EMF Models getting XXL?
An overview of available solutions

Jonathan Lasalle
Benoit Viaud
A framework for designing simulations of large & complex systems

Functional requirements

« Excel-like » tables
« Box-and-arrows » diagrams
Metamodel evolution
Query / export in various format
Validation of data
The context
EMF resources serialized into plain XMI files

- Lazy-loading is possible (at the grain of a resource)
- Unloading is a bit more complex...

In the end: all the model in memory

*The -XMX race for memory ...*  
... until you reach the limit

Swap!
Features:
- Easy integration
- Client / Server
- Transaction based
- Branching
- Auditing
- Lazy loading
- Unloading
- Query mode (SQL)
Limitations:
- Insertion cost (Linear but costly)
- Deletion cost (DB schema / XRef cleaning)
- RAM consumption / release (Opaque behavior)

Explanation:
- Client / Server Management
  - Constraints on database schema
  - Several management layers
CDO: what else ?

Ligthweight basic EMF:
- Lazy-loading using resources
- Using Minimal EObjects

Pro:
- Easy to implement
- Light impact on source code

Cons:
- RAM blast: transversal model queries
Improved CDO:
- Merge management layers
- Optimization of database schema

Pro:
- No impacts on source code

Cons:
- Only few experts can do it
- Opacity / Complexity
Teneo:
- Use of Teneo instead of CDO

Pro:
- Out-of-the-box solution

Cons:
- Opacity
- No longer maintained
- No unloading
CDO: what else?

Hibernate / Java Persistence API (JPA)
- Hibernate for data storage
- JPA for data management (POJO)
- Conversion from POJO to EObject

Pro:
- Hand-made solution (to be tailored)

Cons:
- Hand-made solution
Neo EMF:
- Promising solution
- Graph-based database

Pro:
- Out-of-the-box solution

Cons:
- No mature enough
- No more place on the podium...
## Benchmark results

<table>
<thead>
<tr>
<th></th>
<th>CDO ++</th>
<th>Hibernate / JPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import process</td>
<td>1’00”</td>
<td>29”</td>
</tr>
<tr>
<td>(750 000 objects)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Insertion only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update process</td>
<td>10’35”</td>
<td>37”</td>
</tr>
<tr>
<td>(Coupling)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Export</td>
<td>18”</td>
<td>16”</td>
</tr>
<tr>
<td>Delete 1 item</td>
<td>25”</td>
<td>0,002”</td>
</tr>
<tr>
<td>Delete 100 items</td>
<td>7’24”</td>
<td>0,100”</td>
</tr>
</tbody>
</table>
The Hibernate/JPA solution

Database (H2) → JPA Layer → (Texo) Converter → EMF Layer

Data Import

Data Export

Data Deletion

Complex Algorithm

Nebula Editor

Sirius Editor

OCL Validation

Sync

Query

RCP Application

JPA Processes

EMF Processes
How to explain gain: CDO deletion

CDO_Objects
<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>Bus</td>
</tr>
<tr>
<td>S1</td>
<td>Signal</td>
</tr>
<tr>
<td>SP1</td>
<td>SimuParam</td>
</tr>
</tbody>
</table>

Bus
<table>
<thead>
<tr>
<th>ID</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td></td>
</tr>
</tbody>
</table>

Bus_Signal
<table>
<thead>
<tr>
<th>Bus_ID</th>
<th>Signal_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>S1</td>
</tr>
</tbody>
</table>

Signal
<table>
<thead>
<tr>
<th>ID</th>
<th>Container</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>B1</td>
<td></td>
</tr>
</tbody>
</table>

SimuParam
<table>
<thead>
<tr>
<th>ID</th>
<th>Simulate</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP1</td>
<td>S1</td>
<td></td>
</tr>
</tbody>
</table>
**How to explain gain: CDO deletion**

### CDO_Objects

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Signal</td>
</tr>
<tr>
<td>SP1</td>
<td>SimuParam</td>
</tr>
</tbody>
</table>

### Bus

<table>
<thead>
<tr>
<th>ID</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Bus_Signal

<table>
<thead>
<tr>
<th>Bus_ID</th>
<th>Signal_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>S1</td>
</tr>
</tbody>
</table>

### Signal

<table>
<thead>
<tr>
<th>ID</th>
<th>Container</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>B1</td>
</tr>
</tbody>
</table>

### SimuParam

<table>
<thead>
<tr>
<th>ID</th>
<th>Simulate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP1</td>
<td>S1</td>
</tr>
</tbody>
</table>
### How to explain gain: CDO deletion

![Diagram showing relationships between Bus, Signal, and SimulationParameter objects.](image)

#### CDO_Objects

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP1</td>
<td>SimuParam</td>
</tr>
</tbody>
</table>

