eclipseCON 2006

The Eclipse Communication Framework (ECF)

Chris Aniszczyk <zx@us.ibm.com>
Scott Lewis <slewis@composent.com>

March 20th, 2006

What You Need

- To get the most from this tutorial, you need to bring a laptop capable of running Eclipse and developing plug-ins.
- You must have the following installed:
  - Eclipse 3.2 M5a (or M5)
  - EMF 2.2.M5 and GEF 3.2M5a
  - ECF SDK 0.7.5
  - ECF Plugins w/source (via project set file or anonymous CVS)

Tutorial Overview

- Introduction, Demos: IM/Chat, Collaboration, Shared Editor, VOIP/Call API
- Module 1: ECF Clients
  - ECF Clients
    - Trivial client
    - IM/Chat
    - Datashare
    - Discovery/FileShare
    - Extensibility with Shared Objects
- Module 2: ECF Providers
    - ECF protocol adapters revisited
    - Namespace Extension Point
    - ContainerFactory Extension Point

Tutorial Overview (cont)

- Module 3: You Decide
  - IM Clients
  - Multiplayer Game
  - Shared Editor
  - Other

Demos

- XMPP IM/Chat
- IRC
- Collab
  - Chat
  - URL Sharing
  - Remote view opening
  - Shared Workspaces/Editor

ECF Future

- New Sub-projects
  - Shared Editors
  - Call API/VOIP/Asterisk
  - Application Sharing/VNC
  - ECF+OSGI For Servers

Module 1

ECF API Introduction

ECF: Multi-Protocol Communications for Eclipse/RCP

- org.eclipse.ecf.core.IContainer: Interoperability through protocol abstraction
  - Container creation: ECF ContainerFactory

```
IContainer c =
  ContainerFactory.getDefault.createContainer("type");
```

  - IContainer semantics: connect/disconnect/lifecycle/listeners

```
c.connect(ID,IConnectContext);

...

c.disconnect();
```

  - Protocol adapters: aka Iadaptable abuse

```
IFileshareContainer fsc = (IFileShareContainer)
  c.getAdapter(IFileShareContainer.class);

if (fsc != null) ...
```


ContainerFactory Extension Point

- org.eclipse.ecf.containerFactory
- ECF providers implement communication protocol(s)
  - XMPP/Jabber, IRC, JMS, ECF 'generic', Yahoo
  - Working on: SIP, JXTA, Jingle, Sametime, RSS+SSE
  - Relying upon community involvement/contribution
- Allows clients/apps to write to IContainer API and not to specific implementation API

```
IContainer container =
  ContainerFactory.getDefault.createContainer("ecf.xmpp.sma
  ck");

IContainer.connect(...);
```


ECF Addressing

- Addressing needed for container.connect(**ID**,...);
- Represented by instance of org.eclipse.ecf.core.identity.ID
  - Have URI 'nature'
    - ecftcp://ecf.eclipse.org:3282/server
    - xmpp.smack:slewis@ecf.eclipse.org
    - mailto:slewis@composent.com
  - Not required to have URI syntax (but frequently do)
    - GUID: AF430D2189AFB8D
    - String: “channel1”
- IDs also useful as service identifiers (e.g. Discovery), user identifiers (IM/chat), file identifiers (fileshare), channel identifiers (datashare)

Extending Addressing for New Protocols

- org.eclipse.ecf.namespace extension point
  - Plugins provide extension Namespaces
    - e.g. xmpp jabber id syntax: jid:slewis@ecf.eclipse.org
  - Namespace extensions responsible for creating ID instances that follow ID contract
    - Immutable
    - Unique within Namespace
  - Examples

```
ID id1 = IDFactory.getDefault().createID(ns1,"
  slewis@ecf.eclipse.org");

ID id2 = IDFactory.getDefault().createID(ns1,"
  slewis@composent.com");

ID id3 = IDFactory.getDefault().createID(ns2,"
  slewis@ecf.eclipse.org");

e.g. id1.equals(id2) -> false, id1.equals(id3) -> false
```


ECF Protocol Adapters

- IContainer extends IAdaptable: IContainer.getAdapter(interface);
- Protocol semantics represented by adapter
  - Presence/IM/Chat: IPresenceContainer
  - Datashare: IChannelContainer
  - Fileshare: IFileShareContainer
  - Call API: ICallContainer
  - Shared Object API: ISharedObjectContainer
  - Others...
- Supports runtime querying of supported communications
- Providers choose to implement protocol adapters
- Example

```
IPresenceContainer pc = (IPresenceContainer)
  c.getAdapter(IPresenceContainer.class);
```


