User Interface - Feature #2200

Feature # 1811 (Closed): implement the AJAX client driver

enable a Jetty web server to be initialized and used within a client process

11/01/2013 09:42 AM - Greg Shah

Status:	Closed	**Start date:**	11/01/2013
Priority:	Normal	**Due date:**	11/11/2013
Assignee:	Marius Gligor	**% Done:**	100%
Category:		**Estimated time:**	40.00 hours
Target version:	GUI Support for a Complex ADM2 App		
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 11/01/2013 09:46 AM - Greg Shah

- Due date set to 11/11/2013

This task is just for time tracking and project management. All task history should be placed in the parent task #1811.

#2 - 11/04/2013 07:01 AM - Marius Gligor

- Status changed from New to WIP

#3 - 11/04/2013 12:21 PM - Marius Gligor

Regarding the implementation on P2J I read your specifications.
Here is what I understood so far:

1. The browser client will open a URL that is serviced by Jetty in the P2J server process. I'll call this the "server jetty". The URL will be something like https://p2jserverhost:jettyport/chui/ (for the AJAX ChUI) or https://p2jserverhost:jettyport/gui/ (for the AJAX GUI).

Q: The "server jetty" is in fact the WebServer or another instance, a new design?

2. Dynamically start a new P2J client process (we already do something very similar in starting "appserver instances")
This will be done by a special launcher process that is there to create child processes under the control of the P2J server
The new client will be started in "browser mode" (ChUI or GUI) and the client code will be the standard P2J java client
That P2J client will start up and will make an initial connection to the server's secure port using the P2J protocol (this is the "client jetty")

Q: The "client jetty" is started similar to ServerDriver#launchAppServer?

3. The P2J client will notify the P2J server about the URL for which the client jetty is listening, this will include a custom/dynamic port number
The server jetty will respond to the original browser request with a page that redirects the browser to the new client jetty URL
The browser will connect to the client jetty, which will send down a page that is the proper AJAX client (ChUI or GUI)
That AJAX client will establish a secure websocket connection to the client jetty

Q: The P2J client is a new process allocated per connection having an embedded jetty server which should deliver
Web Sockets Services and the main page?

Q: The port number should be generated by an algorithm randomly or in sequence?

Q: The jetty protocol is wss over https?

Q: The SSL context is the same as the "server jetty" or if in the directory is registered a SSL context
(keystore and credentials) for this purpose use this one as a first option?

#4 - 11/04/2013 01:04 PM - Greg Shah

> Q: The "server jetty" is in fact the WebServer or another instance, a new design?

I don't see any important advantage to have another instance. My plan was to just have a different URL (than the admin console) but to extend the same WebServer.

If you have any reasons to do something different, let me know.

> Q: The "client jetty" is started similar to ServerDriver#launchAppServer?

Not exactly. That code is related, but I am referring to the server-side code in AppServerLauncher.java. That is the code that really does the launching work. Specifically, look at builder() and launchWorker().

But the development for client launching part will be done in another task than #2200.

> Q: The P2J client is a new process allocated per connection having an embedded jetty server which should deliver Web Sockets Services and the main page?

Exactly.

> Q: The port number should be generated by an algorithm randomly or in sequence?

I don't want to allocate it at all. Let the operating system allocate it dynamically. We can read the allocated port number from the resulting socket instance and send it back to the server as part of the complete URL.

> Q: The jetty protocol is wss over https?

Let's not call it the "jetty protocol", but rather the "web client protocol".

Yes, it is wss from an https session.

> Q: The SSL context is the same as the "server jetty" or if in the directory is registered a SSL context (keystore and credentials) for this purpose use this one as a first option?

Yes, exactly.

#5 - 11/05/2013 01:20 PM - Marius Gligor

- % Done changed from 0 to 10

Today I started to design and implement the AJAX Client Driver.
So far I designed the embedded jetty SSL server and a generic
web socket handler used to register web sockets on the server context.

