Connector Paper

Jose Melchor
Winter 2011

Below is the first draft for the ICSEng 2011 conference paper. Currently, only the implementation part is drafted. Diagrams are included but they are only the first version as well. Diagrams need to be referenced by the text.

4. Implementation

Connector.jar package currently contains the following classes:

- FileInputStream: Read remote files (FTP, SFTP, and HTTP).
- FileOutputStream: Write to remote files (FTP and SFTP).
- Connector: Redirects Output, Input, and Error to remote GUI.
- Frame and Graphics: Redirects graphics to remote GUI.

4.1. Daemon Thread

When a Connector object is instantiated, a daemon process is automatically created. A single daemon serves objects from the entire Connector.jar package. For instance, a daemon will serve FileInputStream methods, as well as Frame or Graphics methods. Connector Daemons will connect to a GUI as part of the instantiation process. The Daemon knows where to connect thanks to the file map passed through the connector's constructor. Along with the File-to-URL definitions, the file map contains two lines that define the IP address and port of the GUI listening for a daemon connection.

This design assumes that a remote connector GUI will be listening for incoming connections from a daemon. If no GUI exists on the IP and port defined, daemon functions that rely on a GUI will simply be disabled. However, functionality for reading and writing files will work. Consequently in terms of file I/O operations, connector daemons are not dependent on the availability of a GUI.

A connector daemon object possesses multithreaded capabilities. When a daemon receives a request from a Connector.jar object, a new thread is spawned to deal separately with each request. (Refer to figure)

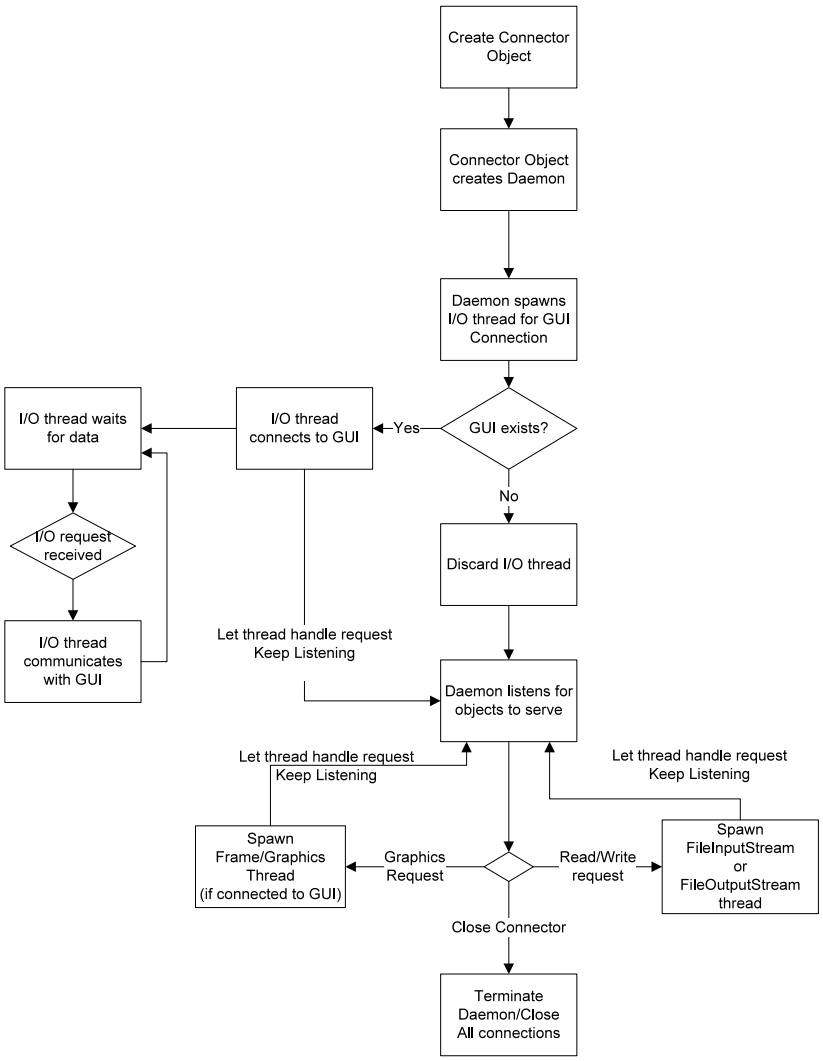

Figure 1 Daemon Life Cycle

4.2. GUI

The GUI uses two channels of communication to deal with data coming from an application. One socket is used to deal with I/O messages and another socket to deal with graphics.