Building Simple Clients

1) Create IContainer
2) Retrieve/setup protocol adapter
3) Create target ID
4) Call IContainer.connect(targetID)
5) Send/receive messages (via adapter)
6) Disconnect
7) Dispose

Create IContainer

```
IContainer c = ContainerFactory.getDefault().
                              createContainer("ecf.generic.client");
```

Note: Exception thrown if no associated provider installed or initialization fails

Existing ECF providers

Protocol	Container Type Name
XMPP (jabber)	ecf.xmpp.smack
Yahoo	ecf.yahoo.jymsg
ECF 'generic' client	ecf.generic.client
ECF 'generic' server	ecf.generic.server
IRC	ecf.irc.irclib
Java Messaging Service	ecf.jms.tcp.client
Zeroconf/Bonjour	ecf.discovery.jmdns
datashare	ecf.generic.channel
fileshare	ecf.generic.fileshare
Sametime	ask Chris A
...	

ECF Protocol Adapters

```
//Presence

IPresenceContainer pc = (IPresenceContainer)
container.getAdapter(IPresenceContainer.class);

if (pc == null) throw new NullPointerException("presence
container not available");

else ...

//Discovery

IDiscoveryContainer dc (IDiscoveryContainer)
container.getAdapter(IDiscoveryContainer.class);

if (dc == null) throw new NullPointerException("discovery
container not available");

//FileShare, Call API, others...
```


ECF (connect) Target IDs

```
// For connection to container...
ID id1 =
IDFactory.getDefault().createID(cont.getConnectNamespace(),”ecftcp
://ecf.eclipse.org:3282/server”);

container.connect(id1,null);


// Using some other namespace
ID id2 = IDFactory.getDefault().createID(namespace,
                        “ecftcp://ecf.eclipse.org:3282/server”);


// GUID(SHA1+Base64)/String ID
ID id3 = IDFactory.getDefault().createGUID();

ID id4 = IDFactory.getDefault().createStringID(“channel1”);
```


ECF ID properties

- IDs instances are:
  - Immutable: Don't go changin'
  - Unique within Namespace: ID.getName()
  - Serializable
  - IAdaptable
  - URI nature: ID.toURI()
  - java.security.Prinicpal

Putting it Together: A Trivial Client

```
// Create container
IContainer container =
ContainerFactory.getDefault().createContainer("ecf.generic.cl
ient");


// Create target ID for connection
ID targetID =
IDFactory.getDefault().createID(container.getConnectNamespace
(),"ecftcp://localhost:3282/server");


// Connect to a target
container.connect(targetID,null);
```


Putting it Together: XMPP Clients

- org.eclipse.ecf.tutorial.ExampleClient1
- org.eclipse.ecf.tutorial.ExampleClient2
- org.eclipse.ecf.tutorial.ExampleClient3
- org.eclipse.ecf.tutorial.ExampleClient4

Protocol Adapter: Datashare

API: org.eclipse.ecf.datashare

Protocol adapter: org.eclipse.ecf.datashare.IChannelContainer

Example

```
IChannelContainer channelContainer = (IChannelContainer)
container.getAdapter(IChannelContainer.class);

IChannel channel =
channelContainer.createChannel(id,listener,props);
```

Channel used to asynchronously send arbitrary data (byte arrays)

```
channel.sendMessage("hello".getBytes());
```


Datashare Characteristics

- Sender Ordering (FIFO). Receivers guaranteed to get messages in same order they were sent
  - No stronger ordering guarantees
- Datashare Events
  - IChannelInitializeEvent – once upon channel creation/initialization
  - IChannelGroupJoinEvent – once for every connect/join
  - IChannelMessageEvent – once for every sendMessage
  - IChannelGroupDepartedEvent – once for every disconnect

Protocol Adapter: Discovery

Discovery API: org.eclipse.ecf.discovery

Protocol adapter: org.eclipse.ecf.discovery.IDiscoveryContainer

Example

```
IDiscoveryContainer discoveryContainer = (IDiscoveryContainer)
container.getAdapter(IDiscoveryContainer.class);

IserviceTypeListener listener = new IServiceTypeListener()
{ ... };

discoveryContainer.addServiceTypeListener(listener);
```