#6 - 11/07/2013 12:56 PM - Marius Gligor

- File mag_upd20131107a.zip added

Web Sockets support - the current design

Each client running on server has an embedded jetty server which offer web sockets support.

package com.goldencode.p2j.ws;

1. WsServer - implements the embedded jetty server.
- The server is running on the default SSL context but if a BootstrapConfig structure is provided the SSL
context will be constructed based on this bootstrap configuration.
- The port number could be specified explicit or could be be dynamically allocated by the OS when the port = 0.
- The host name could be null case on which the server will listen on all available addresses.
- On each server a context path (default /) could be defined.
Example:
https://localhost:8443/
https://localhost:8443/contextPath
- The server provide a method to add Handlers for web resources like HTML pages.
- The server provide a method to register web sockets based on a target string and the web socket class.
- All Handlers are registered into the embedded server context.

2. WsClient - is a contract (Interface) that should be implemented by each client.
He provide two methods for start-up and shut-down the embedded server.
It is possible to transform WsClient into an abstract class that must be extended by the client.
This is not a flexible design if the client implementation should extend another class.

3. TestPageHandler - is an implementation of a Jetty Handler for delivering HTML files resources.
This class does not provide placeholder substitution. If we need placeholder substitutions
just extended this class and override the method handleReplacements in order to provide a custom
code for substitution. (see the TestPageHandler class). Each handler define a target string
which is part of the URL path.

package com.goldencode.p2j.ws.test;

This package provide a simple example of a client which expose web sockets services via an embedded server.

1. TestWsClient is a very simple client implementing the WsClient interface.
An embedded WsServer instance is created, configured and started.

2. TestPageHandler is a Handler for index.html file. The file is parsed and some placeholders are
substituted before to deliver the page. This is basically the main page.

3. TestWsServlet is a Web Socket simple class.
Jetty offer the following methods to create Web Sockets:

```
a. POJO – Using WebSocket annotations.
   b. Implements WebSocketListener interface.
   c. Extends WebSocketAdapter class.
```

This design has been tested by creating an instance of TestWsClient inside the WebServer
class after the web server start statement:

```
new TestWsClient().startup(null);
```

and found to work properly.

On the web browser open: https://localhost:8443/client/page and click on the link inside the page.

I did also tests having two web sockets registered and found to work properly.

#7 - 11/07/2013 06:37 PM - Greg Shah

Initial Feedback

I realize that this is a work in progress and an early version. Thank you for posting it and the useful description of the approach. I will refrain from a full code review because this is still early and will change a great deal.

I like what you are doing in general. There is some very good stuff here. I have the following questions/comments:

1. WsServer has very little to do with WebSockets. Other than the WsHandler inner class and the addWsHandler() method, the code is otherwise quite generic. In fact, it seems to me that it is very, very similar to our current com/goldencode/p2j/main/WebServer. I prefer to maximize common code here, since the server jetty and client jetty will both have similar configuration requirements and will need to support similar usage patterns.

a. I would like to enhance or extend our current WebServer to be used as both the server jetty and client jetty. This eliminates having a 2nd implementation that is so similar to the code we are already maintaining. To the degree that we need to provide more control over the startup/configuration/shutdown of the WebServer class to make it more flexible, we should do so. It also makes it easier to add WebSockets support to the server jetty, which is something will need in the future as we shift our admin console to AJAX.

b. I'd like to see the WsHandler code moved out of that class and put in its own class. The registration code can be hidden there too (to register itself with the WebServer instance). In addition, please call it WebSocketsHandler because the "ws" abbreviation can mean too many different things.

2. I don't understand the need for the WsClient interface. We can always create a helper class for providing common code for the startup/shutdown of the client jetty. It doesn't need to be in the inheritance hierarchy of the client driver. I prefer that approach because there are less layers of abstraction.