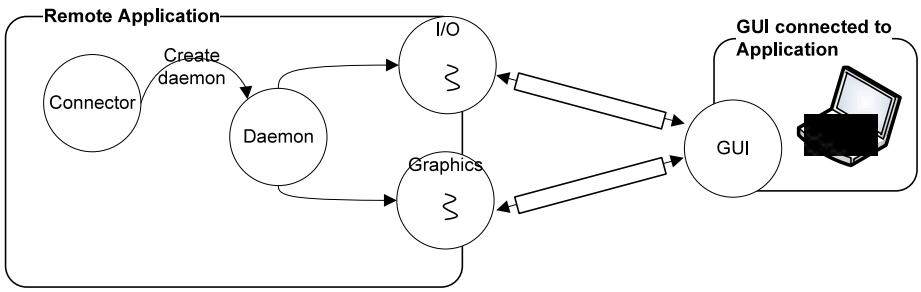

Figure 2 GUI

When a GUI is started, the user may choose a port to be used for incoming connections. Once a port is chosen, the user must press the "connect" button so that the GUI enters a listening state. The GUI then waits for a remote application to start. Interaction begins once a remote application successfully connects to the GUI. When an application finishes, the GUI goes into a listening state once again.

Applications interacting with the GUI may able to migrate to different computing nodes. Therefore, GUI has the ability to sense when an application has moved. The GUI senses when an application is not present, closes all connections, and goes into a listening state. When an application settles on a new computing node and restarts communications with the GUI, the GUI reconnects and continues performing tasks.

The GUI has two techniques to retrieve graphics from applications. Connector's Frame class has constructors where users may choose between redirecting graphics by sending Frame's function names and parameters or by creating an image local to the application and then sending the image as often as the user desires to the GUI. If function names and parameters are sent to the GUI, then the GUI simply reads those messages and calls the methods locally on actual Java Frame and Graphics objects. On the other hand if the GUI receives an image, the GUI repackages the image and displays it locally on a Java Frame every time a new image is received.

4.3. Sensor Server

Sensor server allows users to interact with sensor networks. There are two ways users may interact with sensors. One way is by retrieving sensor data using the Connector.jar package as if data were local files. The other way is managing and configuring sensors through a GUI connection. A different thread handles each request made by users so that multiple users are allowed to work concurrently.

The sensor server needs a map of the entire sensor network to initiate a session. The sensor file map is located remotely and is accessed using a Connector's FileInputStream object (and by default a daemon). Therefore, when a sensor server initiates, a Connector object must be instantiated as well. Once a Connector is up and running, sensor file maps are retrieved from an FTP server.

When sensor interaction is performed through an application, the sensor server behaves as an FTP server to handle requests. FileInputStream objects from the Connector.jar package residing on a remote application may connect to the sensor server and retrieve data from any of the existing sensors. The sensor server limits the FTP server functionality to data retrievals only. Since sensors provide real time data, the use of FileOutputStream to overwrite data to the sensors is disabled.

Managing and configuring sensors is done through a GUI. In such cases, a sensor server needs to be started with the "-g" option. Sensor server starts a connection to a GUI when it senses this option. A new thread is spawn so that I/O can be handled with GUI without any interruption from remote application requests to read data. Users administering the sensor server may change sensors' configuration. Once changes are made to the sensor configuration, sensor server uses Connector's FileInputStream and FileOutputStream to update remotely stored sensor file maps.

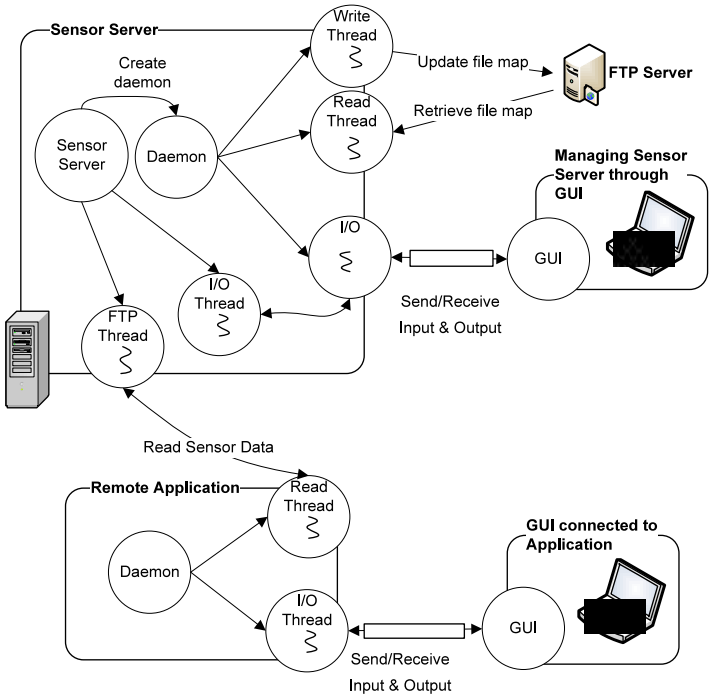

Figure 3 Visualization of Sensor Server

5. Performance Consideration
5.1. Graphics Forwarding
5.2. Sensor-Data Forwarding

6. Conclusions