Connected Data Objects (CDO)

The EMF Model Repository
Agenda

- Introduction \( \leq 5 \) minutes
- Live Demonstrations \( \leq 15 \) minutes
- Detailed Architecture \( \leq 5 \) minutes
- Programming Examples \( \leq 10 \) minutes
- Advanced Features \( \leq 5 \) minutes
- Open Discussion \( \geq 10 \) minutes

\[ \text{~ 50 minutes} \]
Introduction

• About the Author

• EMF Intro
  ♦ EMF Persistence Framework
  ♦ Issues with XML Files

• Distributed Shared Models

• What is CDO About?
About the Author

• Eike Stepper, Germany, Berlin
  ♦ Born in 1970
  ♦ Started programming in 1983
  ♦ Studied mathematics and computer science
  ♦ Founded first company ES-Computersysteme in 1991
  ♦ Consulting in dozens of IT projects
  ♦ First orthogonally persistent system in 2000 (C++)
• First version of CDO in 2003
  ♦ Contribution of CDO to Eclipse.org in 2004
  ♦ Complete rewrite with new design in 2007
EMF Intro

• With EMF you can (out of the box)
  ✷ Create Ecore models
  ✷ Configure generator models
  ✷ Generate Java code for
    ▪ Your Ecore model
    ▪ Command framework
    ▪ Eclipse UI (creation wizard and model editor)
  ✷ Use the EMF persistence framework to
    ▪ Serialize model instances to XML files
    ▪ Deserialize model instances from XML files
    ▪ Resolve model references across files
  ✷ And many, many other things…
EMF Persistence Framework

- **Resource**
  - A named container for model instances
  - URI + Contents
- **Resource.Factory**
  - Creates specialized resource instances
  - Default is XML / XMI
- **ResourceSet**
  - Container for a set of resource instances
  - Package registry for resolving model references
  - URI converter for resolving resource URIs
Issues with XML Files

• Limited resource size
  ▮ No lazy loading of instances
  ▮ No lazy loading of lists

• No unloading of instances
  ▮ Bad influence on garbage collection
  ▮ Influence on model design (containment)

• No concurrent modification of resources
  ▮ No fine grained locking
  ▮ No transactions
  ▮ No remote update notification

• Just don’t behave like multi user databases
Distributed Shared Models

• Central persistent model repository
  - Contains all models (packages and classes)
  - Contains all instances (resources and regular objects)
  - Represents a potentially huge object graph in form of containment trees scattered across resources
  - Manages remote client sessions

• Multiple remote clients share a common view of the central persistent models and instances
  - Represent partial views of the overall object graph
  - Concurrently alter the state of the object graph
  - Are immediately notified about modifications that happened in the context of other sessions
What is CDO About?

• Overcomes all the issues with XML files
• Provides distributed shared models for EMF
• Integrates with the EMF persistence framework
• Uses Net4j to implement a network protocol
• Configures multiple repositories on the server
• Connects with heterogeneous back ends
• Uses OSGi at client and server side
• By the way
  • “Connected” indicates that objects in a client session always stay connected with their repository pendants
live Demonstrations

• Developing a CDO Model

• Setting Up a CDO Server

• Using the CDO Client
Developing a CDO Model

• Create an Ecore model
  ▶ Just as you are used to it
  ▶ No additional expenses to be met

• Derive a generator model
  ▶ Use the CDO Importer or the CDO Migrator
  ▶ Do it manually
Setting Up a CDO Server

**Product Definition**
This section describes general information about the product.

**Product Name:** CDO Server

**Product ID:** org.eclipse.emf.cdo.server.product

**Application:** org.eclipse.emf.cdo.server.app

*The product configuration* is based on: plug-ins, features.
Setting Up a CDO Server

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cdcServer>
  <acceptor type="tcp" listenAddr="0.0.0.0" port="2036">
    <negotiator type="challenge" description="/temp/users.db"/>
  </acceptor>
  <repository name="repo1">
    <!-- ... -->
  </repository>
</cdcServer>
```
Setting Up a CDO Server

```xml
cdo-server.xml

1<?xml version="1.0" encoding="UTF-8"?>
2<cdoServer>
3
4  <repository name="repol">
5    <property name="overrideUUID" value="1ff5d226-b1f0-40fb-aba2-0c31b38c764f"/>
6    <property name="supportingAudits" value="true"/>
7    <property name="verifyingRevisions" value="false"/>
8    <property name="currentLRUCapacity" value="10000"/>
9    <property name="revisedLRUCapacity" value="100"/>

