IBM

Getting started with the Eclipse Communication Framework

Use ECF to create communications-based applications quickly

Chris Aniszczyk, Software Engineer, IBM, Software Group
Borna Safabakhsh (borna@us.ibm.com), Software Engineer, IBM, Software Group

Summary: The Eclipse Communication Framework (ECF) is a new Eclipse project devoted to providing an open source framework supporting the creation of communications-based applications on the Eclipse platform. Find out about the ECF, its basic capabilities, and its future direction.

Date: 14 Mar 2006
Level: Introductory
Activity: 1078 views
Comments:

Average rating

Background

The goal of the ECF project, most simply, is to provide a cross-protocol API. ECF creates value for four constituencies:

Communications providers

Who -- Plug-in developers implementing new and existing communication providers (Yahoo, AIM, etc.) for Eclipse-based applications and plug-ins.
Value -- Interoperability. Developers and users can allow their applications to work together in and across protocols.

Component developers

Who -- Plug-in developers implementing generic component-level features (file sharing or whiteboard sharing, for example).
Value -- Reusability. Application designers now can reuse components as long as the provider implementation supports it.

Tool integrators

Who -- Plug-in developers who integrate existing applications with ECF technology to add new value. A good example would be adding collaboration support to a graphical editor.

Value -- Feature enrichment. Developers can breathe new life into their applications via ECF.

User interface (UI) developers

Who -- Plug-in developers who build or integrate user interfaces. ECF provides the bare bones, so you can provide your own UIs for chat rooms, file sharing, etc.

Value -- Usability. UIs can be improved and customized independent of underlying implementation, at design or even integration.

ECF aims to meet this goal by providing a set of high-level abstractions, rather than yet another messaging API. This design choice empowers you to reuse high-level communications components (instant messaging, file sharing, video conferencing, etc.) in varying application contexts and UIs. Imagine developing an application that requires instant messaging, blogging, BitTorrent, file sharing, and voice conferencing. With ECF, development can be expedited over all the communication code for each of those services, allowing you to focus on business logic and UIs.

Installing ECF

ECF is installed as a set of Eclipse plug-ins, via the standard plug-in installation processes:

- Eclipse Project
- Eclipse ECF update

The ECF project also hosts a server containing extra providers and application examples (IRC, JMS, etc.):

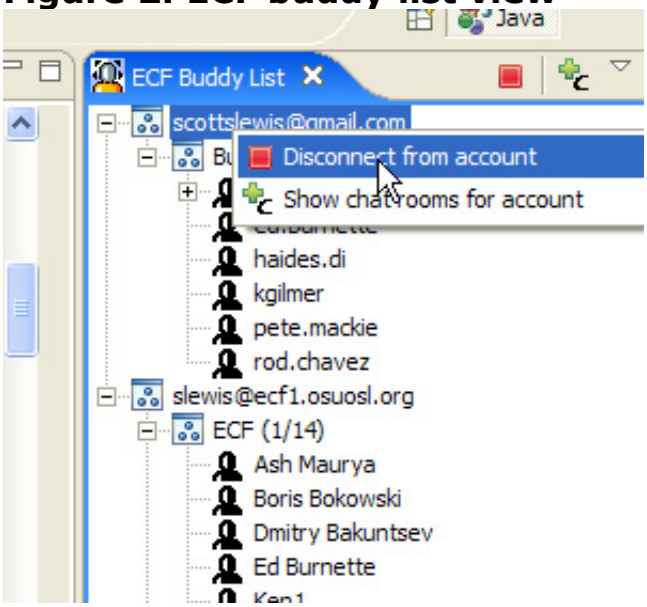
- Eclipse Communication Framework
- Eclipse Communication Framework update

ECF eye candy

We believe that eye candy is essential to the success of any article (must keep readers interested). Figure 1 shows the sample ECF IRC provider demonstrating some of the chat-room capabilities of the framework. Figure 2 shows the sample buddy list view with two messaging connections.