3. If I understand the code properly, all the "static" HTML can be loaded from our p2j.jar, right? That is exactly how we want this to work, since the code will only change when we have a revision of P2J. That avoids maintaining separate files in the filesystem which is very good.

4. I like the handleReplacements() support you have provided. It is useful and will be needed in the future. However, for the specific use case in your test code, I think you can use the javascript location object to get the host, port and so forth.

#8 - 11/08/2013 12:10 PM - Marius Gligor

- % Done changed from 10 to 80

- File mag_upd20131108a.zip added

Release candidate after your feedback.

1.a The embedded server was renamed as GenericeWebServer. This class could be extended or embedded.
I changed the old WebServer class to benefit from the new design. Now the WebServer extends
the GenericWebServer and can expose web sockets services.

1.b The Web Socket generic handler is no longer an inner class. The name of the class is PojoWebSocketHandler
I preferred this name because PojoWebSocketHandler already extends the jetty WebSocketHandler class.

2. I called the "jetty client" server Agents. I understood that an Agent is lunched by an instance of a ProcessBuilder class which is designed to create OS processes. I our case an Agent is a Java class having a main method and the ProcessBuilder will construct and lunch a command like:

```
java -cp bin;lib/lib1.jar;lib/lib2.jar com.example.AgentClass
```

```
The agent will run as a separate process having its own JVM isolated from other processes.
    Using the command line arguments passed (server ports and others) as parameters on the main method the Age
nt
    will be able to build an interface in order to communicate with the server via p2j RMI.
    For testing purposes I created a WebSocketAgent which extends the GenericWebServer and have a main method.
    Until the luncher of agents will be designed and implemented I tested this agent in the context of the ser
ver.
```

3. The handler for static pages has been designed to load the html pages for the jar file (a resource on the classpath). Nevertheless other handlers could be desigen to serve other resource from different URI's.

4. I'm using window.location.host in JS instead placeholders, thanks for this hint!
Only the ${context} placeholder is resolved on the handleReplacements().

Please let me know if you agree the new design.
Also I would like to ask you if I should move the classes from my ws package to the main package and perhaps the ws.test package to ws.

#9 - 11/08/2013 01:47 PM - Greg Shah

Feedback

This is very good. Some thoughts:

1. Change the "ws" package to "web". This will be the place for generic web server classes.

2. As you get a real client implementation going, you will not need the test package anymore. You can keep it for now, for development purposes, but the final update should not have it.

3. Make sure you are reviewing the ClientDriver, com/goldencode/p2j/ui/chui/ThinClient (in regards to how it integrates with the "drivers" and also how it does much of the startup processing) and the ChUI drivers (com/goldencode/p2j/ui/client/chui/driver/*). These will help you design the changes for how the ChUI web client should be implemented.

4. In regards to coding standards, we now have a 98 character line limit instead of 78 (docs aren't up to date yet). Also, please make sure that all methods and members are javadoc'd. There will be more details on this later, when the code is nearing completion.

Keep going. It is good work.

#10 - 11/11/2013 10:42 AM - Marius Gligor

Today I did a lot of tests regarding the ClientDriver and ThinClient.
I'm able to starts and debug from my IDE both instances, the server and the client.
Here is what I understood from this tests. Please let me know if something are missing.

1. The ClientDriver is a remote client started from a command line which run on separate process having only one thread, the main thread.

The command line arguments define the ScreenDriver type and parameters.
Also the server host and port are specified as command line parameters.
It is possible to provide the same parameters using a configuration file like client.xml

```
net:server:host=localhost
net:server:insecure_port=3333

client:driver:type=swing_chui_frame
client:chui:rows=24
client:chui:columns=80
client:chui:background=0x000000
client:chui:foreground=0xFFA500
client:chui:selection=0x0000FF
client:chui:fontname=monospaced
client:chui:fontsize=12
```

2. From the command line arguments a BootstrapConfig object is constructed used to create, initialize and start a generic client ThinClient (CoreClient#start).
It's possible to create a remote standalone client running on a separate process like ClientDriver do or a client running within the server process.