#### Bus

<table>
<thead>
<tr>
<th>ID</th>
<th>…</th>
</tr>
</thead>
</table>

#### Bus_Signal

<table>
<thead>
<tr>
<th>Bus_ID</th>
<th>Signal_ID</th>
</tr>
</thead>
</table>

#### Signal

<table>
<thead>
<tr>
<th>ID</th>
<th>Container</th>
<th>…</th>
</tr>
</thead>
</table>

#### SimuParam

<table>
<thead>
<tr>
<th>ID</th>
<th>Simulate</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP1</td>
<td>S1</td>
</tr>
</tbody>
</table>
How to explain gain: CDO deletion

- Quite heavy database mechanism
How to explain gain: CDO deletion

- Quite heavy database mechanism

`org.eclipse.emf.cdo.util.DanglingReferenceException`
How to explain gain: CDO deletion

- Quite heavy database mechanism
- Must clean non-containment references (e.g. using CDO XReferencer)
  => Get and build all impacted EObjects (even if they were not loaded)
  => Clean associated features (or delete items)
How to explain gain: Hib/JPA deletion

**Diagram:**
- **Bus** connected to **[0..*] signals**
- **Signal** connected to **[1..1] simulate**

**Tables:**
- **CDO_Objects:**
  - ID: B1, S1, SP1
  - SimuParam
- **Bus**:
  - ID: B1
- **Bus_Signal**:
  - Bus: B1, S1
  - Signal_ID: S1
- **Signal**:
  - ID: S1
  - Container: B1
- **SimuParam**:
  - ID: SP1
  - Simulate: S1
How to explain gain: Hib/JPA deletion

Bus ID
B1

Signal ID
Container
S1 B1

SimulationParameter ID
Simulate
SP1 S1

[0..*] signals

[1..1] simulate

Foreign Key

FK
How to explain gain: Hib/JPA deletion
How to explain gain: Hib/JPA deletion

Foreign Key

Bus
ID...×
×...

Signal
ID...×
×...

SimulationParameter
IDSimulate...
SP1S1

How to explain gain: Hib/JPA deletion

Foreign Key

Bus
ID...×
×...

Signal
ID...×
×...

SimulationParameter
IDSimulate...
SP1S1
How to explain gain: Hib/JPA deletion
- Optimized database mechanism (using Hibernate native features)
- Non-containment references automatically cleaned (or deleted)
- Must synchronize only loaded EObjects
• Success recipe:
  – Take a JPA-compliant EMF Metamodel
  – Add a pinch of Texo
  – Spread it on an Hibernate pastry
• Success recipe:
  – Take a JPA-compliant EMF Metamodel
  – Add a pinch of Texo
  – Spread it on an Hibernate pastry
  – For a faster deletion:
    stuff it with a maximum of foreign keys
  – Stir it with a transparent implementation
    of a lazy loading mechanism
  – Filter the unused EObject from RAM
• Success recipe:
  – Take a JPA-compliant EMF Metamodel
  – Add a pinch of Texo
  – Spread it on an Hibernate pastry
  – For a faster deletion:
    stuff it with a maximum of foreign keys
  – Stir it with a transparent implementation
    of a lazy loading mechanism
  – Filter the unused EObject from RAM
  – Dust some query-DSL to lighten
    some algorithms
The Result: a technical framework

- Sirius Connectors
- EMF Layer
  - EObject Cache
  - Lazy Mode
  - Converter
  - Sharing
- Database (H2)
- JPA Layer
  - Deletion Helper
  - Query API
- Converter
The Result: a technical framework

- EObject Cache
- Sharing
- Lazy Mode
- Converter
- EMF Layer

Connectors
- Sirius

Database (H2)

JPA Layer
- Help
- Query API

+ A LOT OF GOOD PRACTICES
Step 0: Identify your requirements regarding perf

- Consider using CDO if
  - You need out-of-the-box advanced features: concurrent access, branching & merging, etc
  - You don’t need extra performances

- Consider using a Hibernate/JPA-flavor solution if
  - Your context allows to release some constraints on features
  - Performances are never good enough...
Step 1: Introduce the technical framework

Cost: low
- Very low if resources management is already identified

Benefits
- Lazy loading
- Faster Deletion => reduce CPU usage
Step 2.1: Refactor the Meta-Model

Cost: low to medium
- If you know what you are doing/expecting

Benefits
- Unloading => reduce memory usage
- Extra-fast deletion
Step 2.2: Refactor algorithms & HMI

Cost: from low to high
- Requires dual expertise: Hibernate/JPA & EMF/TEXO

Benefits
- Allow creation of data at lower level
- Allow optimized queries *alla* SQL
  - *with the support of higher level language Query DSL*
What we can do for

- **Related Technology providers:**
  - Publish (EPL) our benchmark challenge & framework

  Propose *your* solution

- **S/W products developers / integrators:**
  - Publish our solution as a ready-to-use package

  Test *your* requirements

Contact us to go further
Evaluate the Sessions

Sign in and vote at eclipsecon.org