CellStore – the Vision of Pure Object Database

Jan Vraný

Department of Computer Science, FEE, Czech Technical University in Prague,
Karlovo náměstí 13, 120 00, Praha, Czech Republic
vranyj1@fel.cvut.cz

Abstract. This paper describes a vision of CellStore, a kind of universal database system, which would be capable of storing and operating on several different data models – object, network, hierarchical and even relational one. Features of CellStore will be described as well as underlying storage model and database architecture.

1 Motivation

The world's mainstream programming paradigm for robust, large scale, mission-critical application is object-oriented programming (OOP). Many of such applications need support of database to maintain its data. But nobody doubts that the database should be relational or object-relational one. The semantic gap between those two totally different paradigms brings some problems, that has to be solved. Basically, there are three possible solutions:

- The application operates on data in a "relational way", i.e. the programmer has to use SQL queries to access data directly. In this case, usage of objects is limited only to usage of OO libraries for GUI and so on.
- Some kind of object-relational mapper is used (GLORP [5] or Hibernate [6] are examples of such O-R mappers). This allows programmers to manipulate data in a "object" way, but architecture and capabilities of O-R mapper limits the design of application and underlying database schema.
- Network or object database is used instead of relational one.

2 Currently available object-oriented databases

There are currently many so-called "object databases" – OmniBase, DB4Objects, ZODB, GOODS, Elephant, GemStone/S. In fact many of them are network rather than object ones. Both network and object database are very similar. Both can store any arbitrary object structure. The difference is that an object database also stores code (methods) together with regular data. Object database can execute any code stored in it itself, no client environment is needed.

V. Snášel, K. Richta, J. Pokorný (Eds.): Dateso 2006, pp. 32–39, ISBN 80-248-1025-5.

2.1 OmniBase

OmniBase [7] is embedded network database written in smalltalk. It is available for many different smalltalk dialects – Doplhin Smalltalk, Squeak, VisualWorks, Smalltalk/X and VAST. OmniBase supports multi-version concurrency control, object clustering, online backups and thread-safe operations.

Garbage collecting is supported, but cannot be performed on live database.

2.2 DB4Objects

DB4Objects [8] is less or more similar to OmniBase, but there are two differences:

1. DB4Objects is targeted on Java and .NET (C#) platforms.
2. DB4Objects can operate as embedded database or can run as normal database server which communicates with clients over the network.

2.3 GemStone/S

GemStone/S [9] is full-featured object database based on smalltalk dialect called Smalltalk DB. GemStone application consists of three parts: a client (usually VisualWorks smalltalk), a Gem (a part of GemStone responsible for evaluating, transaction processing and so on) and a Stone (a part responsible for managing low-level storage). Each part can run on different node in a network. A special Gem called GcGem is responsible for garbage collecting, which is performed during normal processing of client requests.

3 The CellStore project

The basic motivation for CellStore project was development of an experimental database, which can be used as a basis for experimenting with various database algorithms like locking and caching strategies, transaction policies, different data type models, etc.

The project is divided into three relatively independent parts:

- *CellStore low-level storage*, which provides a basic storage management,
- *CellStore/OODB*, an experimental object database,
- *CellStore/XML*, an experimental native XML database.

The design and implementation of CellStore is focused more on simple OO design and modularity than on implementation performance. As long as CellStore is experimental system, saving several bytes of memory or several processor instructions doesn't matter.

3.1 The low-level storage model

The low-level storage model gave CellStore its name. It is combination of storage models of Lisp, Smalltalk and Oracle RDBM. Basically, the storage is divided into two main spaces.

- *Cell space*, which contains only the structural information about stored data. Structure is kept in fixed-size *cells*. Each cell has several *fields*, which can contain pointer to another cell in the cell space or a pointer to a record in the data space. Cells describe only relationships between data elements (objects) stored in a CellStore database.
- *Data space*, which contain actual data, i.e. a byte arrays. Data space is organised into blocks, each block may contain several *records*. Each record in data space is identified by a unique data pointer. The internal organisation of data space is similar to data blocks in Oracle or in any other relational database.

This approach has several advantages:

1. usage of fixed-length cell simplifies cell allocation and automatic storage reclamation
2. it is possible to store many different object models

The second advantage is a more important one. It allows to store different data (class-based objects, prototype-based objects, XML data and even relational data) together in the same database. Thus CellStore can act as a pure object database or as an XML database. Note that data are stored in their native form, mapping of data to cell and data space is less or more straightforward. This is why we call CellStore a universal database.

Mapping objects into cell and data space In this section, mapping of objects will be described. Consider class-based object model and eight-field cells.

Each object occupies at least one cell, called the *head cell* and zero or more cells called *tail cells*.

