eclipseCON™ 2008

Connected Data Objects (CDO)

The EMF Model Repository

Second Floor

Tuesday, 14:30, 50 minutes | Theatre |

7 · 8 · 9 · 10 · 11 · 12 · 13 · 14 · 15 · 16 · 17 · 18 · 19

Agenda

• Introduction	≤ 5 minutes
• Live Demonstrations	≤ 15 minutes
• Detailed Architecture	≤ 5 minutes
• Programming Examples	≤ 10 minutes
• Advanced Features	≤ 5 minutes
• Open Discussion	≥ 10 minutes
	~ 50 minutes

≤ 5 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

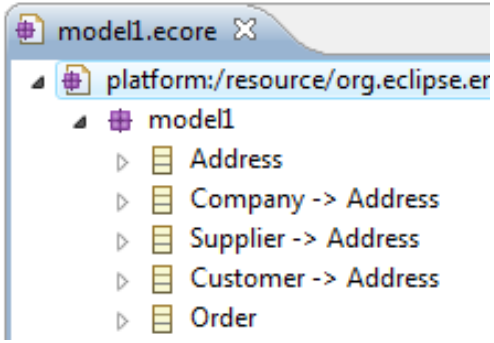
≤ 15 minutes

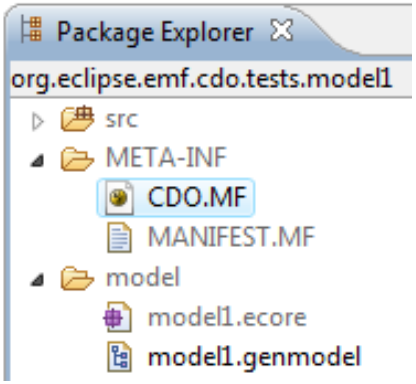
Live Demonstrations

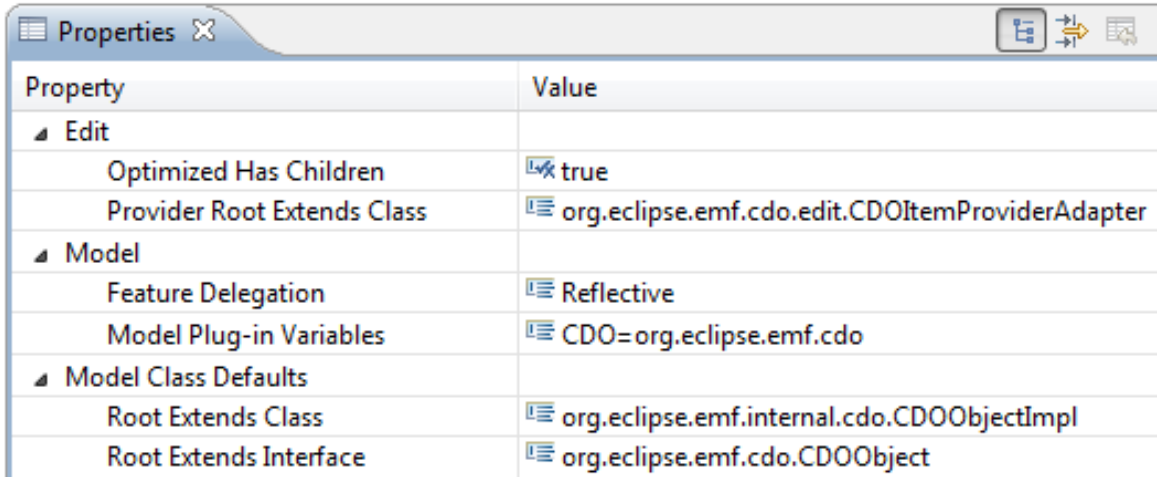
- Developing a CDO Model
- Setting Up a CDO Server
- Using the CDO Client

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

CDOServer.product

Overview

Product Definition

This section describes general information about the product.

Specify the name that appears in the title bar of the application.

Product Name: CDO Server

Specify the product identifier.

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

Specify the application to run when launching this product.

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

The product configuration is based on: plug-ins features

CDOServer.product

Configuration

Features

List the features that constitute the product.

- org.eclipse.emf.cdo.server.product
- org.eclipse.net4j
- org.eclipse.net4j.db
- org.eclipse.emf.cdo.server

Setting Up a CDO Server

cdo-server.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <cdoServer>
3
4   <acceptor type="tcp" listenAddr="0.0.0.0" port="2036">
5     <negotiator type="challenge" description="/temp/users.db"/>
6   </acceptor>
7
8   <repository name="repo1">
9     <!-- ... -->
10
```


Setting Up a CDO Server

cdo-server.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cdoServer>

  <repository name="repo1">
    <property name="overrideUUID" value="1ff5d226-b1f0-40fb-aba2-0c31b38c764f"/>
    <property name="supportingAudits" value="true"/>
    <property name="verifyingRevisions" value="false"/>
    <property name="currentLRUCapacity" value="10000"/>
    <property name="revisedLRUCapacity" value="100"/>

    <store type="db">
      <mappingStrategy type="horizontal">
        <property name="toManyReferences" value="ONE_TABLE_PER_REFERENCE"/>
        <property name="toOneReferences" value="LIKE_ATTRIBUTES"/>
        <property name="mappingPrecedence" value="MODEL"/>
      </mappingStrategy>

      <dataSource class="org.apache.derby.jdbc.EmbeddedDataSource"
        databaseName="/temp/cdodb1"
        createDatabase="create"/>
    </store>
  </repository>

</cdoServer>
```

