User Interface - Feature #4912

move UI portions of the web client to the server-side

09/24/2020 09:58 AM - Greg Shah

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #4065: server-side processing of client pl...	**WIP**
Related to Runtime Infrastructure - Feature #4406: server-side REST execution...	**New**
Related to Base Language - Feature #3254: add support for running 4GL on mult...	**WIP**

History

#1 - 09/24/2020 10:00 AM - Greg Shah

- Related to Feature #4065: server-side processing of client platform dependencies added

#2 - 09/24/2020 10:00 AM - Greg Shah

- Related to Feature #4406: server-side REST execution without appserver agents added

#3 - 09/24/2020 10:01 AM - Greg Shah

- Related to Feature #3254: add support for running 4GL on multiple threads in a single session added

#4 - 09/24/2020 11:06 AM - Greg Shah

When the original UI architecture of FWD was designed, there were 2 "near term" targets: ChUI and fat client GUI (Swing). ChUI uses NCURSES/WIN32 console API and Swing uses AWT which in turn uses the native platform GUI API (WIN32/GDI or X-Windows). Interestingly, all of these APIs have a **fixed design dependency** on a **one process one user** approach. In other words, you can't have 1 JVM that supports 2 Swing users or 2 NCURSES users.

We also knew that we needed a separate per-user client JVM to properly process all Client Platform Dependencies. Due to the **one process one user** requirement, this would also be the logical place for the UI code to run.

We always also intended to maximize the code executed in the server. The world was moving to the cloud and the web and to application servers and moving away from fat clients. So we wanted to obtain the benefits of server-side processing and minimize the amount of fat client in the solution. So we deliberately designed the system such that the converted code only exists/executes on the server and the client UI is data driven via an API that splits off the implementation of the UI onto a "thin client" that runs per-user.

The 4GL runs everything as fat client, even their appserver agents are just headless fat clients managed by a broker. They even process their queries in the 4GL client instead of at the database (although with v12 they are starting to add server-side queries; only 35 years after it was common for RDBMS systems). We certainly did not want hat approach in FWD. This thin client idea in FWD adds a great deal of complexity and slower performance to our approach. It has real advantages but there are real costs.

Even in that early architecture, we wanted a 2 level abstraction layer for the UI. The high level layer would be common between ChUI and GUI. The low level layer would be unique to GUI or ChUI. We have indeed implemented this approach. And we achieved our objective of limiting the complexity and code in that low level layer, which we now refer to as our pluggable UI driver. These abstraction layers allow us to maximize common code and they also separate the visualization and the behavior of the 4GL from the low level primitives that implement the specific drivers.

We knew this would allow us to extend our list of drivers over time, first to a web driver which we successfully implemented. In the future, we intend to add a mobile client driver (native Android and iOS).

Interestingly, these new UI modalities (like the web) were designed in a cloud/web/server world. They assume that much of the UI is running on the server and only a lightweight client is running where the user is located. In other words, the web client (and mobile too probably) is a natural fit to allow multiple users to execute in the same server-side process. This task is meant to explore the idea that we can move the Java portions of the

web client UI into the FWD server itself.

- The JS portions would remain as they exist today, but they would communicate to the FWD server instead of having the Jetty down in the client JVM. This means we have work to do to host multiple clients in the FWD server Jetty. This may have some nice benefits if we can avoid redirection and keeping the same origin.
- All the UI classes would be made safe for execution on either the client or in the server.
  - When run on the server, we would need to allow multiple instances of ThinClient and all the other singleton classes.
  - Static data may need to be mad part of instances or made context-local (we want to limit context-local if possible because it is costly).
  - Any implicit dependencies on only one client must be resolved.
- We already have single process mode where the Swing client can be executed in the same process with the FWD server making a kind of single-user mode that simulates the benefit of this idea. I'm not sure if it still works, I put it in with revision 10099. By setting process:arch:single in the bootstrap config, you can use the ServerDriver as a unified client + server. At the time, the biggest part of that work was to implement a mechanism to have a local redirector for the proxy usage so that code that normally works with a RemoteObject proxy could work directly with in-process instances. In addition to making sure this still works, we would want to ensure that it is really efficient. I worry that there is still some cost to this layer.

The core benefit here is that it should perform significantly faster than our current split design. All of our customers that have GUI apps care about the web client as their primary objective. This means that all of these clients will get a huge benefit. This would become the most common approach for our GUI implementations.

We also have some work envisioned in #4065 where we optionally would move non-UI client processing to the server process. The UI stuff for the web client is safe to move. There is no negative security implication that I can think of. But the non-UI stuff does have security implications and would need extra security controls to make it possible. In addition, some 4GL code approaches for non-UI client dependencies can't be handled without a real client JVM (for example, using non-reentrant native library calls). In the future, when this task and #4065 were both complete, some applications could be migrated to an approach where the client JVMs are no longer used. That would also be a huge benefit (much simpler to implement, debug and support). But for the purposes of this task, we are not assuming a dependency on #4065. So we would still have a ClientDriver and a client JVM, but it would only be used for non-UI stuff.

I know this is not something to be done this week. :)

I wanted to start the discussion in case we decide to move ahead. What are the issues you can see with this idea? Do you see any items that make it not possible? Is there work that I haven't listed above?

#5 - 09/24/2020 11:52 AM - Hynek Cihlar

Greg, wouldn't it make more sense to think of this problem the other way around? Keep the client processes as they are, with their os dependencies, spawning, etc., and move the server legacy code execution to the client processes? This seems to be a lot less work to me, while keeping the benefit of process spawning. With spawning the system could keep the ability to scale out and handle virtually unlimited number of active users.