Base Language - Feature #2145

eliminate program-name limitations

05/02/2013 07:24 PM - Greg Shah

Status:	Closed	**Start date:**	08/20/2013
Priority:	Normal	**Due date:**	08/21/2013
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	16.00 hours
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 01/30/2014 01:05 PM - Greg Shah

To understand the limitations of the P2J PROGRAM-NAME implementation, please see the JavaDoc for EnvironmentOps.getSourceName().

#2 - 01/30/2014 01:46 PM - Constantin Asofiei

This suggests that looking into the java stack trace is not correct. Instead, ControlFlowOps should keep a stack for the legacy code calls and interogate that.

Other cases to check:
- are user-defined function calls reported too?
- what about triggers?
- how are super procedure calls handled? Does this report the original name (at the SOURCE-PROCEDURE where the procedure or user-defined function is invoked)?
- how are RUN SUPER and SUPER calls reported?

#3 - 01/30/2014 02:33 PM - Greg Shah

> This suggests that looking into the java stack trace is not correct.

Yes. At the time it was originally coded, the converted business logic for RUN was directly invoking the execute() method on instances created from hard coded class names. Since then, you have fully implemented the RUN support properly (i.e. completely indirect calling mechanism, managed by the runtime).

> Instead, ControlFlowOps should keep a stack for the legacy code calls and interogate that.

Agreed. This will allow a 100% correct program-name implementation.

Yes, your other points are also valid. Some more investigation is needed.

#4 - 02/15/2014 09:06 AM - Constantin Asofiei

This one proves a little more challenging: the procedure names in error messages and the procedure's FILE-NAME and NAME attributes follow the PROCEDURE-NAME function. This means that we need to report the relative name (used at the RUN statement) on error messages and whenever the FILE-NAME and NAME attributes are checked for a procedure. Internally (i.e. when interrogating SourceNameMapper APIs) we need to use absolute names (this can be done from the referent's class name, using SourceNameMapper.convertJavaProg): my concern is that, if the PROPATH gets modified, we might resolve the relative name to a completely different program.

#5 - 02/15/2014 01:20 PM - Greg Shah

> my concern is that, if the PROPATH gets modified, we might resolve the relative name to a completely different program.

I don't fully understand your concern. Isn't PROPATH assignment supposed to cause this behavior in the 4GL?

#6 - 02/15/2014 01:24 PM - Constantin Asofiei

Greg Shah wrote:

> > my concern is that, if the PROPATH gets modified, we might resolve the relative name to a completely different program.
>
> I don't fully understand your concern. Isn't PROPATH assignment supposed to cause this behavior in the 4GL?

The idea was that internally I can't use the relative name and call i.e. SourceNameMapper.getParameterModes, each time I need it. For this reason, I need to keep two names: for internal usage, the absolute name (resolved and saved when the RUN statement is called for that program). For legacy usage (i.e. error messages, PROGRAM-NAME, etc), the name used at RUN statement (relative name, in my terms).

#7 - 02/18/2014 01:50 PM - Constantin Asofiei

- File ca_upd20140218b.zip added

This should solve the SOURCE- and TARGET-PROCEDURE cases (when invoking an external program or a trigger). Also, it implements a compatible PROGRAM-NAME function.

I'm putting this through regression testing now, to catch any problems, but I still need to finish testing the resource-delete, super-procedure and other relevant cases, to make sure I do not regress anything.

#8 - 02/20/2014 09:21 AM - Constantin Asofiei

- Status changed from New to WIP

- File ca_upd20140220d.zip added

This solves the OOME and the SOURCE- and TARGET-PROCEDURE deviations. Is going through testing now.

#9 - 02/20/2014 09:58 AM - Greg Shah

Code Review 0220d

I am OK with the changes. As always with the procedure processing, this stuff is pretty nasty. Great work!

We don't have much choice in coding so much dependence on reflection and the old approach was also dependent (but broken in 4GL terms). So this is a good improvement, but we need to keep that dependence in mind. I doubt the inner class naming mechanism will change however when we implement the new Java 8 lambda expressions, this code will possibly need revisions.

The only other thing I wonder about is why we need to use findRootEnclosingInstance() in TriggerDefinition. Doesn't outer always properly reference the root enclosing instance?

#10 - 02/20/2014 10:56 AM - Constantin Asofiei

- % Done changed from 0 to 80

- Status changed from WIP to Review

Greg Shah wrote:

> We don't have much choice in coding so much dependence on reflection and the old approach was also dependent (but broken in 4GL terms). So this is a good improvement, but we need to keep that dependence in mind. I doubt the inner class naming mechanism will change however when we implement the new Java 8 lambda expressions, this code will possibly need revisions.

I'm not sure what you mean here. The only dependence on reflection is the "java-style calls of legacy user-def functions", when we need to determine what function is being invoked.