10  <store type="db">
11    <mappingStrategy type="horizontal">
12      <property name="toManyReferences" value="ONE_TABLE_PER_REFERENCE"/>
13      <property name="toOneReferences" value="LIKE_ATTRIBUTES"/>
14      <property name="mappingPrecedence" value="MODEL"/>
15    </mappingStrategy>
16    <dataSource class="org.apache.derby.jdbc.EmbeddedDataSource">
17      databaseName="/temp/cdodb1"
18      createDatabase="create"/
19    </dataSource>
20  </store>
21
22 </repository>
23
24 </cdoServer>
```
Setting Up a CDO Server

```
CDOServer [Eclipse Application] C:\Program Files\Java\jdk1.5.0_14\bin\javaw.exe (17.02.2008 09:59:41)

org.eclipse.net4j.internal.util.security.ChallengeNegotiatorConfigurer@30d82d
org.eclipse.internal.net4j.Net4jTransportInjector@c09554
org.eclipse.net4j.internal.tcp.TCPSelectorInjector@18bf072
Acceptor.receiveExecutor = java.util.concurrent.ThreadPoolExecutor@9a8a68
Acceptor.lifecycleEventConverter = org.eclipse.internal.net4j.acceptor.Acceptor$1@1f4e571
Acceptor.acceptedConnectors =
selector = TCPSelector
selectionKey = sun.nio.ch.SelectionKeyImpl@1d8c528
startSynchronously = true
synchronousStartTimeout = 4000
startLatch = java.util.concurrent.CountDownLatch@5976c2[Count = 0]
serverSocketChannel = sun.nio.ch.ServerSocketChannelImpl[/0.0.0.0:2036]
address = 0.0.0.0
port = 2036

[INFO] CDO Server started
Application Started: 5016
```
Using the CDO Client
Using the CDO Client

![Screenshot of CDO Client interface](image)
Using the CDO Client
Using the CDO Client
Using the CDO Client
Using the CDO Client
Detailed Architecture

• Deployment Options
  ¬ Networked Remote Server
  ¬ Embedded Server

• Static Decomposition
  ¬ Server Components
  ¬ Client Components

• Component Interaction
  ¬ Committing a Transaction
  ¬ Demand Loading Objects
Deployment Options - Networked

- Client Applications
  - CDO Client
  - CDO Protocol
  - Net4j TCP
  - Net4j
  - OSGi / Eclipse

- CDO Store
  - CDO Server
  - CDO Protocol
  - Net4j TCP
  - Net4j
  - OSGi / Eclipse

- Backend
Deployment Options - Embedded

- Client Applications
- CDO Store
- CDO Client
- CDO Server
- CDO Protocol
- Net4j JVM
- Net4j
- OSGi / Eclipse

Backend
Detailed Architecture

Static Decomposition - Server

IREpository -> IStore

IPackageManager -> CDOPackages
IRevisionManager -> CDORevisions
IResourceManager -> Path Mappings
ISessionManager -> ISessions

IStore -> IStoreAccessors

creates
Static Decomposition: Client

CDOSession

CDOPackageManager → CDOPackages

CDOREvisionManager → CDORevisions

CDOViews

CDOTransactions → Deltas

extends
Component Interaction – CDOStateMachine (1)

- **PERSISTENT**
  - TRANSIENT
  - NEW
  - DIRTY
  - CLEAN
  - PROXY
  - CONFLICT

- **TRANSIENT**
  - attach
  - detach

- **NEW**
  - commit

- **DIRTY**
  - write

- **CLEAN**
  - read
  - invalidate

- **PROXY**
  - reload
Component Interaction – CDOStateMachine (2)

**Detailed Architecture**

<table>
<thead>
<tr>
<th>State</th>
<th>CDOID</th>
<th>CDOView</th>
<th>CDORevision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TRANSIENT</strong></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>(state is in object)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NEW</strong></td>
<td>With Temp</td>
<td>With</td>
<td>With</td>
</tr>
<tr>
<td></td>
<td>CDOID</td>
<td>CDOView</td>
<td>CDORevision</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(state is in revision)</td>
<td></td>
</tr>
<tr>
<td><strong>PERSISTENT</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Component Interaction - Committing

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client adds/modifies CDOObjects</strong></td>
<td><strong>Server passes data to the configured store</strong></td>
</tr>
<tr>
<td><strong>Client transaction creates temporary IDs for new objects and records change deltas</strong></td>
<td><strong>Store remaps temporary IDs and persists the data</strong></td>
</tr>
<tr>
<td><strong>Commit() sends new packages, new revisions and revision deltas to the server</strong></td>
<td><strong>Server sends back ID mappings</strong></td>
</tr>
<tr>
<td><strong>Server notifies other sessions about invalidations</strong></td>
<td><strong>Client transaction applies ID mappings</strong></td>
</tr>
</tbody>
</table>
## Component Interaction – Demand Loading

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client accesses an EReference</td>
<td>IRevisionManager looks up CDOResRevision</td>
<td>Creates new CDOObject, links it with revision, finished</td>
</tr>
<tr>
<td>CDOResRevision delivers target CDOID</td>
<td>If not found → Loads CDOResRevision from IStore and caches it</td>
<td></td>
</tr>
<tr>
<td>CDOView looks up target CDOObject</td>
<td>IRevisionManager sends back CDOResRevision to client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDOResRevisionManager looks up CDOResRevision</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDOResRevisionManager sends LoadRevisionRequest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDOResRevisionManager caches CDOResRevision</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Found → Creates new CDOObject, links it with revision, finished</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Found → Finished</td>
</tr>
</tbody>
</table>

- Found → Finished
- CDOResRevisionManager looks up CDOResRevision
  - Found → Creates new CDOObject, links it with revision, finished
  - CDOResRevisionManager sends LoadRevisionRequest
- IRevisionManager looks up CDOResRevision
  - If not found → Loads CDOResRevision from IStore and caches it
  - IRevisionManager sends back CDOResRevision to client
- CDOResRevisionManager caches CDOResRevision
  - Creates new CDOObject, links it with revision, finished
Programming

• Using a Managed Container

• Using the Server API

• Using the Client API
Using a Managed Container (1)