3. Next the UI services are initialized: ThinClient#initializePrep(config, driver, single);
An OutputManager instance wrapping a ScreenDriver object is created.
The OutputManager is used to draw on the screen via an OutputPrimitives interface (ScreenDriver#getPrimitives)
Currently p2j has 2 screen drivers:

- ConsoleDriver - or chui native based on NCURSES package.
- SwingChuiDriver - a Java SWING based client.

I tested both screen drivers and I found to work within ClientDriver.

Also a Java applet ChuiApplet client can be used embedded in an html page loaded by a web browser.

https://localhost:7443/chui

According to my tests this client only draw the main window and a cursor on the top left corner.
It seems that this client is not fully implemented.

4. When the ThinClient instance is created the ClientExports and SessionExports (for static methods) interfaces are exported remotely.
(The SessionExports interface seems to be not used remotely. I didn't found any remote import for this interface).
The registered (exported) ClientExports interface is imported by the LogicalTerminal instance an used to do RMI calls to ThinClient.

5. Next the client is authenticated and a security context for this thread is created.

6. Next finish initializing the thin client: ThinClient.initializePost(session, context)
Imports ServerExports and RemoteErrorData interfaces exported by the remote StandardServer used for RMI calls to server.

7. Finally the MainEntry interface exported (registered) on the StandardServer is imported, a ClientParameters is constructed having the converted 4gl application command line parameters and the remote application is started.
In my case Ask#execute is called which is the Java converted version of the 4gl Ask.p

```
// call the application
running = main.standardEntry(params);
```

8. The client code waits here until the user will terminate the running application.

9. When the application is terminated by the user, some clean-up statements are executed and finally the client exits: System.exit(0);

As a conclusion of my tests, I have a question:

I think that we have to design a new ScreenDriver having an embedded server used to communicate with the JS code which run inside the HTML page on the web browser via web sockets.
In this case what kind of UI components are used inside the HTML page?

a - HTML specific tags?
b - JS based?
c - java SWING applet?
d - others?

#11 - 11/11/2013 01:21 PM - Marius Gligor

- Status changed from WIP to Review

- File mag_upd20131111a.zip added

- % Done changed from 80 to 90

#12 - 11/11/2013 01:28 PM - Marius Gligor

- File deleted (mag_upd20131111a.zip)

#13 - 11/11/2013 01:28 PM - Marius Gligor

- File mag_upd20131111a.zip added

#14 - 11/11/2013 03:14 PM - Greg Shah

Before I review the code, I will provide some comments and answers to note 10.

> The ClientDriver is a remote client started from a command line which run on separate process having only one thread, the main thread.

Yes, ClientDriver is a separate Java process that runs in its own JVM, outside of the P2J server.

Yes, it is remote in the sense that it communicates with the P2J server using our own proprietary remote object protocol.

No, it does not only have 1 thread. There is a type-ahead (key reading) thread that supports our requirement to provide asynchronous interrupts for CTRL-C. And the network socket that connects the client to the P2J server has 2 threads (1 reader and 1 writer). The main thread is in fact the primary thread for doing real work. It is enslaved to the "conversation mode" that we implement between the client and server. See the Conversation class in the net package.

FYI, there is a special-purpose mode in which the ServerDriver can be run as both client and server in one JVM. This is not something we use often and no customers use it today.

> The command line arguments define the ScreenDriver type and parameters.
> Also the server host and port are specified as command line parameters.
> It is possible to provide the same parameters using a configuration file like client.xml

Yes, mostly right. I would note that the bootstrap config can be given as a file and any values in it can be overridden on the command line. The idea is to use that for values that are needed for early configuration of the client (or server, since it can be used there too). Generally, we try to push as much configuration into the P2J server's directory.xml (often just referred to as the "directory") as possible. This maximizes central management of the configuration and can make deployments easier.