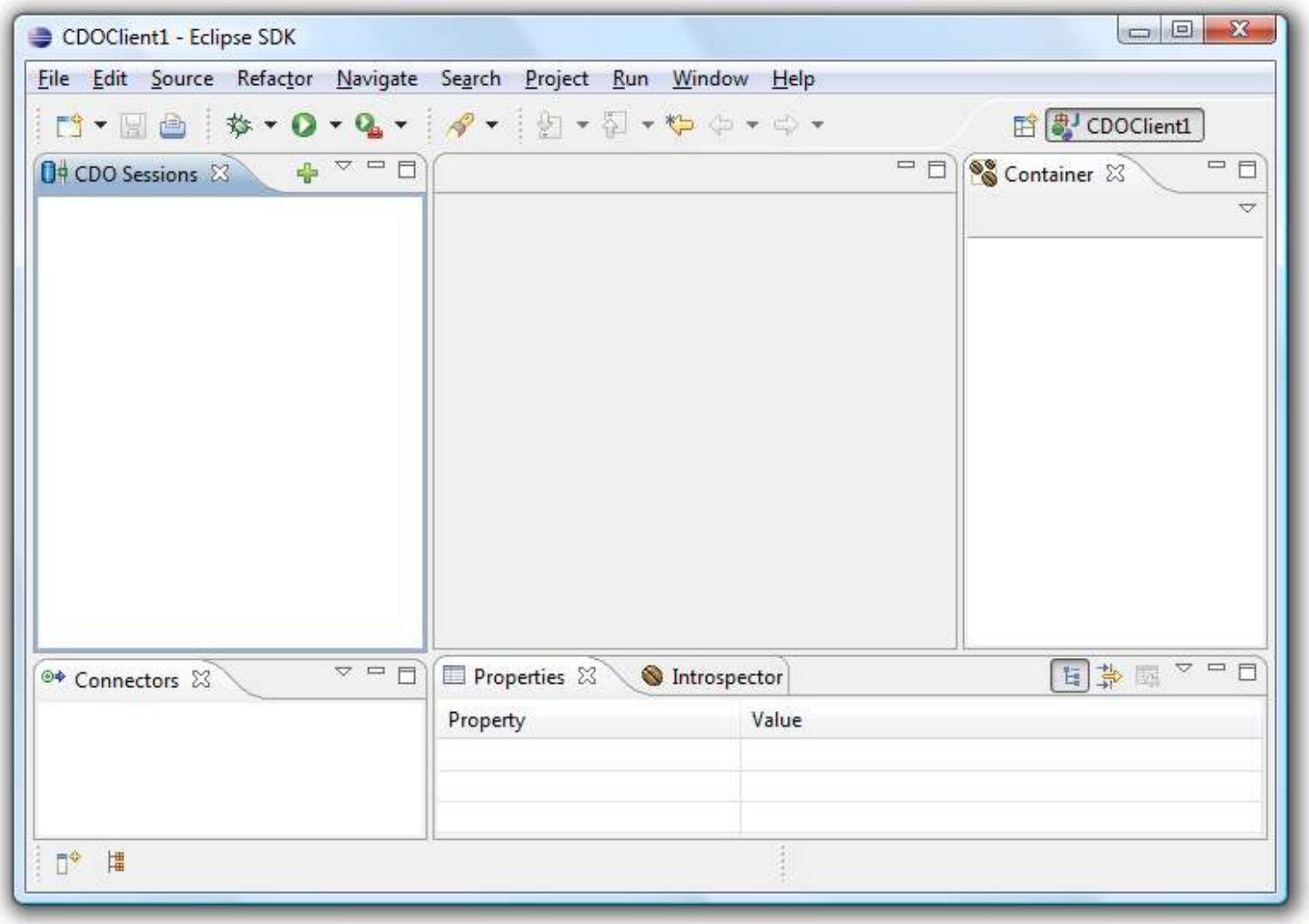

Setting Up a CDO Server

Console

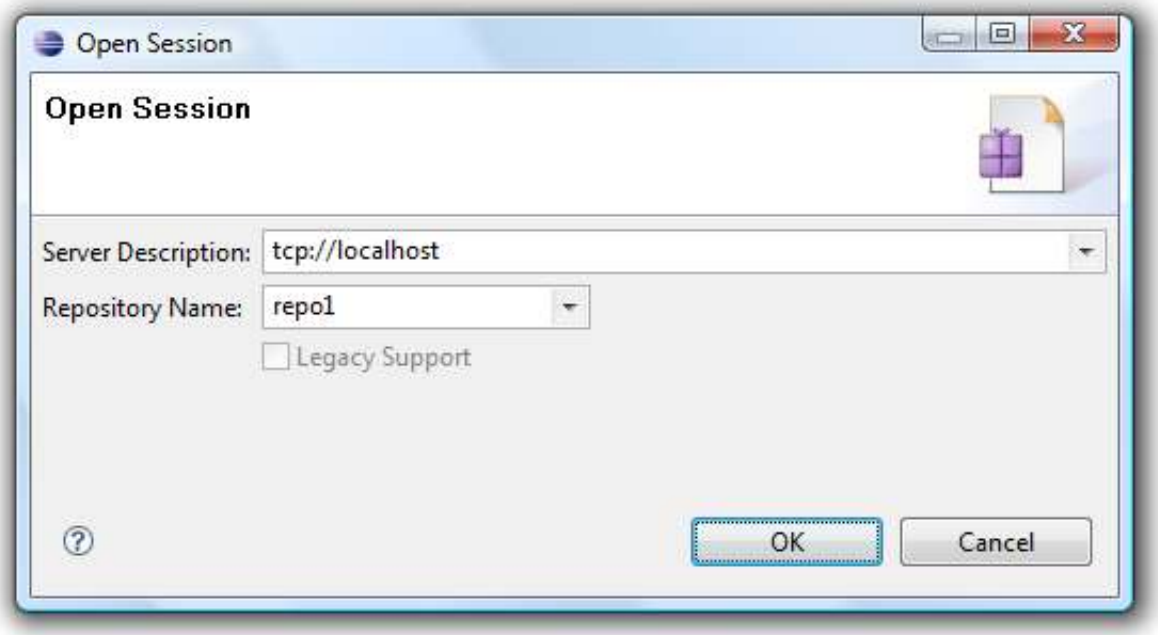
```
CDOServer [Eclipse Application] C:\Program Files\Java\jdk1.5.0_14\bin\javaw.exe (17.02.2008 09:59:41)
  org.eclipse.net4j.internal.util.security.ChallengeNegotiatorConfigurer@30d82d
  org.eclipse.internal.net4j.Net4jTransportInjector@c09554
  org.eclipse.net4j.internal.tcp.TCPSelectorInjector@18bf072
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
```

