Application Overview

The goal of this application was to create the absolute, most bare-bones, GeoPlex Client-Server Peer/Peerlet demonstration application. The purpose of this exercise was to attempt to package a minimal configuration for a client-server GeoPlex service. This package can be used for educational purposes and extended to be the platform on which developers can build more sophisticated applications.

The **HelloWorld** application includes both a service Peer and Peerlet, as well as a client Peerlet. The service-side provides a service that awaits for connections and returns the text string “Hello World.” The client-side accesses this service via the GeoPlex cloud, connects to it and (if successful) prints out the string that it receives in a Java/Swing window. In the current implementation there is no need for special GeoPlex proxies.

Implementation Details

Since this is a client-server package, there are two parts. For the server side, you can choose among monolithic peer and a peerlet, but their functionality is basically the same with only a couple of differences. The monolithic peer communicates all the output to the console, while the peerlet in absence of `System.out` and `System.err` uses a simple Java-based Swing output window. The monolithic peer authenticates the user to the cloud while the peerlet assumes that the user is already authenticated with the help of the `PeerConsole`. Otherwise both of them are the same. The software tries to open the server socket and announce the service for the opened port on the peer machine. Any failure causes the software to stop completely. After the announcement succeeds, the service is ready to take connections. Any new connection is output and the client is sent “Hello World” message. To stop the peerlet-type service, the user can press “Stop Service” button. When the service is stopped for any reason, it is denounced and the server socket is closed. Even if the denouncement was not occurring, it would be automatically done by the cloud when the administrator of the service logs out. Before this service can be run, it has to be registered on the cloud. Whoever registers the service would become the administrator for the service.

The client side consists of only a single peerlet. It makes no difference if the service for it is a monolithic peer or a peerlet. This peerlet too assumes that the user is already authenticated to the cloud prior to running it.
(a) diagram of how the HelloWorld service works.

(b) Peer-based Server for HelloWorld service.

(c) Peerlet-based Server and Client for HelloWorld service.

Figure 1:
connect to the service, the user has to input the ip and port of the service. This information is available through the GetASRPeerlet. If you are using gunet.net, HelloWorld service already exists and you can use it (handle helloworld2). After entering the connectivity information, the user presses “Connect” button. The peerlet tries to establish connection to the service. If successful, it reads a line from the socket and displays the received message in a dialog window.

This setup assumes that the connection to the service is made through the cloud. However, because of known problems with the GeoPlex, that might not be the case. The connection is actually established shortest route.

**Future Ideas**

There are several ways in which this tool could be extended and enhanced:

1. **Smart Proxies**: Set up GeoPlex Gate proxies to channel (tunnel) the traffic for the service through the cloud.

2. **Wrappers for Legacy Applications**: Enhance the Server-side Peer/Peerlet to wrap up an existing legacy application and make it accessible through GeoPlex.

3. **Add Caller ID to the service**: Currently because of no assurance of particular path of the connection, this might cause problems. In the ideal situation, however, the service would be using getCallerID method to limit its accessibility to only particular users if needed.