For programs that use the P2J classes to make their own connection to a P2J server, the bootstrap configuration can be completely built and controlled under program control.

> According to my tests this client only draw the main window and a cursor on the top left corner.

That may be a regression or perhaps it is some applet problem on your system/browser. When I put that applet in, it was working without issues. However, if you were trying it with MAJIC, it would not work today because MAJIC is very dependent upon executing child processes (and other direct filesystem access). That is not possible in the applet approach.

> The SessionExports interface seems to be not used remotely. I didn't found any remote import for this interface

No, it is in use. Please see com/goldencode/p2j/util/SessionUtils.java. It is used to "push" some 4GL session-level configuration down to the client so that to the extent that the UI can render or otherwise need to use this configuration, it is in synch with the state on the server.

> The registered (exported) ClientExports interface is imported by the LogicalTerminal instance an used to do RMI calls to ThinClient.

Almost. I would not call our approach RMI, which is a very specific Java implementation. However, the idea is the same. We have our own "remote object" protocol (sometimes called the "DAP"/distributed application protocol, but mostly just called "the protocol"). This is the equivalent of Java's RMI, but it has better security and it is easier to implement as a developer. It also doesn't need a name registry process to be running.

But otherwise, your description is correct.

Also note that we instantiate some other classes (e.g. FileSystemDaemon, StreamDaemon, ProcessDaemon, MemoryDaemon...) which also register client-side services that can be called from the server. These are the daemons that allow the client-side process launching, file system access and so forth.

The remote object protocol is designed such that either side of the link can export an API. This peer-based or birdirectional RPC is actively used.

> Next the client is authenticated and a security context for this thread is created.

Yes. Note that the context is on the server. On the client, there is only one "session" and the security manager is intentionally a null thing. Since all the actual converted code is running on the server, in a session that has a specific context associated with it, this is OK.

> The client code waits here until the user will terminate the running application.

Not exactly. The main thread will be unblocked to execute remote API calls through our exported objects (like ClientExports). And with validation expressions and UI triggers, called client-side code can recursively call back to the server, which can then call back down to the client and so forth. In other words, there can be a complex stack of operations where the control flow of execution for the program will shift back and forth between the client and server, possibly on a deeply nested basis. Ultimately, these nested calls will eventually return and pop off the stack until the program exits, returning here for the main thread to exit (or to restart depending on the "stop disposition").

> I think that we have to design a new ScreenDriver having an embedded server used to communicate with the JS code which run inside the HTML page on the web browser via web sockets.

Yes.

> a - HTML specific tags?

No.

> b - JS based?

Yes. But this is not part of this task. The short answer is that we will use the Canvas interfaces to directly draw our own "Window" implementation and everything that is inside of it. There will not be any actual browser components or plugins used.

> c - java SWING applet?

No.

> d - others?

No.

#15 - 11/11/2013 03:15 PM - Greg Shah

Unfortunately, I have just realized that all of these discussions have been going on in the wrong redmine task. Please note my original comment in note 1:

> This task is just for time tracking and project management. All task history should be placed in the parent task #1811.

I've been just using the email link to reply back to your posts, but we need to move all this to #1811.

#16 - 11/14/2013 12:22 PM - Marius Gligor

- % Done changed from 90 to 100

#17 - 11/18/2013 05:21 PM - Greg Shah

- Status changed from Review to Closed

#18 - 11/16/2016 12:13 PM - Greg Shah

- Target version changed from Milestone 12 to GUI Support for a Complex ADM2 App

Files

mag_upd20131107a.zip	10.2 KB	11/07/2013	Marius Gligor
mag_upd20131108a.zip	13 KB	11/08/2013	Marius Gligor
mag_upd20131111a.zip	9.09 KB	11/11/2013	Marius Gligor