Figure 1. ECF IRC provider

Figure 2. ECF buddy list view



Now that you have seen what ECF looks like, we can discuss the finer details.

ECF adapters

The first step in understanding how ECF works is to take a quick look at its APIs, particularly adapters. In ECF, the fundamental building blocks are the communication containers of instance *IContainer*. IContainer instances provide a common interface for communication within ECF. It can represent an instance of a communication context, be it of a client-server or publish-subscribe communication model.

IContainer is an important interface within ECF, and the extensibility it provides via the familiar Eclipse IAdaptable interface is crucial. Since ECF containers support runtime extensibility by being IAdaptable, you can query runtime instances about their capabilities and act on them. Examples of capabilities in the context of ECF are presence/chat, chat rooms, etc. The table below lists common capabilities found in ECF.

Common ECF capabilities

IPresenceContainer	Exposes the capability of setting up listeners for **presence messages** (away,available,etc...), **text messages**, **subscription requests**. A good implementation of this interface would be any chat messaging protocol (Yahoo, AIM, Gadu Gadu, etc.).
ISharedObjectContainer	Exposes the capability to build arbitrary communication protocols by providing the power to pass arbitrary information around.
IDiscoveryContainer	Exposes the capability to support look-up and discovery facilities (computers, devices and services on IP networks). Apple's Bonjour is a good example of a possible implementer of this adapter.
IChatRoomManager	Exposes the capability to support chat rooms within ECF. Example implementations of this interface are protocols that support chat rooms (Yahoo, XMPP, IRC, etc.).

As ECF matures, these adapters will be refined and new ones will be added.

ECF example

We will play the role of a communications provider, particularly, with the Yahoo! messaging protocol (YMSG). We will use an open source implementation of the protocol, jYMSG. This article includes a basic ECF implementation of the YMSG protocol through an ECF presence container (`IPresenceContainer`). We will outline the steps taken to implement this and have made the final code available for download. (Note that this code will most likely end up being contributed to the ECF project.) The steps involved in implementing a presence container for YMSG are as follows:

1. Defining and registering a `Namespace`
2. Implementing an `ID`
3. Implementing the `Container`
4. Defining and registering a `ContainerInstantiator`
5. Implementing the `IPresenceContainer`, Yahoo-style

Namespaces and identifiers

Namespaces are responsible for creating new `ID` instances via ECF's `IDFactory`. They are given the power to restrict what type of ID instances are created via the factory. In our `YahooNamespace` implementation, we are keeping it simple by requiring only a username to construct an ID.

Listing 1. YahooNamespace.java

```
public class YahooNamespace extends Namespace {
```

```
...
        public ID createInstance(Class[] argTypes, Object[] args) \
        throws IDInstantiationException {
                try {
                        return new YahooID(this, (String) args[0]);
                } catch (Exception e) {
                        throw new IDInstantiationException\
                        ("Yahoo ID creation exception", e);
                }
        }
}
```

We also need to make sure that we properly register the namespace so ECF knows about it. This is accomplished via the org.eclipse.ecf.namespace extension point.

Listing 2. plugin.xml

```
   <extension
         point="org.eclipse.ecf.namespace">
      <namespace
            class="org.eclipse.ecf.provider.yahoo.identity.YahooNamespace"
            description="Yahoo Namespace"
            name="ecf.yahoo"/>
   </extension>
```

After creating the namespace, we create the identifier `YahooID`, used to uniquely identify an entity in the Yahoo! namespace. (see Download).

Containers

In our case, a connection-oriented client-server scenario, ECF containers represent the notion of a communications session. We will implement the `IContainer` interface and make the `YahooContainer` adaptable to the `IPresenceContainer` interface. See Listing 3 for a glance at the important methods.

