask
    - **DevTask**
      - MountDevTask
      - InitDevTask
      - DestroyDevTask
    - **FileTask**
      - CreateFileTask
      - DestroyFileTask
      - CreateRecordTask
    - **RootIndexTask**
      - DestroyRootIndexEntriesTask
      - CreateRootIndexEntryTask
      - RetrieveAllRootIndexTask
Threading strategy

- Single Executor Service
  - Instantiated upon top-level initialisation
  - Passed to subordinates as necessary

  → Single dedicated thread (extensible)

- Synchronous execution mode
Relational Data Manager

Demo application

Interface

Relational Metadata Manager

Relational Data Manager

Shore-MT Tasks

Shore-MT Executor Service

Shore-MT

Data structures for Relational Data Manager

depends on / uses
Functional design

- Databases → tables → tuples → fields
- Schema → field descriptions (name, type, size, nullability)

Operations:
- Initialisation, shut-down
- DB creation/deletion/selection
- Table creation/deletion
- Tuple insertion/retrieval

- At most one database is selected (in use)

- No persistence of relational metadata
RECAP: Shore-MT storage structures

Device [/path/to/dev]

Volume [vol-id]

File [file-id]

Record [rec-id-1]
- header
- body

Record [rec-id-2]
- header
- body

... Record [rec-id-N]
- header
- body

Index [index-id]

- key-1
  - value-1
- key-2
  - value-2
- key-M
  - value-M
Relational model → Storage structures

Device ↔ DATABASE

Volume ↔ (DATABASE)

File ↔ TABLE

Record ↔ TUPLE

Record ↔ TUPLE

Record ↔ TUPLE

Body

...
## SQL → system data types

<table>
<thead>
<tr>
<th>SQL type</th>
<th>C++ type</th>
<th>Description</th>
<th>Single unit?</th>
<th>Fixed length?</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>bool</td>
<td>Boolean (bit)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
<td>Short integer</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>INT</td>
<td>int</td>
<td>Integer</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LONG</td>
<td>long</td>
<td>Long integer</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FLOAT</td>
<td>float</td>
<td>Floating-point</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>double</td>
<td>Double-precision floating-point</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CHAR</td>
<td>char[]</td>
<td>Fixed-length character string</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>VARCHAR</td>
<td>char[]</td>
<td>Variable-length character string</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Relational tuples → Shore-MT records
Data structures
Core object design
Relational Metadata Manager

- Demo application
- Interface
- Relational Metadata Manager
- Relational Data Manager
- Shore-MT Tasks
- Shore-MT Executor Service
- Data structures for Relational Data Manager
Persisting relational metadata

Device [/path/to/master] ↔ MASTER DEVICE

ROOT Index

dbName-1 → path-1

Device [/path/to/dev] ↔ DATABASE

Volume [vol-ID] ↔ (DATABASE)

ROOT Index

tableName-1, meta#tableName-1

File [file-ID] ↔ USER TABLE

Record ↔ TUPLE

File [file-ID] ↔ META-TABLE

Record ↔ TUPLE

file-ID-1

File ID-1

tableName-1

DATABASE

name VARCHAR

type SMALLINT

sizeconfig LONG

nullable BIT

« ??? »
Finalised top-level API
Demo application

- Demo application
- Interface
  - Relational Metadata Manager
  - Relational Data Manager
- Shore-MT Tasks
- Shore-MT Executor Service
- Data structures for Relational Data Manager

Shore-MT

depends on / uses
SWIG-ing into the scripting world

- Demo application
- Interface
- Relational Metadata Manager
- Relational Data Manager
- Shore-MT Tasks
- Shore-MT Executor Service
- Data structures for Relational Data Manager

depends on / uses
Interfacing process

SWIG interface file (.i) -> SWIG -> SWIG glue script (.py)

-> SWIG glue code (.cpp)

-> Python headers (.h)

-> Shore-MT headers (.h)

-> Shore-MT front-end headers (.h)

-> C++ compilation

-> SWIG glue binary (.o)

-> Shore-MT front-end libraries

-> Linking

-> Shore-MT front-end Python extension module (.so)
Conclusions

- **Achievements**
  - Fully functional deliverables
  - Well-documented design & source codes

- **Lessons learnt**
  - Interfacing very efficient if designed with target scripting in mind
  - Modular design with object-oriented approach boosts development
Possible future work

- Separation & enhancement of Executor Service
- Index supports for user data
- SQL operators for prototyping of transaction processing
- SWIG interface for other target languages
- SQL parser
- Multi-threaded transactions
- etc.
Thank you for your attention!