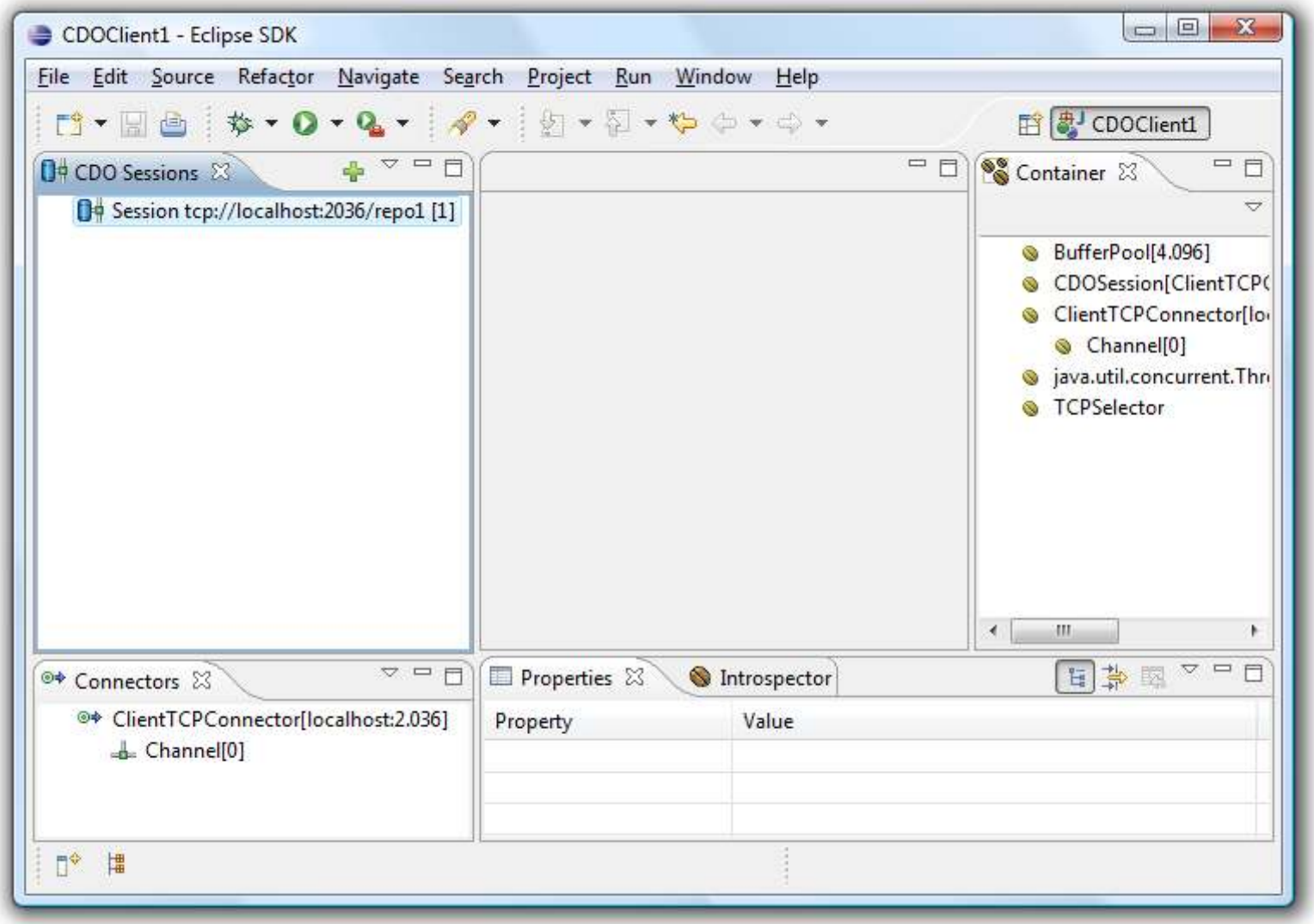

Using the CDO Client



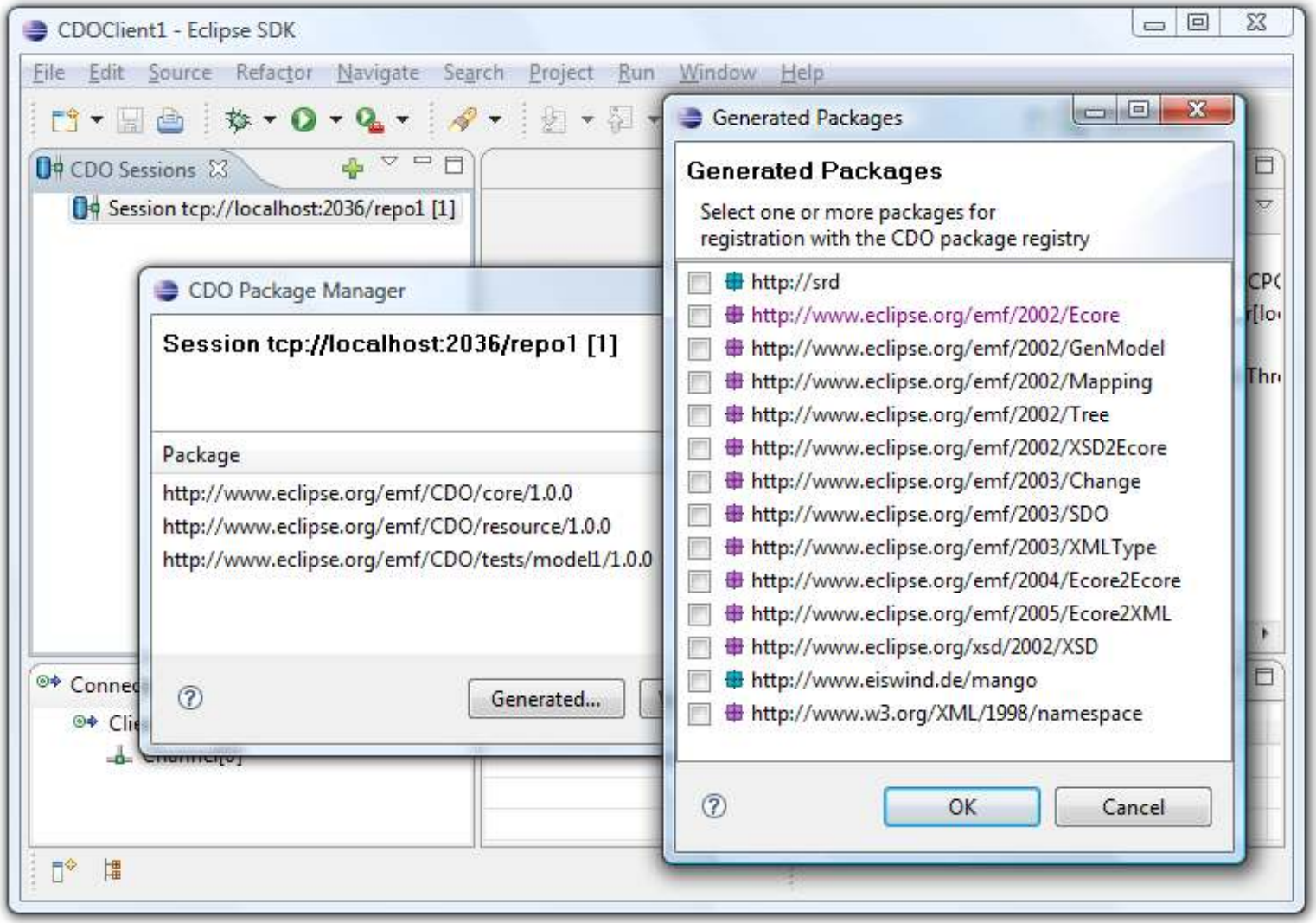
Using the CDO Client



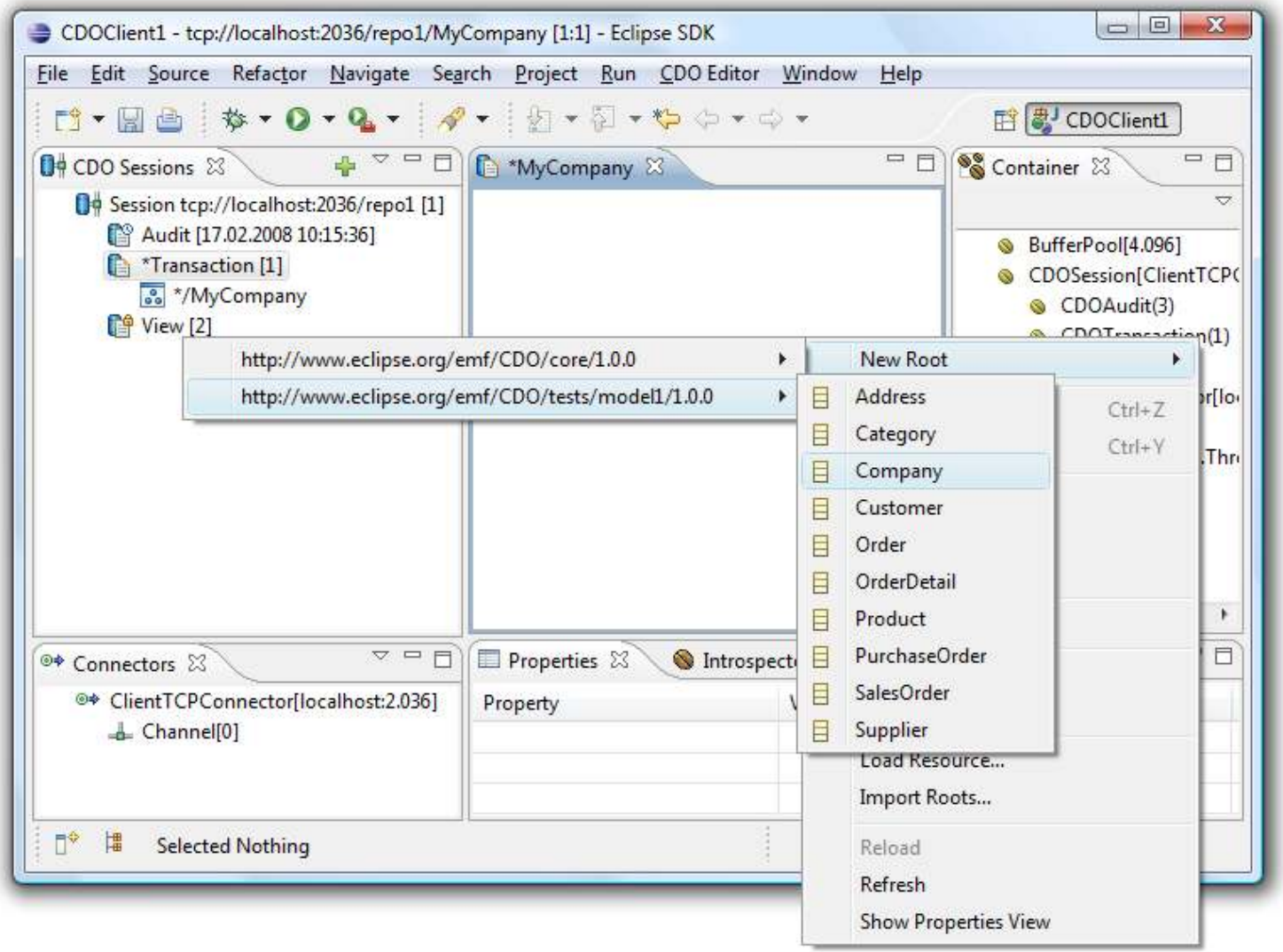
Using the CDO Client



Using the CDO Client



Using the CDO Client



Using the CDO Client

CDOClient1 - tcp://localhost:2036/repo1/MyCompany [1:1] - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run CDO Editor Window Help

CDOClient1

CDO Sessions
- Session tcp://localhost:2036/repo1 [1]
  - Audit [17.02.2008 10:15:36]
  - *Transaction [1]
    - */MyCompany
  - View [2]

*MyCompany
- Company ES-Computersysteme
  - Category MDA Consulting
  - Category OSGi Development
    - Product CDO
  - Customer Eclipse.org
  - Sales Order 4711
    - Order Detail 0.0

Container
- BufferPool[4.096]
- CDOSession[ClientTCP(
  - CDOAudit(3)
  - CDOTransaction(1)
  - CDOView(2)
- ClientTCPConnector[lo
  - Channel[0]
- java.util.concurrent.Thr
- TCPSelector

Connectors
- ClientTCPConnector[localhost:2.036]
  - Channel[0]