Listing 3. YahooContainer.java

```
public class YahooContainer implements IContainer {
...
private Session session; // jYMSG session
...
        public void connect(ID targetID, IConnectContext connectContext)
                throws ContainerConnectException {
                String password = getPassword(connectContext);
```

```
                this.targetID = (YahooID) targetID;
                try {
                        // login to yahoo messaging
                        session.login(this.targetID.getUsername(), password);
                }
...
                // listen to yahoo messaging events
                session.addSessionListener(new YahooSessionListener(presenceContainer));
                // notify ECF we joined
                presenceContainer.fireContainerJoined(getConnectedID());
        }
...
        public Object getAdapter(Class serviceType) {
                if (serviceType.equals(IPresenceContainer.class)) {
                        return presenceContainer;
                }
                return Platform.getAdapterManager().getAdapter(this, serviceType);
        }
...
```

In the code above, we connect to the Yahoo! messaging server and notify ECF of our successful connection. Also notice that we adapt to the `IPresenceContainer` interface in the `getAdapter` method.

Container instantiators

Container instantiators are simple classes responsible for -- you guessed it: creating ECF containers. In order to be a container instantiator, you need to implement the `IContainerInstantiator` interface. See Listing 4 for our simple `YahooConstainerInstatiator`. There's not much to it.

Listing 4. YahooContainerInstantiator.java

```
public class YahooContainerInstantiator implements IContainerInstantiator {

        public IContainer createInstance(
        ContainerDescription description,
        Class[] argTypes,
        Object[] args) throws ContainerInstantiationException {
                ID guid;
                try {
                        guid = IDFactory.getDefault().createGUID();
                } catch (IDInstantiationException e) {
                        throw new ContainerInstantiationException\
                        ("Exception creating ID",e);
                }
                return new YahooContainer(guid);
        }

}
```

IPresenceContainer

The `YahooPresenceContainer` is the most active class in our arsenal. For simplicity, we'll focus on a few methods, see Listing 5.

Listing 5. YahooPresenceContainer.java

```
public class YahooPresenceContainer implements IPresenceContainer {
...
        public IMessageSender getMessageSender() {
                return new IMessageSender() {
                        public void sendMessage(
                                        ID fromID,
                                        ID toID,
                                        Type type,
                                        String subject,
                                        String messageBody) {
                                try {
                                        session.sendMessage\
                                        (toID.getName(), messageBody);
                                } catch (IllegalStateException e) {
                                        e.printStackTrace();
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                };
        }

        public void handleMessageReceived(SessionEvent event) {
                for(int i = 0; i < messageListeners.size(); i++) {
                        IMessageListener l = (IMessageListener) messageListeners.get(i);
                        ID from = makeIDFromName(event.getFrom());
                        ID to = makeIDFromName(event.getTo());
                        l.handleMessage(
                                        from,
                                        to,
                                        IMessageListener.Type.NORMAL,
                                        event.getFrom(),
                                        event.getMessage());
                }
        }
...
}
```

Notice that we implement the `getMessageSender` method to send messages to users. In

our case, we're using our Yahoo `Session` and sending the appropriate message. And we have a helper method called `handleMessageReceived` responsible for notifying `IMessageListener`s on our presence container if we have received a message from a Yahoo user.