```java
public interface IManagedContainer extends IContainer<Object> {
    public IRegistry<IFactoryKey, IFactory> getFactoryRegistry();
    public IManagedContainer registerFactory(IFactory factory);

    public List<IElementProcessor> getPostProcessors();
    public void addPostProcessor(IElementProcessor postProcessor, boolean processExistingElements);
    public void addPostProcessor(IElementProcessor postProcessor);
    public void removePostProcessor(IElementProcessor postProcessor);

    public Set<String> getProductGroups();
    public Set<String> getFactoryTypes(String productGroup);
    public IFactory getFactory(String productGroup, String factoryType);

    public Object putElement(String productGroup, String factoryType, String description, Object element);
    public Object removeElement(String productGroup, String factoryType, String description);
    public Object getElement(String productGroup, String factoryType, String description);
    public Object[] getElements(String productGroup, String factoryType);
    public String[] getElementKey(Object element);

    public void clearElements();
    public void loadElements(InputStream stream) throws IOException;
    public void saveElements(OutputStream stream) throws IOException;
}
```
Using a Managed Container (2)

```xml
<plugin>
  <extension point="org.eclipse.net4j.util.factories">
    <factory class="org.eclipse.net4j.internal.tcp.TCPAcceptorFactory"
             productGroup="org.eclipse.net4j.acceptors"
             type="tcp"/>
    <factory class="org.eclipse.net4j.internal.tcp.TCPConnectorFactory"
             productGroup="org.eclipse.net4j.connectors"
             type="tcp"/>
    <factory class="org.eclipse.net4j.internal.tcp.TCPSelectorFactory"
             productGroup="org.eclipse.net4j.selectors"
             type="tcp"/>
  </extension>