Properties | Introspector

Property	Value
City	Berlin
Name	ES-Computersysteme
Street	Eclipse Ave. 39

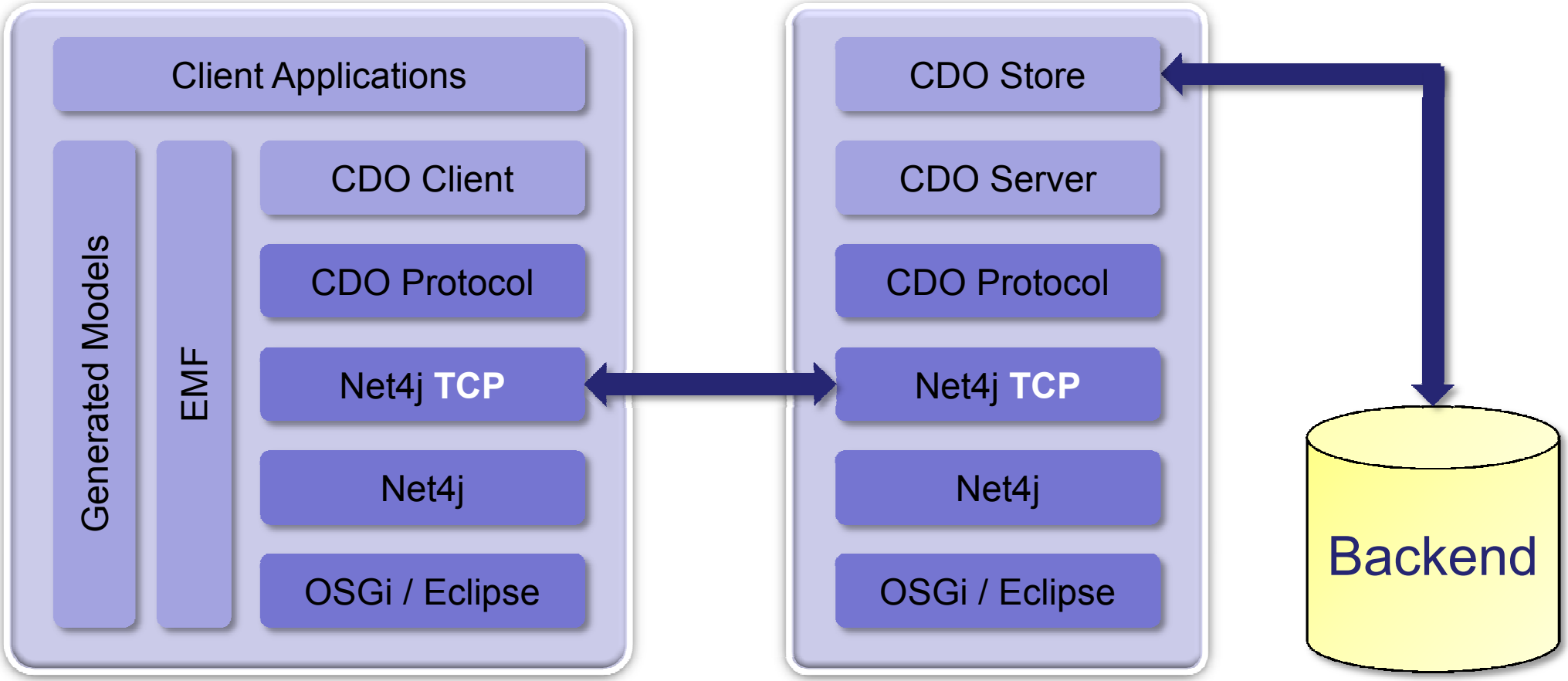
Selected Object: Company ES-Computersysteme

