GJTAPI Web Services Bridge

Richard Deadman

Deadman Consulting, http://www.deadman.ca

October, 2003

Overview

The GJTAPI Web Services Bridge is an remote adapter that allows the GJTAPI framework to be connected to a service provider located on another machine using web services (aka SOAP or xml-rpc). The GJTAPI framework is an implementation of JTAPI 1.3 that supports the following packages: core, most of callcontrol, media, and privatedata. GJTAPI also support Jain Jcc 1.1. GJTAPI is designed as a generic framework supporting multiple telephony fabrics through the use of plugged in service providers. Service Providers are available for Dialogic boards, PBX systems, MS Tapi and voice modems to name a few. More information on GJTAPI can be found at http://gjtapi.sourceforge.net.

Occasionally it is useful for the GJTAPI framework and the “telephony fabric” to exist on different machines. By acting as a delegate service provider, the Web Services service provider connects a GJTAPI instance to any server-based telephony service provider. The document outlines the architecture and how to set up the system.

Licensing

Unlike other GJTAPI service providers, the web service GJTAPI provider is dual-licensed. It is licenced under both the GPL and a commercial licence. While this may be confusing, in reality it offers more flexibility than the GPL alone. The rules are simple:

If you intend to use the GJTAPI Web Service bridge in an application that is released under the GPL, you are free to do so. Unlike software only released under the GPL, if you are developing a commercial application you also have the ability to use this GJTAPI service provider. If your application falls into the second category, you must contact Deadman Consulting to acquire a license.

Requirements

1. Java virtual machine of version 1.3 or greater
2. JTAPI 1.1
3. Optionally Jain Jcc 1.1
4. GJTAPI 1.6

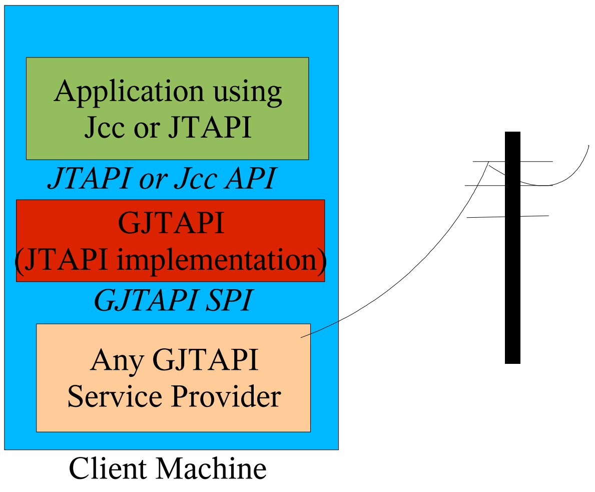
5. The server must support servlets 1.1
6. The server must support Jax-rpc version 1.1 FCS
7. Gjtapi-rpc.war server-side application file.
8. GjtapiWebService.jar client jar file.

Architecture

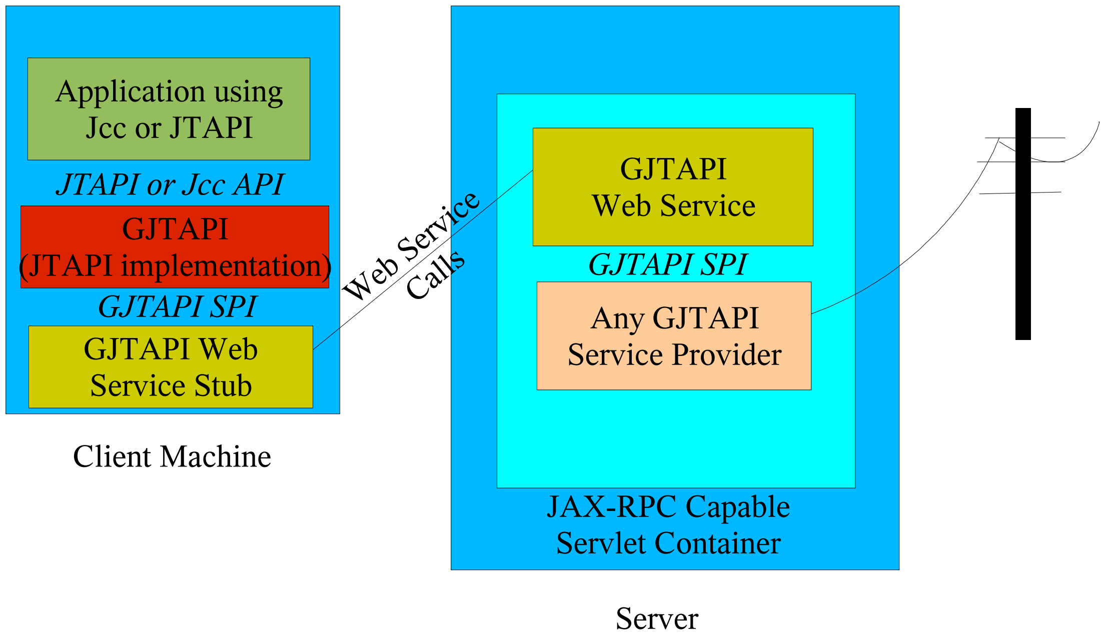
Like other remote bridges for GJTAPI, the web services provider contains two parts, a server-side part that plugs another GJTAPI service provider into it and a client part that connects the GJTAPI "framework" to the server-side service provider.