  <extension point="org.eclipse.net4j.util.elementProcessors">
    <elementProcessor class="org.eclipse.net4j.internal.tcp.TCPSelectorInjector"/>
  </extension>
</plugin>
```
Using a Managed Container (3)

1. IManagedContainer container = IPluginContainer.INSTANCE;
2. IConnector connector = (IConnector)container.getElement(
3.   "org.eclipse.net4j.connectors",
4.   "tcp",
5.   "localhost:2036");

> TCPSelector [debug.lifecycle.dump] DUMP TCPClientConnector@8
>     Connector.userID = null
>     Connector.negotiator = null
>     Connector.negotiationContext = null
>     Connector.bufferProvider = BufferPool[4.096]
>     Connector.receiveExecutor = java.util.concurrent.ThreadPoolExecutor@dd7404
>     Connector.nextChannelID = 1
>     Connector.connectorState = CONNECTED
>     TCPConnector.selector = TCPSelector
>     TCPConnector.controlChannel = Channel[Control]
>     TCPConnector.host = localhost
>     TCPConnector.port = 2036
Using a Managed Container (4)

// Turn on tracing
OMPlatform.INSTANCE.setDebugging(true);

// Prepare the standalone infra structure
// Not needed when running inside Eclipse
IManagedContainer container = ContainerUtil.createContainer();

Net4jUtil.prepareContainer(container); // Prepare the Net4j kernel
JVMUtil.prepareContainer(container);  // Prepare the JVM transport
CDOServerUtil.prepareContainer(container); // Prepare the CDO server
CDOUtil.prepareContainer(container, false); // Prepare the CDO client

// Start the JVM transport
IAcceptor acceptor = JVMUtil.getAcceptor(container, "default");

// Open a JVM connection
IConnector connector = JVMUtil.getConnector(container, "default");
// Prepare store parameters
IMappingStrategy strategy = CDODBUtil.createMappingStrategy("horizontal");
IDBAdapter adapter = DBUtil.getDBAdapter("mysql");
IConnectionProvider provider = DBUtil.createConnectionProvider(dataSource);

// Create a DBStore
IStore store = CDODBUtil.createStore(strategy, adapter, provider);

// Create a repository
Map<String, String> props = new HashMap<String, String>();
props.put(Props.PROP_SUPPORTING_REVISION_DELTAS, "true");
props.put(Props.PROP_CURRENT_LRU_CAPACITY, "10000");
props.put(Props.PROP_REVISED_LRU_CAPACITY, "10000");
IRepository repository = CDOServerUtil.createRepository("repo", store, props);

// Start the repository
CDOServerUtil.addRepository(container, repository);
Using the Client API

// Open an embedded connection
IConnector connector = JVMUtil.getConnector(container, "default");

// Open a session and register the model
CDOUtil.openSession(connector, "repo", true);
session.getPackageRegistry().putEPackage(Model1Package.eINSTANCE);

// Start a transaction and create a resource
CDOTransaction transaction = session.openTransaction();
Resource resource = transaction.createResource("/my/big/resource");

// Work normally with the EMF resource
resource.getContents().add(getInputModel());
transaction.commit();

// Cleanup
session.close();
connector.disconnect();
Advanced Features

• Models

• Optimizations

• Network Protocol

• Server Side

• DB Store
Models

• Support for dynamic models
  - just load .ecore file and commit to repository

• Support for legacy models
  - for compiled models without access to .genmodel
Optimizations

• Sharing of objects between views/transactions
  - Modeled state resides in the session

• Demand loading and unloading of objects
  - Containment does not prevent laziness

• Transmission of only change deltas
  - Currently from client to server

• Partial collection loading (chunking)

• Adaptable object pre-fetching
  - Configurable per view
  - Intelligent model usage analyzers
  - Optionally done in background
Network Protocol

- Net4j based binary application protocol
  - Buffered, non-blocking, asynchronous
- Pluggable transport layer
  - NIO socket transport
  - JVM embedded transport
- Pluggable fail over support
- Pluggable authentication
  - Challenge/response negotiation
- Multiple acceptors per server
Server Side

• Multiple repositories per server
  ✷ Configurable storage adapter per repository
    ▪ Shipped with JDBC based O/R mapping adapter
    ▪ Known to work with an Objectivity OODB adapter
    ▪ Work on a Hibernate adapter is underway
  ✷ Configurable caching per repository

• Supported Environments
  ✷ OSGi and Eclipse
  ✷ Standalone applications
DB Store

• Supports the auditing mode of the repository

• Pluggable mapping strategies
  ▪ Horizontal mappings
  ▪ Vertical mappings
  ▪ Different mapping modes for collections

• Pluggable SQL dialect adapters
  ▪ Derby adapter
  ▪ Mysql adapter
  ▪ HsqlDB adapter
Open Discussion

Thank you for listening!

http://wiki.eclipse.org/CDO
http://wiki.eclipse.org/Net4j

Questions?

Comments?

Suggestions?