≤ 5 minutes

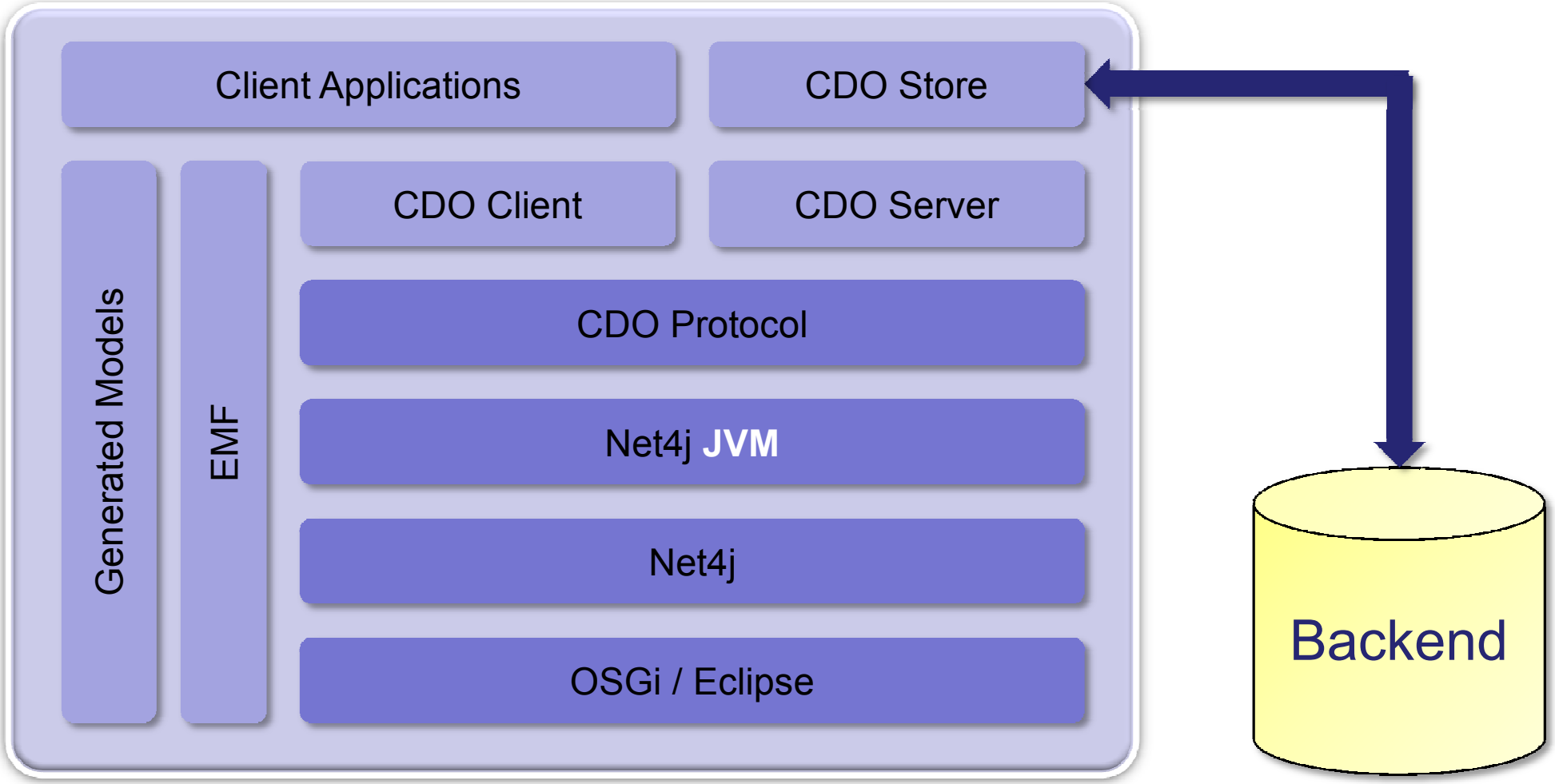
Detailed Architecture

- Deployment Options
  - Networked Remote Server
  - Embedded Server
- Static Decomposition
  - Server Components
  - Client Components
- Component Interaction
  - Committing a Transaction
  - Demand Loading Objects