The head cell contains cell header, which contains cell type (non-indexable class-based instance for example), other information like the number of cells occupied by this instance, gc support information, tail-cell flag and more.

Second field of the head cell contains pointer to ACL set, pointer to another object (in fact, pointer to another object's head cell), which contains all information needed for access control to this object (because we are designing multi-user database).

Third field of the head cell contains pointer to object's class, which is also an object represented by head cell.

Other fields contain pointers to ordinary instance variables. If all instance variables cannot be stored in a single cell, the last field contains pointer to the next (possibly tail) cell. Another possibility is to use something like indirect pointers as used in inode-based file systems.

Data of indexed classes (arrays, byte arrays) are stored in the data space.

An example of objects structure and its mapping to cell and data space is on figure 1 and figure 2. Note, that integers are stored as immediate values [1].

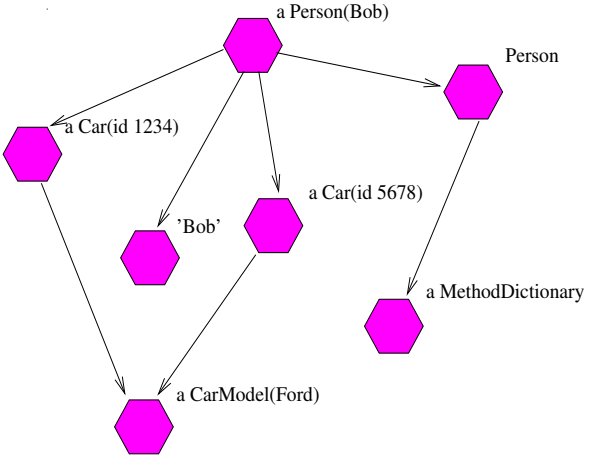

Fig. 1. Example of object structure

Mapping XML data into cell and data space Another example of data that can be stored in CellStore is XML data. Although XML data can be stored into CellStore as normal objects (DOM nodes) as shown above, we are using more efficient, XML specific mapping.

There are 9 types of cells:

- character data cell
- attribute cell
- element cell
- document cell
- document type cell
- processing instruction cell
- comment cell
- xml resource cell
- collection cell

The last two cell types represent XML:DB objects as described in [2]. Each cell has a pointer to its parent cell, first child cell and sibling cell. Meanings of the last four fields depend of the type of the cell (see table 1).

Children of any cell are linked through the sibling pointer and parent holds pointer to the first child.

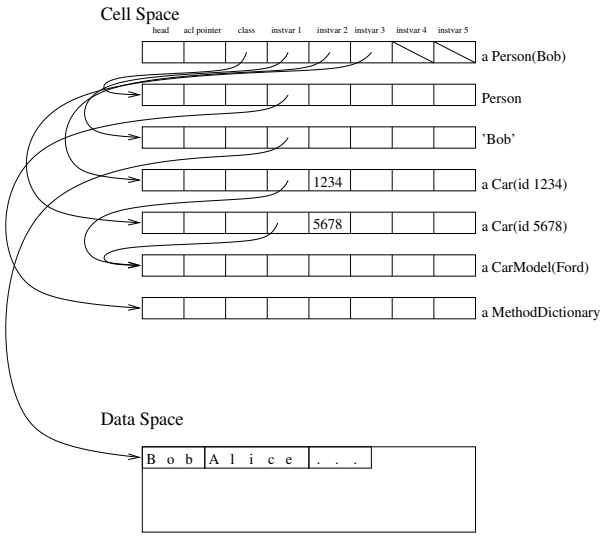

Fig. 2. Example mapping objects into the cell and data space

3.2 The CellStore's virtual machine

Classic virtual machine consists of an object memory and an interpreter. Object memory is responsible for managing objects in memory, for efficient storage reclamation and the interpreter defines all the execution semantics. We think that it's possible to implement virtual machine on the top of CellStore storage, so one can think about CellStore as one large multi-user virtual machine with persistent, transaction-capable object memory.

The idea is to move as much functionality as possible to CellStore's virtual machine. This includes indexing algorithms, garbage collector, jitter etc. The CellStore should provide only basic object memory management and common,

Table 1. Meanings of fields in XML cells

	Field			
Cell type	5	6	7	8
character data	data 1	(data 2)	(data 3)	(data 3)
attribute	local name	namespace qualifier	namespace uri	value
element	local name	namespace qualifier	namespace uri	first attribute
document	document type	encoding	unused	unused
document type	public id	system id	unused	unused
processing instruction	target	data	unused	unused
comment	data 1	(data 2)	(data 3)	(data 3)
xml resource	resource name	unused	unused	unused
collection	name	unused	unused	first resource

flexible object model. Everything else could be implemented on the top of CellStore.