Listeners called asynchronously when service types are discovered

To register service types/services

```
discoveryContainer.registerServiceType("_ecftcp._tcp.local.");
discoveryContainer.registerService(serviceInfo);
```


Protocol Adapter: Shared Object Containers/Shared Objects

Shared Object API: org.eclipse.ecf.core

Protocol adapter: org.eclipse.ecf.core.ISharedObjectContainer

Example

```
ISharedObjectContainer soContainer = (ISharedObjectContainer)
container.getAdapter(ISharedObjectContainer.class);

ISharedObject so =
SharedObjectFactory.getDefault().createSharedObject("mysharedo
bject");

ID soID = IDFactory.getDefault().createGUID();

soContainer.getSharedObjectManager().addSharedObject(soID,so,n
ew HashMap());
```


Module 2

ECF Client Creation

Yahoo IM Client Creation

- This tutorial will cover creating an ECF IM client for the Yahoo messaging protocol
  - We'll use jymsg (http://jymsg9.sourceforge.net/)
- The steps will apply to any other type of messaging protocol so we encourage users to create IM clients for their favorite protocol
- Reference implementation available via CVS
  - ecf1.osuosl.org/ecf/plugins/org.eclipse.ecf.provider.yahoo

Yahoo IM Client Creation

- Step 1
  - **Create an identity**
    - Create a class, `YahooID` that extends `org.eclipse.ecf.core.identity.BaseID`

Yahoo IM Client Creation

- Step 2
  - **Define and register a namespace**
    - Create a class, `YahooNamespace` that extends `org.eclipse.ecf.core.identity.Namespace`
    - Register with the *org.eclipse.ecf.namespace* extension point

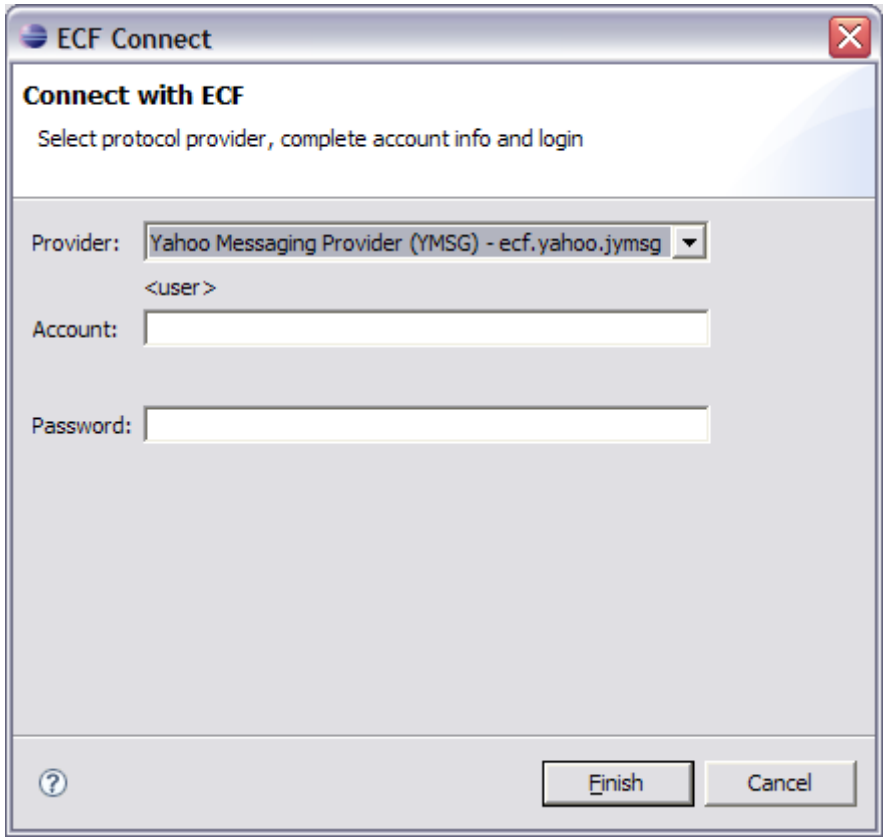
Yahoo IM Client Creation

- Step 3a
  - **Create a container instantiator**
    - Create a class, `YahooContainerInstantiator` that implements `org.eclipse.ecf.core.provider.IContainerInstantiator`
    - All we need to do here is create an `ID` via the `IDFactory` and return a new `YahooContainer` instance (stubbed)

