Get dirty with CDO

Mathieu Velten
Atos Toulouse Methods & Tools
Quick overview of CDO

- CDO: Connected Data Objects
- EMF-compatible way to store and manage a model and its variants
- No files
- Automatic lazy loading of objects
- Live notifications
Context

► Central architecture model database

► Need to exchange with various tools to edit or analyse the data

► Advanced versioning & branching mechanisms needed to keep the history and be able to create & evaluate some architecture trade-offs

► Currently not implemented but planned & needed: fine grained ACL
Context

Client API

- Hide CDO
- Retrieve and filter data according to user role and targeted tool
- Store IDs mapping between tools & central model

- Final goal: standalone java framework to allow direct integration in tools
Problems encountered

- CDO IDs handling
- Diffing & Merging on CDO
- Optimize performances client-side for our use cases
- IDs mapping:
  - Performances vs memory pressure
  - Database size
- Meta model evolution in the CDO world
- ACL (quick overview, mostly outdated by new luna security model)
Use of client side allocated UUIDs

- **Pro:**
  - Allow to play with final object's id before commit (useful for mapping !)
  - Easier to export in standard EMF XMI format

- **Con:**
  - Slower (how much is untested)
  - Not the most used part of CDO => some shortcomings / bugs
IDs

- Introduced internal API driven by our need: `CDOTransactionImpl.resurrectObject` (`CDOObject`, `CDOID`)

- Trigger a PRIMARY_KEY bug on the database backend (fixed on luna / kepler SR2)

- Some weird behavior with `Resource.getURIFragment/getEObject` regarding U_ prefix
Diffing / Merging

Integration between CDO & EMF Diff/Merge, mainly use resurrectObject in the Merge Policy

3-way merge using ancestor

Currently not optimized: CDO can provide changed objects to narrow down the diff like in the CDO/EMF Compare integration
General performances

- Lazy loading is great, sometimes 😊

- Full load needed for 95% of our operations

- Use of cdoPrefetch(CDORevision.DEPTH_INFINITE) to improve perfs a lot by avoiding network roundtrips

- Bug with prefetch & branches: ineffective when prefetching a branch with nearly no modifications from its branchpoint (436246)
IDs mapping

- Tables needed for mapping between tool model elements and central model elements

- Potentially *..* mapping

- Need to be common between users => stored in CDO, not on the client side
First naive approach: EMF model with some maps-backed eOperations for fast retrieving.
IDs mapping

- Maps became to be memory expensive when dealing with 200 000 elements (2 maps & 2 multimaps needed to handle various *..* use cases)

- Use of SoftReference so the GC can collect the maps if needed
IDs mapping

- Usually one link object needed for each model element

- Really heavy on the database size!
  - History is kept (and not needed)
  - Nearly add up half of the full model size for each mapping
IDs mapping

What to do?
- Store the mapping table in a serialized & compressed way
- Easy in a file-based environment, and (mostly) easy in CDO too!

Please welcome and cheers TextResource & BinaryResource \party/

Can be manipulated easily using standard
InputStream/OutputStream/Reader/Writer
Nearly no changes to the meta model
• Use of EMF-compatible lighter link objects (gain in memory)

• Using JSON for serialization & XZ to compress the text based mapping

• Gain of 300 (~50MB vs ~150KB)

• XZ vs GZIP : big advantage for XZ (highly redondant data) for small speed/memory penalty
IDs mapping

- Trick & tips for implementation:
  - Serialization can be done before commit, easy!

```java
CDOTransaction.
  addTransactionHandler=>committingTransaction
```

- Deserialization is trickier: can be done when an object state changes from TRANSIENT to CLEAN

```java
CDOView. addObjectHandler=>objectStateChanged
```

- Last but not least: disable notifications during load
Meta Model evolution

- Typical problem in model based technologies

- Problem solved when dealing with an unversioned file based model instance: write M2M transformation, launch it, (debug, launch it,)* done!

- How to do in the CDO world with possibly various branches?
First approach:

- Implement hooks in the server administration interface to add/rename features or classes

- Pros: Fast, keep the history clean

- Cons: no real model transformation possible, useful for real simple meta model evolution
Meta Model evolution

Second approach:

- Migrate the branches and their branchpoints using a classical ATL transformation
- Keep the IDs during the transformation

Pros: no real limit to the transformation possibilities

Cons: slow, full history is lost
Meta Model evolution
Easily extensible by providing custom permissions to the security meta model

Simple low level Java interface to validate a permission:
boolean isApplicable(CDORevision, CDOBranchPoint)
ACL

- Still available in Luna

- Much more powerful ACL meta model provided by default on Luna, for more infos:
  https://wiki.eclipse.org/CDO/Security_Manager

- WIP: handle Permissions with CRUD instead of R/W