CellStore - the Vision of Pure Object Database

Jan Vraný

27.4.2006
Outline

- CellStore project
- Low-level storage model
  - Cell model
  - Mapping scheme
- CellStore/OODB
  - Object Virtual Machine
- Current prototype
Main goal of CellStore project is to implement an experimental, hybrid (non-relational) database engine

The project consists of three main parts:

- **Low-level storage**
- **CellStore/XML**
- **CellStore/OODB**

CellStore is developed within SWING research group at Department of Computer Science, FEE, CTU
CellStore - architecture

CellStore/OODB

- Jitter
- St. interpreter
- Java interpreter

CellStore/XML

- Bootstrap interpreter
- XQuery/XPath executor
- OODB TM
- XML TM

Low-level storage

- Cache Manager
- Storage Manager

Cell space
Data space
CellStore - architecture (Continued)
Low-level storage I.

- based on LISP idea of fixed-length cells, each cell consists of several fields
- divides the structure form raw data (strings, byte sequences)
- data are stored in two separate spaces:
  - Cell space which contain only structural information
  - Data space which contain raw data as strings, texts, images and so on
Using this model, it is possible to store:

- XML data
- any object structure based on class-instance object model (Smalltalk, Java)
- any object structure based on prototype object model (Self, ECMAscript)
- any relational data

*All mentioned types of data can be stored together in one database instance.*
For each data model a different *mapping schema* must be used. Mapping schema just gives a concrete meanings to the cell fields.

Example of mapping schema for XML tree

- each DOM node is mapped to one cell
- first field contains pointer to parent cell (i.e. parent node)
- second field contains pointer first child cell (i.e. first child node)
- third field contains pointer to sibling cell (i.e. sibling node)
- subsequent field meanings differ with type of cell.
One can thing about OODB as about

- a kind of database engine
  - transactions
  - persistency
  - access control
- a kind of multi-user virtual machine with persistent object memory
Today's virtual machines are unmodifiable, it is difficult to debug them, to experiment with them, to port them to another platform.

- How easy is to modify the garbage collector?
- How easy is to modify the jitter?
- How easy is to change the code semantics (interpeter) ?

One possible solution is to implement as much as possible on the top of light-weight virtual machine.
Requirements for the VM:

- simple object model capable to store any arbitrary object model
- object memory with simple and clean interface
- support for n-code cache (jitter output) and n-code execution
- *bootstrap interpreter* (naive one will be sufficient)

Everything else (full-featured interpreter, GC, jitter etc.) could be implemented on the top of VM
CellStore - architecture (Continued)
Current prototype

We have prototype implementation (in Smalltalk/X):

- **Low level storage**
  - configurable cell space manager
  - naive data space manager
  - prototype implementation of cache and recovery manager

- **CellStore/XML**
  - XML:DB API Core level 0 (modified for Smalltalk language)
  - naive XPath query service (no indexes, no types, no functions)
  - prototype implementation of transaction manager
Current prototype (Continued)

- *Cellstore/OODB* - nothing done
- access control on object (graph-like) structure
- what about long-term transaction and nested transactions
- database distribution, distributed GC
Ongoing development

- refactoring of low-level storage manager
- access control lists
- mapping models
- garbage collector interface
- interpreter interface
- XQuery module
Thank you for your attention