Yahoo IM Client Creation

- Step 3b
  - **Define a container factory**
    - Extend the *org.eclipse.ecf.containerFactory* extension point
  - Create three properties
    - org.eclipse.ecf.example.collab.ui.JoinGroupWizardPage.usepassword
      - true
    - org.eclipse.ecf.example.collab.ui.JoinGroupWizardPage.examplegroupid
      - <user>
    - org.eclipse.ecf.example.collab.ui.JoinGroupWizardPage.groupIDLabel
      - “Account”

Yahoo IM Client Creation



Yahoo IM Client Creation

- Step 4a
  - **Implement the IContainer interface**
    - Create a class, `YahooContainer` that implements `org.eclipse.ecf.core.provider.IContainer`
  - This container will represent the connection to the Yahoo servers

Yahoo IM Client Creation

- Step 4b
  - Constructor
    - Grab the `localID`
    - Create a new yahoo `Session` instance
    - Create a new `YahooPresenceContainer`

Yahoo IM Client Creation

- Step 4c
  - `connect(…)`
    - Grab the user information from the targetID
    - Login to the Yahoo messaging server
    - Populate the initial roster (buddy list)
  - Recommend looking at the reference implementation to complete the steps

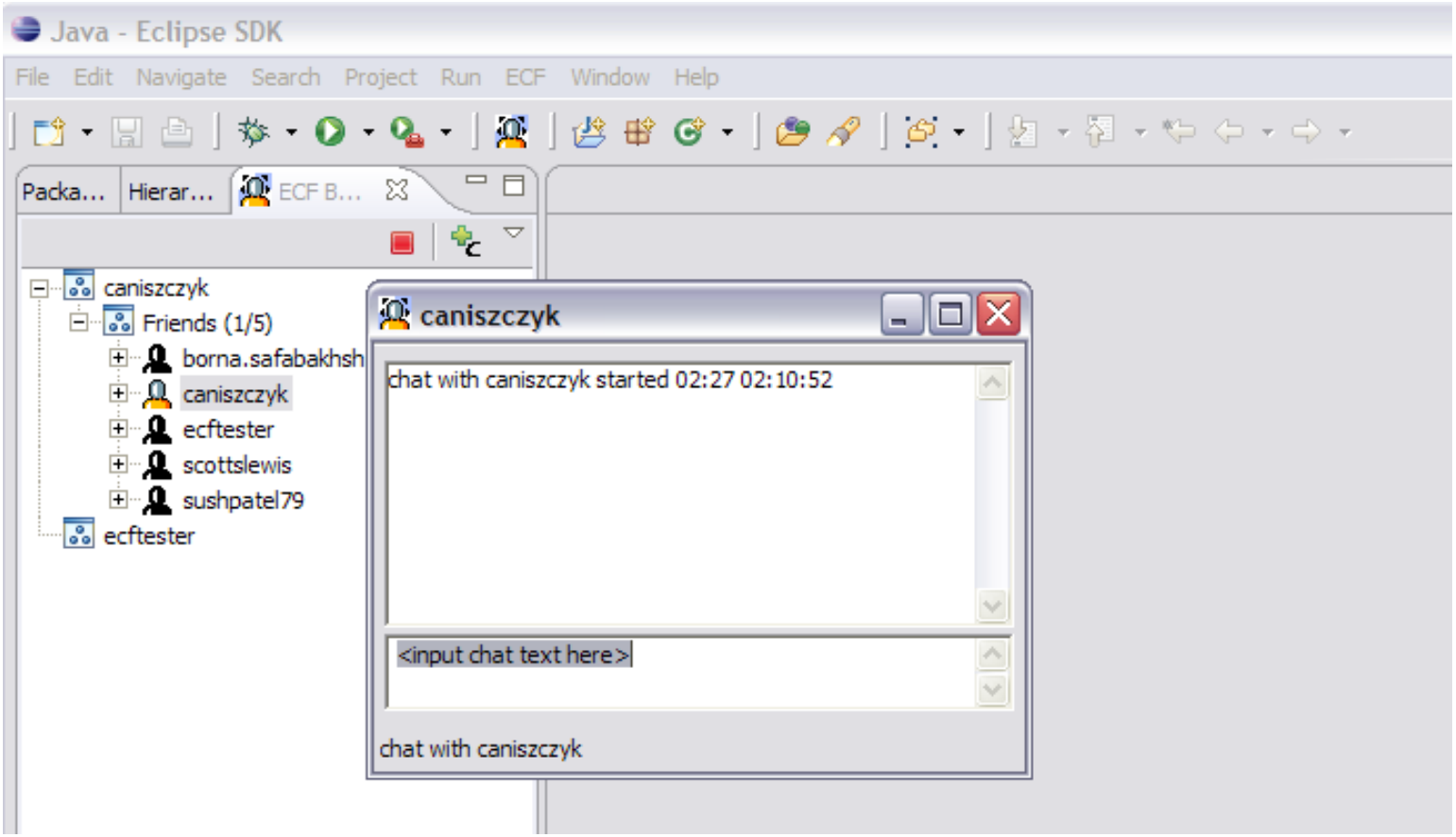
Yahoo IM Client Creation

- Step 5a
  - Create a `YahooPresenceContainer` that extends `AbstractPresenceContainer`

Yahoo IM Client Creation