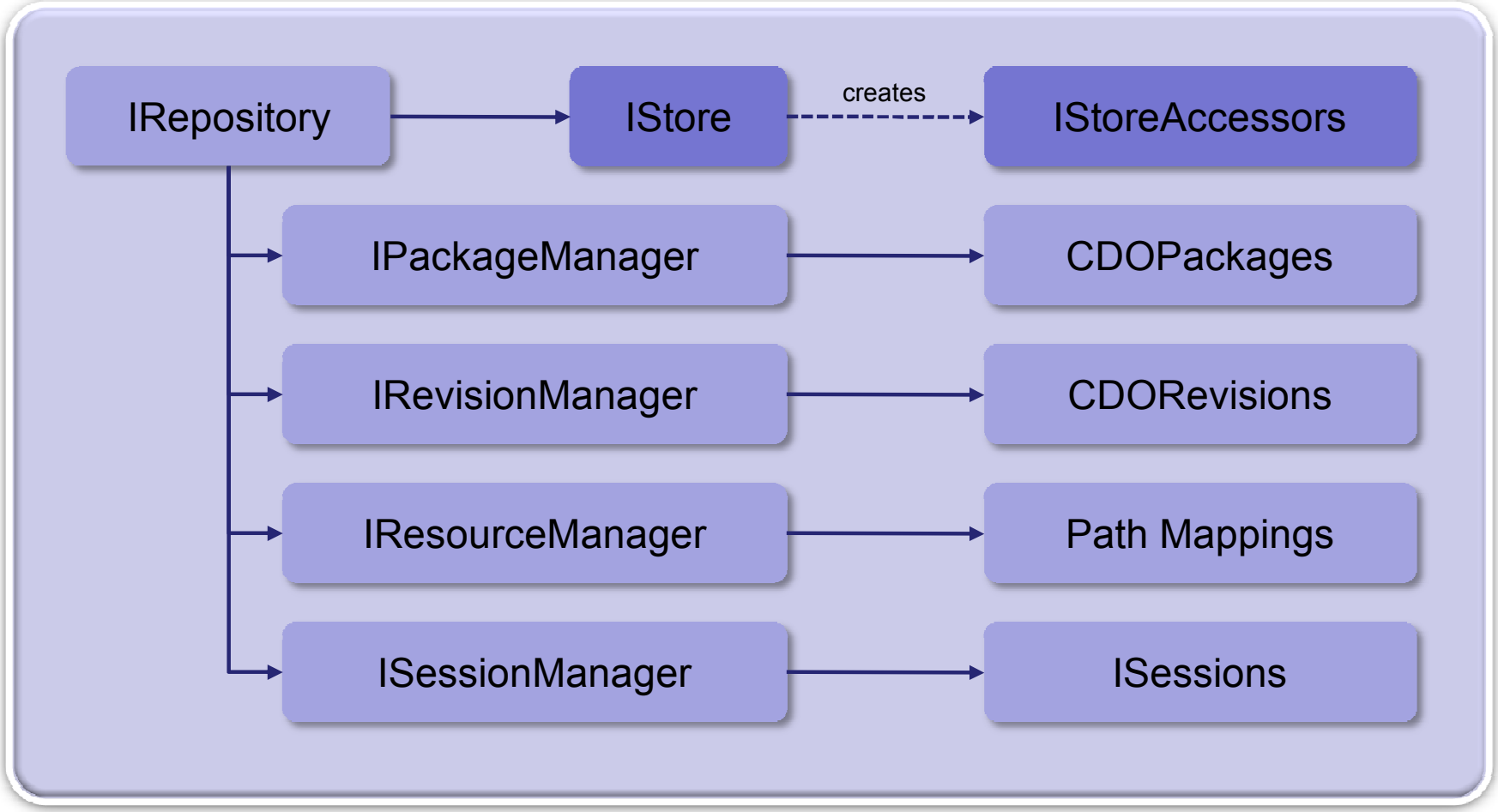
Deployment Options - Networked



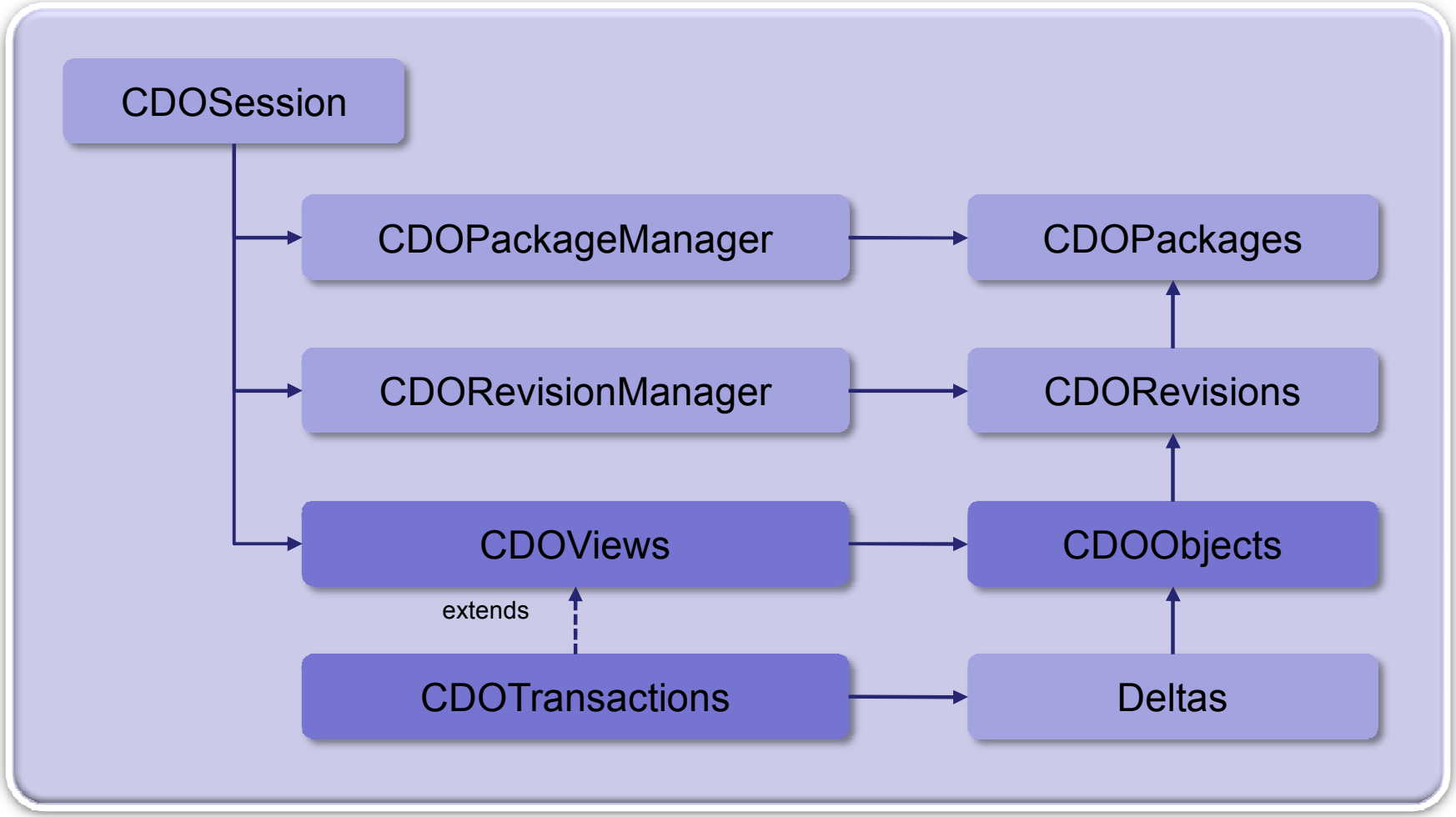
Deployment Options - Embedded



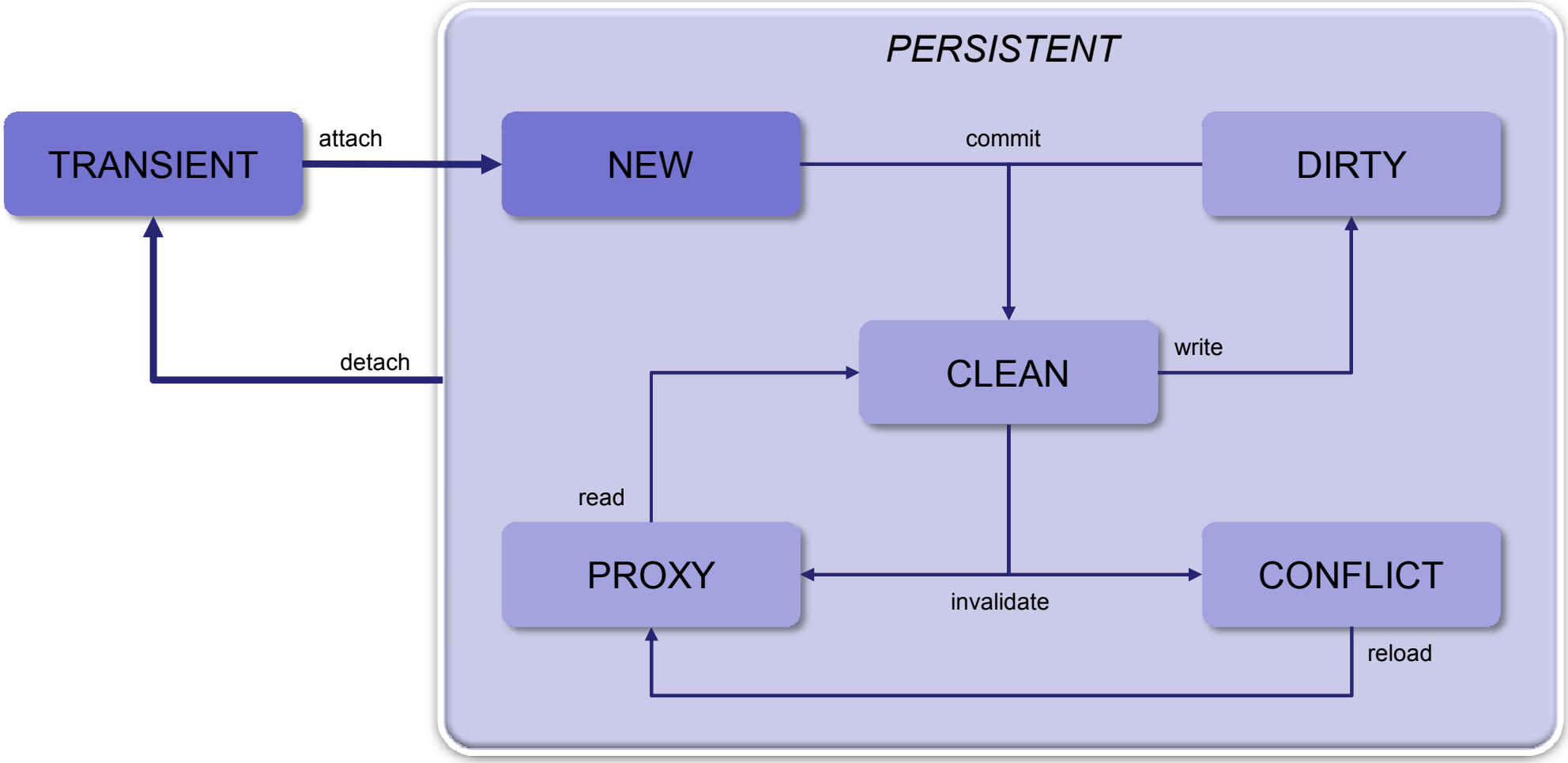
Static Decomposition - Server



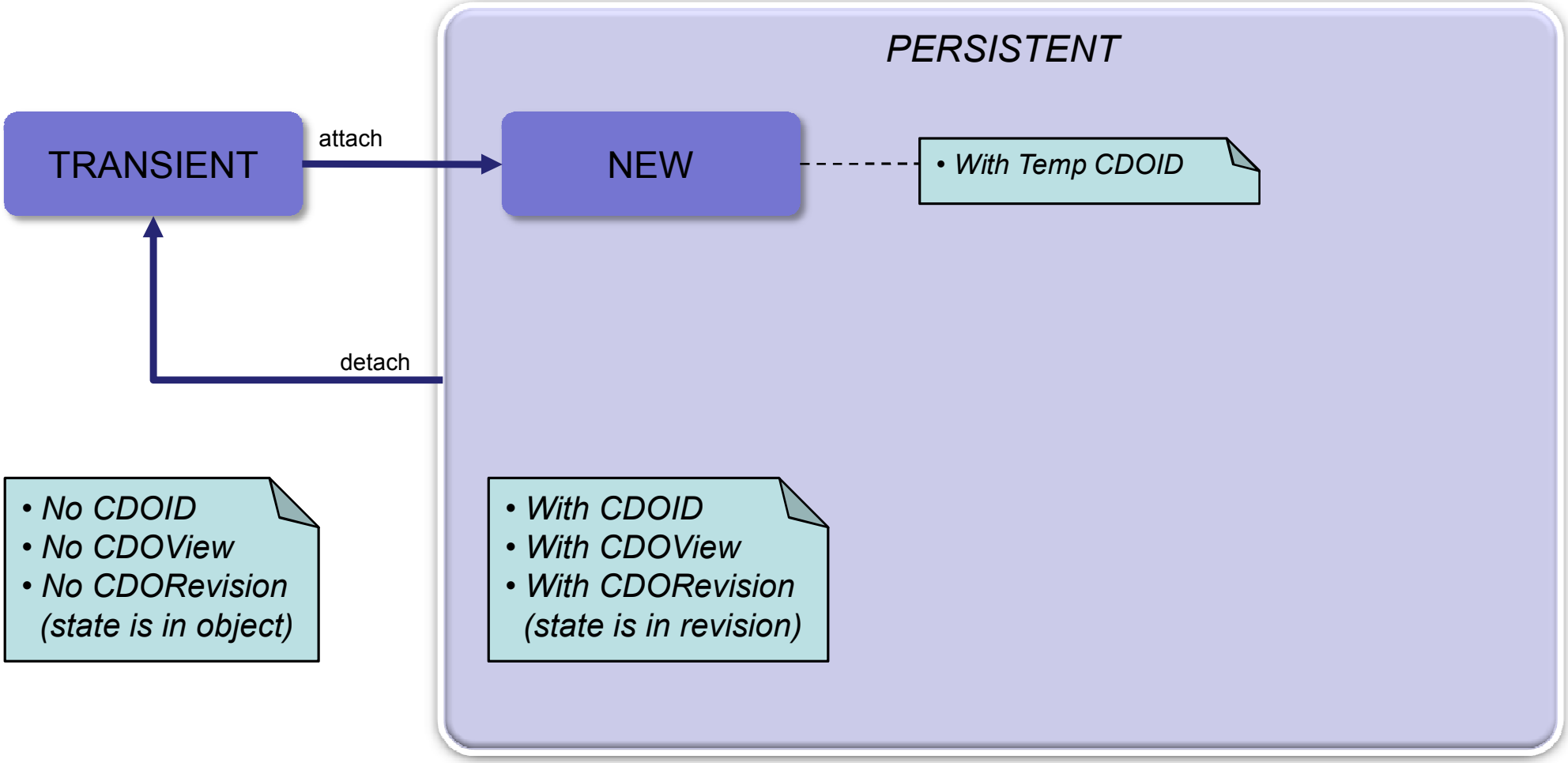
Static Decomposition: Client



Component Interaction – CDOStateMachine (1)



Component Interaction – CDOStateMachine (2)



Component Interaction - Committing

Client

- Client adds/modifies CDOObjects
- Client transaction creates temporary IDs for new objects and records change deltas
- Commit() sends new packages, new revisions and revision deltas to the server

Server

- Server passes data to the configured store
- Store remaps temporary IDs and persists the data
- Server sends back ID mappings
- Server notifies other sessions about invalidations

Client

- Client transaction applies ID mappings

Component Interaction – Demand Loading

Client

- Client accesses an EReference
- CDORevision delivers target CDOID
- CDOView looks up target CDOObject
  - Found → Finished
  - CDORevisionManager looks up CDORevision
    - Found → Creates new CDOObject, links it with revision, finished
    - CDORevisionManager sends LoadRevisionRequest

Server

    - IRevisionManager looks up CDORevision
      - If not found → Loads CDORevision from IStore and caches it
    - IRevisionManager sends back CDORevision to client

Client

    - CDORevisionManager caches CDORevision
- Creates new CDOObject, links it with revision, finished

≤ 10 minutes

Programming

- Using a Managed Container
- Using the Server API
- Using the Client API

Using a Managed Container (1)

```
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 Object[] getElements(String productGroup);
  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)

```
<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">
      </elementProcessor>
   </extension>
</plugin>
```


Using a Managed Container (3)

```
1  IManagedContainer container = IPluginContainer.INSTANCE;
2
3  IConnector connector = (IConnector)container.getElement(
4                          "org.eclipse.net4j.connectors",
5                          "tcp",
6                          "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");
```


Using the Server API

```
// 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
CDOSession session = 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();
```


≤ 5 minutes

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?