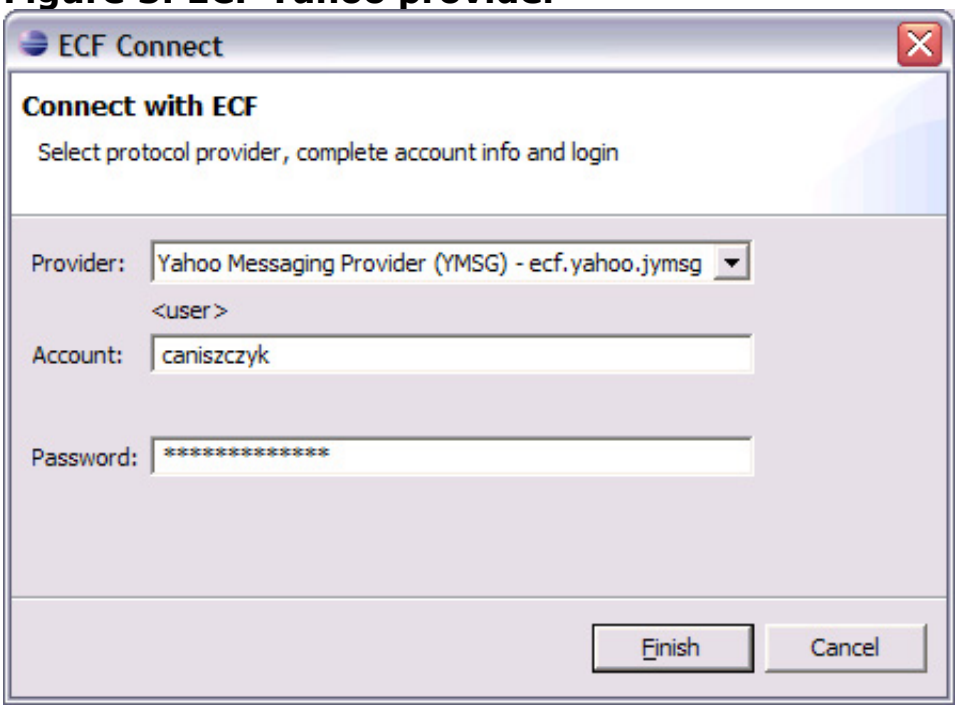
There are other required methods on `YahooPresenceContainer` purposely left to be null for simplicity (or the jYMSG APIs limited us). These methods are:

- `getAccountManager` -- Responsible for actions that deal with account maintenance (changing passwords, creating accounts, etc.)
- `getPresenceSender` -- Responsible for notifying people of interest about your presence updates (away/available, add buddy, etc.)
- `getChatRoomManager` -- Responsible for managing chat rooms (for example, Yahoo chat lobbies)

As an exercise, we recommend implementing chat room support via the `getChatRoomManager` method.

After everything is implemented, to test the functionality of our provider, we simply use facilities provided by ECF (see Figure 3) to run and select our provider.

Figure 3. ECF Yahoo provider



Conclusion

The future for ECF is bright, with collaborative computing on the rise and features like VoIP and video conferencing around the corner. Also, there is talk about the

Eclipse Foundation using ECF to further support its committers through better collaboration tooling.

We gave you a quick glimpse into ECF and an example that demonstrated one of the many capabilities the framework provides. We hope you find many uses for ECF in the future and consider contributing to the project.

Acknowledgments

The authors thank Scott Lewis and the ECF team for their support in writing this article.

Download

Description	Name	Size	Download method
Yahoo ECF Provider	os-ecl-commfwk_org.eclipse.ecf.provider.yahoo.zip	142KB	HTTP

Information about download methods

Resources

Learn

- Learn more about Eclipse Communication Framework.
- Learn more about the Eclipse Foundation and its many projects.
- Visit developerWorks' Eclipse project resources to learn more about Eclipse.
- Stay current with developerWorks technical events and webcasts.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- Get involved with development by checking out the Eclipse.org ECF mailing lists.
- Find help at the Eclipse.org ECF Newsgroup. (Note that clicking on this link will launch your computer's Usenet newsgroup reader, if you have it configured correctly.)
- The Eclipse newsgroup has lots of resources for people interested in using and extending Eclipse.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the authors

Chris Aniszczyk is a software engineer at IBM Lotus focusing on OSGi related development. He is an open source enthusiast at heart, and he works on the Gentoo Linux distribution and is a committer on a few Eclipse projects (PDE, ECF, EMFT). He's always available to discuss open source and Eclipse over a frosty beverage.

Borna Safabakhsh is a software engineer with IBM Tivoli Security and a graduate of IBM's Extreme Blue internship program. A proponent of the problem-solving powers of information and technology, he pursues new approaches and applications for answers to today's and tomorrow's questions. Also a fan of motorsports, Safabakhsh is a fan of all things fast.