The Value of Smalltalk: valuing and risk management in VisualWorks and GemStone

Version 1.1: conference talk

Colin Lewis, JPMorgan Chase

Niall Ross, eXtremeMetaProgrammers

nfr@bigwig.net **colin.t.lewis@jpmorgan.com**

These are the slides of the talk presented by Niall Ross at ESUG 2004 in Kothen, September 6th-10th, 2004.

Overview

- **Kapital Overview**
  - **Domain and Use**
- **How VisualWorks and GemStone deliver Business Value**
  - **Actual Delivery**
  - **Rapid Delivery**
  - **Scalability**
  - **Reengineering**
  - **Migration**
- **Ongoing Issues**
- **Discussion**

Kapital: a *hard* problem

Kapital values and risk-manages a wide range of complex financial products.

- **Commoditised vanilla trades: margin is in volume**
- **New really complex products: margin is in being the first to handle them**

You dare not sell or buy what you cannot price. You dare not hold what you cannot risk-manage. The capacity of your system for doing this determines your volume.

Kapital's goal: "The ease of a spreadsheet with the scalability of a system"

Kapital is used in three modes

- **batch: overnight, batches value all traders' books against immense range of conditions — derive map of possible risks**
- **interactive: businesses manage their trades, value fresh trades, ...**
- **housekeeping: weekend runs clean data, archive, verify, do sanity checks**

Kapital has 60+ in team and more than 500 end-users on 7 sites.

***The* enabler of $Large pa revenues for the Investment Bank.**

The Value of Smalltalk: actual delivery

Kapital's domain (and our understanding of it) changes: needs dynamic domain models.

- **Infrastructure meta-model: graphs, closures, references**
  - **— financial developers don't have to worry about persistence**
  - **— all objects can walk their graph to assure data integrity, audit, ...**
  - **— behaviour in both VW and GemStone**
- **Financial meta-model: markets, trades, cash streams, ...**
  - **— all financial objects can "mark to market" (i.e. value) themselves**
  - **— all financial objects can walk their graph to explain their values, ...**
  - **— domain behaviour in VW only: GemStone is a virtual memory extension**

It took 1.5 years to get these meta-models right. It was *so* worth it.

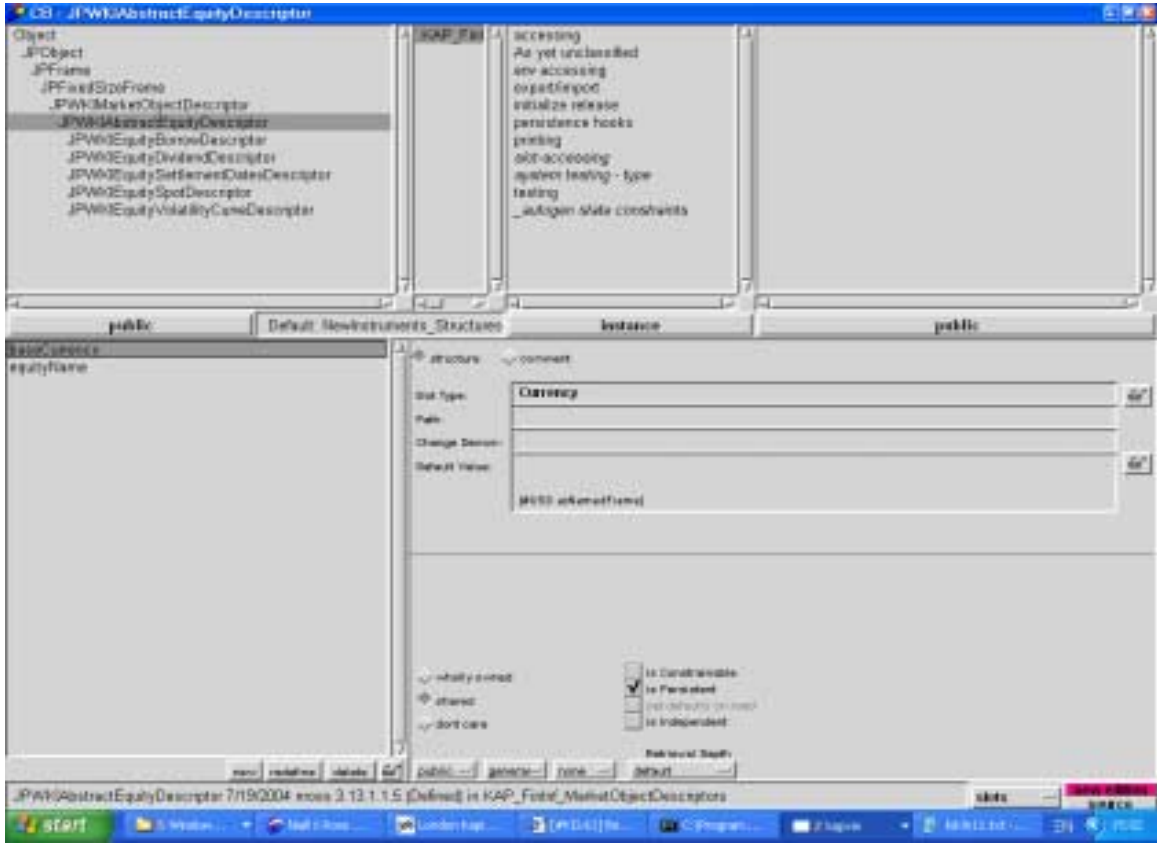
"New financial developers find the ease of the persistence framework unbelievable."

