Base Language - Feature #6343

class-based appserver exports

05/10/2022 01:08 PM - 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 #6373: direct Java object access to conver...	**New**

History

#1 - 05/10/2022 02:39 PM - Greg Shah

The 4GL enables the export of external procedures, internal procedures and user-defined functions over an appserver connection. The problem is that OO classes cannot be used over an appserver connection. One cannot instantiate objects or call methods (instance or static) over appserver.

In more recent years, the strategic approach for 4GL server applications has shifted to exporting of REST calls. The problem here is that if one is heavily invested in appserver APIs, then a modern OO design cannot be directly supported. Instead, one must generate a layer of procedure proxies whose only purpose is to instantiate objects and call methods on those objects.

We have a customer that does this extensively, generating tens of thousands of these stub proxies, which complicate their code and which slows down the solution. In FWD, we can eliminate the complexity and the performance hit. The idea is that we will allow the "remoting" of OO 4GL objects. This will be allowed in the FWD appserver client. The core facility itself should be directly available in our RemoteObject approach, even if we make it easy to access via our appserver client.

On the client side, the RemoteObject approach already makes it easy to call a remote object that is gotten via RemoteObject.obtainNetworkInstance(). The problem here is that on the server side that object is implemented in one of two ways:

- As a singleton instance of a class that implements the interface(s) of the proxied object (RemoteObject.registerNetworkServer()). OR
- As a set of static methods that "implement" the interface(s) of the proxied object (RemoteObject.registerStaticNetworkServer()).

Either way, our current RemoteObject approach:

- Does not have a mechanism to create a new instance of a server-side object instance for each proxy created by the client side.
- Does not have a mechanism to call static methods on the server side. I don't know that this part is needed. It isn't really compatible with the idea of getting a proxy for an interface, so we will probably ignore this unless a requirement for it pops up.

In regard to the per-proxy instance approach, I think we need a mechanism for the server-side code to register a factory class. This factory would be called whenever a new instance is needed and it would be able to do any initialization or processing needed before the instance is returned. This will be needed for the specific customer's approach (they have their own factory approach which is called by their "procedure stubs" implementation).

#3 - 05/17/2022 10:37 AM - Greg Shah

- Related to Feature #6373: direct Java object access to converted Java code from in-JVM non-converted Java code added