- Step 5b
  - Create a new `IMessageSender` implementation
    - See getMessageSender() in `IPresenceContainer`
  - We will simply send a text message using the jymsg API
    - See sendMessage(…) in `Session`

Yahoo IM Client Creation

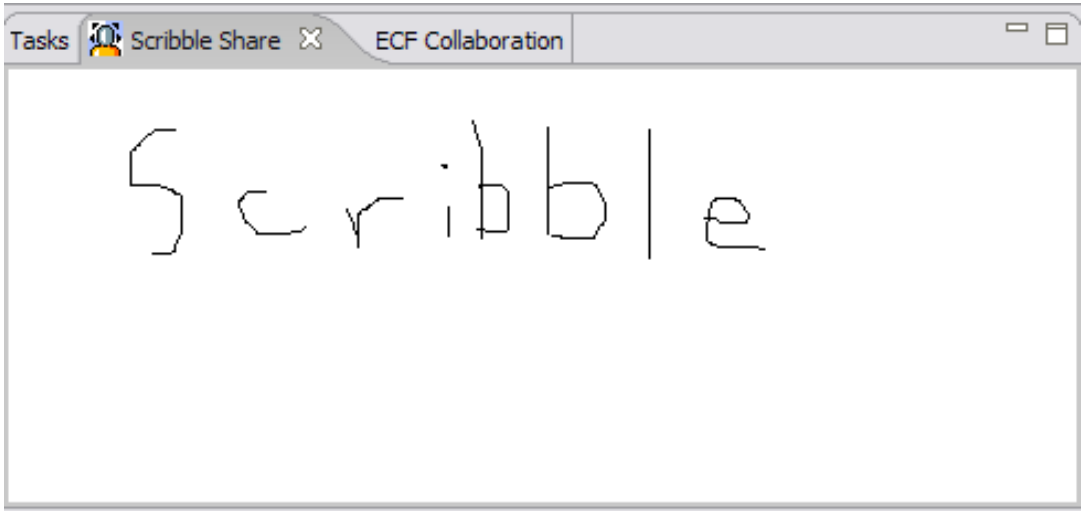


ECF ScribbleShare Application

- This tutorial will cover creating a ECF ScribbleShare application that uses the datashare APIs
- Reference implementation available via CVS
  - dev.eclipse.org/cvsroot/technology/org.eclipse.ecf/plugins/org.eclipse.ecf.tutorial

ECF ScribbleShare Application

- Step 1
  - Create a simple view so we can scribble some stuff
  - `ScribbleView` extends *org.eclipse.ui.part.ViewPart*
    - See `BasicScribbleView` for reference implementation



ECF ScribbleShare Application

- Step 2
  - Create a basic scribble client `ScribbleClient`
    - Create a container instance of *ecf.generic.channel*
    - Open the scribble view
    - Create the channel to communicate
    - Connect the container to the target server

ECF ScribbleShare Application

- Step 3
  - Update `ScribbleView`
    - Handle the drawing of lines
    - Send the drawing of lines to the channel

Module 3

You Decide

You Decide

- Choose your own adventure!
  - ECF-based game?
  - ECF-based application?
    - IM Clients?
  - ECF roundtable discussion?