Meta-modelling is so easy in Smalltalk - that you can deliver.

Kapital: modelling the domain

Standard Smalltalk meta-modelling approach: classInstVar 'meta' holds slotDefs, etc.

- **Mix and match what is done by Smalltalk classes and what by 'meta' objects**
  - **— fine-grained refactoring between them as Kapital developed**
- **Easy to integrate specific methods with generic graph-walking methods**
  - **— no type-system obstacles to refactoring between code and 'meta'-code[1]**
- **Tool reuse / refactoring**



Smalltalk exposes its meta-model: everything is an object, few reserved words.

1. See Niall Ross' talk at ESUG99 for detailed analysis of this issue.

The Value of Smalltalk: rapid delivery

Kapital survived DarkPlace Dogytalk, GemStone's Brokat phase and mergers because:

- **can mould it to do what users actually want in volumes they cannot handle elsewhere**
- **frequent prototype demos: ST lets us change things in front of users as we demo**

Example

Not so long ago, *a new financial product* appeared and was very popular in the markets.

- **Every client of every investment bank asked for quotes for it**
- **For *a key period of high market activity*, only JPMorgan Chase quoted for them**

  — competitors worried existing systems could not risk-manage volume new business

  JPMC gained the business, the clients and the premium for being only one to quote.
- **Later**

  — (we believe) competitors used many extra staff and spreadsheets to support volume

  — New product added minimal performance and maintenance costs to Kapital

  JPMC kept lower overheads and freedom of action (opportunity cost).

The Value of Smalltalk: scalability

Visibility is key for scalability: see true bottlenecks in production state.

- **Unlike typical systems of this size (14,000+ classes), Kapital is not stripped at delivery.**
  - **A given release image has the same codebase across all sites and businesses**
  - **Production images have (hidden) development tools**
- **Distribution architecture of Client, Agent and Resource Manager images**
  - **Agents do calculations, Clients task agents and persist results, RMs start images**
  - **Configurations for how clients group tasks for agents: major performance effects**
  - **Stressed DST more than anyone else: vendor fed our fixes back into product**

Common code base and Smalltalk's dynamic features let us *study* bottlenecks.

Examples

Kapital has overcome one performance threshold after another as it has grown.

- **fixed 'Friday slowdown': could never have *guessed* cause; had to *see* it.**
- **on-boarded Credit Hybrid derivatives, wholly new business, faster than competitors**
- **switched from SMP to blade technology in six months: Smalltalk is easy to move**

The Value of Smalltalk: reengineering

Fine-grained access is key to reengineering; you can get at so much in VW and GemStone.

Example Background

Concurrent systems need canonical (i.e. unique) objects. Kapital DBs store their unique graph roots in a performant symbol-keyed cache, each image loading what it needs.

Dates: saving many copies of the same date bloats DBs. Kapital was reengineered to synch image date with unique DB date lazily. (Others do this too, e.g. Joe Bacanskas at WMB.)

Example

Kapital started with 200 financial time-series objects (‘curves’), now uses 70,000

- **retrieving their keys (‘curve descriptors’) began to slow an important UI operation**

  Reengineered to use date-style lazy synchronisation for curve descriptors

Derived Descriptors:

- **slow and costly to gather data for certain curves**

  — new theory: can derive all such curves from a few base curves

  Needed curve descriptors created on the fly. Reengineered to do lazy synch or save.

The Value of Smalltalk: data migration

The persistence framework is integrated with the mutation (data migration) framework.

- **data is lazily 'mutated' as it is loaded into an image, thus migrated if saved**
- **for some changes, one must write #upgrade methods for classes**
- **an amazing amount happens automatically**

Kapital's schema keeps evolving to keep pace with the market.

- **Key financial objects: migrated on each release for all production databases**
- **other data: lazily mutated if loaded**

Thus

- **developers can run the latest codebase against copies of production databases without having to upgrade everything**
- **data upgrade on release takes less time**

Corollary: in GemStone, selected classes (backport classes) push method changes to their previous versions, so only structural changes to such classes force migration.

Ongoing Issues

Things we would like to do better

- **Performant ‘meta’-enabled collection classes**
  - **— at the moment, we clone them :-/)**
- **Eliminate dead code**
  - **— rate of business change means 90Mb image might now have 20Mb dead-code**
  - **— ongoing work on static and dynamic dead code detection and deletion**
- **Eliminate dead data**
  - **— GS object table can be 2 Gig; occupies whole shared cache**
  - **— ongoing work to remove unneeded objects in DB (‘dark matter’)**
    - **– approach is to copy needed to new DB, not hunt unneeded in old DB**
- **Enforce good coding patterns: we have superb code and not-so-superb code**

Discussion

More info on Kapital:

- **notes for talk: http://www.whysmalltalk.com/events/nfrESUG2004report.pdf, page 33**
- **Cincom Smalltalk success stories: www.cincom.com/pdf/CS040819-1.pdf**
- **https://secure.cwheroes.org/briefingroom_2004/pdf_frame/index.asp?id=4909**
- **https://secure.cwheroes.org/briefingroom_2004/pdf_frame/press.asp?id=4909**