This allows user (programmer) to experiment with different algorithms, jitters, garbage collectors and, as long as the interpreter itself will be implemented on the top of CellStore, with different programming languages and code semantics.

The CellStore's virtual machine should provide only the following:

- object memory management supporting only common object model as described in section 3.1
- dumb, built-in interpreter which is capable if interpreting simple, limited language (bytecode) – we called it the *bootstrap interpreter*
- capability of trap out unknown language (bytecode) and let user-level interpreter to evaluate them.
- basic support for installing native (jitted) code into VM's native code cache

There is no need for speed of any interpreter as long as the interpreter will be able to interpret jitter (implemented in any language). The jitter can translate itself into the native code to make itself fast and then translate the rest.

3.3 Architecture of CellStore database

The high level architecture of CellStore is shown on figure 3.

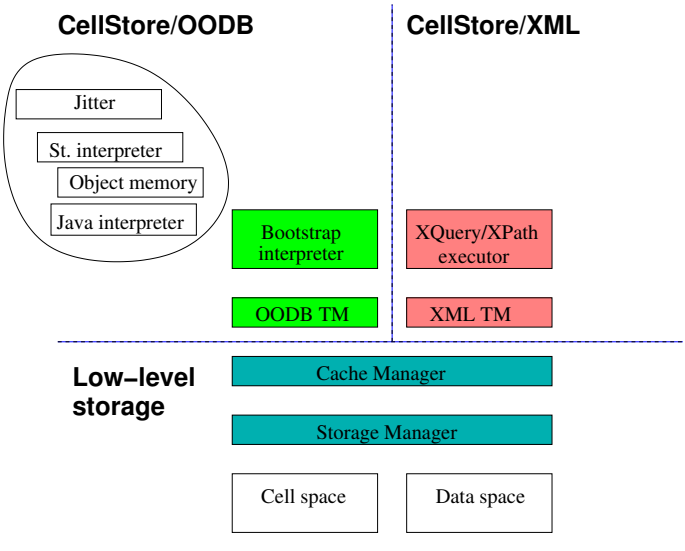

Fig. 3. High level architecture of CellStore

From the VM's side of view, OODB transaction manager plays the role of object memory, so it should provide interface similar to Smalltalk-80's object

memory [1]. In addition, it must provide an interface for transaction managing (start, commit, abort) and an interface for garbage collector.

3.4 Status of the CellStore project

The Cellstore project is developed at Department of Computer Science, FEE CTU Prague by Michal Valenta, Jan Vrany, Pavel Strnad, Karel Prihoda and Jan Zak.

Whole the project is developed in Smalltalk/X – a free smalltalk implementation. Smalltalk/X has been chosen because of its pure object orientation, source code availability, outstanding development tools and because of its extreme agility. To achieve practical performance, system can be translated to C [4].

In these days, only the lowest level storage manager is implemented. It can manage cell and data spaces. First experiments show that the storage is able to store whole INEX database [10] (about 500MB of XML documents) using mapping described in section 3.1 without significant performance lost, that means that the document reconstruction time of single, randomly chosen document n was almost independent on database size.

First versions of cache and XML transaction managers are implemented and tested but they are not integrated to the rest of the system, yet.

4 Conclusion and future work

This paper presented the vision of a pure object database built on the top of CellStore storage model. In CellStore virtual machine, as much components as possible is lifted up to "user-space", making experiments with different languages, semantics, jitter, garbage collectors and other algorithms and techniques very easy.

To make such system working, several things has to be developed:

- OODB transaction manager and its interface to bootstrap interpreter.
- tiny bootstrap interpreter
- experimental, naive one-to-one non optimising jitter
- other language interpreter

Once things mentioned above will be implemented and tested, we will have a working database system, which can be used as test bed for many different algorithms. Such system will make development of new approaches and algorithms very easy.

References

1. A. Goldberg, D. Robson. *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, 1983.

2. XML:DB initiative. *XML:DB Working Draft*, http://xmldb-org.sourceforge.net/xapi/xapi-draft.html
3. Camp Smalltalk. *VM Issues*, http://wiki.cs.uiuc.edu/CampSmalltalk/VM+Issues
4. D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, A. Kay. *Back to the Future. The Story of Squeak, A Practical Smalltalk Written in Itself* http://users.ipa.net/~dwighth/squeak/oopsla_squeak.html
5. Camp Smalltalk. *GLORP: Generic Lightweight Object-Relational Persistence*, http://glorp.org/
6. *Relational Persistence for Java and .NET*, http://www.hibernate.org/
7. *OmniBase*, http://www.gorisek.com
8. *DB4Objects*, http://db4objects.org
9. *GemStone/S*, http://www.gemstone.com
10. *INEX: Initiative for the Evaluation of XML Retrieval*, http://inex.is.informatik.uni-duisburg.de/2006/