A diagram is in order:

Typical GJTAPI Architecture:



GJTAPI Architecture with Web Service Bridge:



Basically, the Web Service Bridge is an adapter that delegates GJTAPI service-provider-interface calls on to a remote GJTAPI service provider. By adherring to the GJTAPI SPI on both ends, the Web Service Bridge can be injected anywhere in the GJTAPI application stack to allow access to a service provider on a remote machine.

GJTAPI provides similar remote bridges for CORBA and RMI, with one difference: CORBA and RMI are peer-to-peer systems which make it easier for the "client" to be notified by the "server" of asynchonous telephony events. SOAP/web services does not allow for callbacks. To get around this, the Web

Service Bridge employs a polling architecture where events are cached on the server for each client and then picked up by a client thread that polls the server periodically for events.

Both server and client parts can be configured using a properties file that specifies how they should behave. On the server, this properties file specifies one main property (`ca.deadman.gjtapi.wsAdapter`) that names the class used to connect up to the service provider to delegate GJTAPI SPI calls to. As well, this properties file can contain any properties specific to the service provider selected.

On the client, there is also a properties file that specifies the web service location. The entries in this file can be overridden in the JTAPI call to JtapiPeer.getProvider(String). The most interesting property is “server” which specifies the host name and port that the web service is installed on.

Howto

On the server:

1. Ensure that you have a web service servlet container that supports JAX-RPC 1.1 FCS.
2. Place the JTAPI 1.1 or Jain Jcc 1.1 jar file and GJTAPI 1.6 jar file into the “shared” jars directory of your web server.
3. Unpack Gjtapi-rpc.war using a zip tool and edit the file “WEB_INF/classes/ca/deadman/gjtapi/raw/remote/webservices/server.props”. This file should contain an entry for `ca.deadman.gjtapi.wsAdapter` that specifies the class name of the GJTAPI service provider to delegate calls to. By default this is set to the “Emulator” service provider.
4. Add any other properties to server.props that are required.
5. Repack the war file using a zip tool.
6. Drop the war file into the applications directory of your web server.
7. Use the web servers JAX-RPC management web page to check the proper deployment.

Now for your client application, to use GJTAPI and the Web Service Bridge:

1. Place JTAPI 1.3 and optionally the Jcc 1.1 jar file onto your application class path
2. Place the GJTAPI 1.6 and GjtapiWebService jar files onto your application class path
3. Use the following call in your application to connect to the server-side service provider:

```
JtapiPeer peer = JtapiPeerFactory.getJtapiPeer
("net.sourceforge.gjtapi.GenericJtapiPeer");
```

```
Provider prov = prov = peer.getProvider
("ca.deadman.gjtapi.raw.remote.webservices.WebProvider;server=localhos
t:8080");
```

This tells GJTAPI to connect to the remote web service “service provider” located on machine “localhost” at port “8080”.

Further Information

This is meant as a simple introduction to the GJTAPI Web Service bridge. For further information, please use the http://gjtapi.sourceforge.net web site and its mailing list, or contact Deadman Consulting at http://www.deadman.ca .