> The only other thing I wonder about is why we need to use findRootEnclosingInstance() in TriggerDefinition. Doesn't outer always properly reference the root enclosing instance?

I was thinking if someone manages to pass the anon inner block instance instead of ExternalClass.this; the conversion code works properly (it emits ExternalClass.this).

#11 - 02/20/2014 11:05 AM - Greg Shah

> The only dependence on reflection is the "java-style calls of legacy user-def functions", when we need to determine what function is being invoked.

For example, the code is very dependent on the the naming scheme of inner classes ("this$" and so forth). Such hard coded references may need to be changed when we move to lambda expressions.

> I was thinking if someone manages to pass the anon inner block instance instead of ExternalClass.this;

OK, it makes sense.

#12 - 02/21/2014 04:10 AM - Constantin Asofiei

- % Done changed from 80 to 0

- Status changed from Review to WIP

Regression testing showed problems related to calling the hand-written java code: we need to determine the policy of invoking such code. The problems arise from the fact that there are java-style procedure calls which bypass the ControlFlowOps APIs. We need to choose from:
1. Moving all java-style procedure calls to ControlFlowOps APIs (originating both from legacy code and from hand-written java code).
2. Use a similar way to java-style user def function calls: mark calls originating from ControlFlowOps and, if the call is not originating from ControlFlowOps, do what ControlFlowOps does. Although this will make it nice for hand-written java code, it will duplicate the logic from the ControlFlowOps.invoke... I don't like this.
3. Try to move all the pre-invocation logic related to maintaining the procedure stacks (source, target, etc) into the scopeStart/scopeFinished logic. At this time, I'm not even sure if it is possible.

#13 - 02/21/2014 08:31 AM - Greg Shah

Option 3 is the best solution, IF it is possible.

A modified option 1 may be the next best approach. The idea is to provide 3 new ControlFlowOps APIs for "direct" calls to converted code. There would be the following:

- invokeDirectExternalProcedure
- invokeDirectInternalProcedure
- invokeDirectFunction

The idea of "direct" is to suggest that the calls are made with a specific Java class (and method names for the last 2) instead of being determined at runtime. I prefer not to do this because it kills compile-time type safety, makes things more indirect and is generally nasty. But it is better than forcing hand-written code to use legacy names and the fully dynamic CFO approach.

Or perhaps option 2 would be not as bad if the CFO code was refactored such that it could be reused for this case and thus there was common code.

#14 - 02/21/2014 09:15 AM - Constantin Asofiei

Greg Shah wrote:

> Option 3 is the best solution, IF it is possible.

I think I will go ahead with this one. The only info which currently can be determined only from CFO side is this:

1. the "superCall" state
2. the "relative name" used to invoke the procedure/user-def function
3. the target procedure, i.e. which external program was assumed the internal entry was part of, not the actual location of the resolved internal entry.

If I push this to some ProcedureManager stack, ProcedureManager's scopeStart will just need to interogate the top of this stack, before pushing the data to the other stacks (i.e. target procedure, source procedure, etc). Beside this, BlockManager will be changed to push correct superCall/relative name/target procedure, in case of java-style calls (what we are doing with BlockManager.functionBlock will be done for internal and external procedures, too).

#15 - 02/22/2014 05:05 AM - Constantin Asofiei

- File ca_upd20140222b.zip added

This version allows java-style calls for internal and external procedures and centralizes the maintenance of procedure system handles and other proc-related data. Is going to regression testing now.

#16 - 02/22/2014 09:47 AM - Constantin Asofiei

Failures during testing of 0222b.zip: there are cases like this (MAJIC uses them):

```
public static void something()
{
   externalProcedure(...);
}
```

When we need to determine the root enclosing instance, there is no such thing, as the external-program call originates from a static method. But I don't need to fix this in BlockManager.checkJavaCall... this is more like a case when instead of externalProcedure it should have been internalProcedure, so that THIS-PROCEDURE at the callee is inherited from the caller. I'll fix it in MAJIC.

#17 - 02/23/2014 04:14 AM - Constantin Asofiei

The 0222b.zip and 0222e.zip (from #2248) have passed regression testing, after checking the cumulative regression testing results.

#18 - 02/23/2014 04:18 AM - Constantin Asofiei

- Status changed from WIP to Review

- % Done changed from 0 to 90

#19 - 02/24/2014 10:49 AM - Greg Shah

Code Review 0222b/e

I am fine with the changes. This is another nice improvement/refactoring of the code such that the logic is better centralized.

Have you retested the "standalone" super proc/persistent proc testcases? The only dangerous part of the change is really whether it properly pushes/pops state properly for all cases.

Otherwise, I am fine with you checking this in.

#20 - 02/24/2014 11:50 AM - Constantin Asofiei

Greg Shah wrote:

> Have you retested the "standalone" super proc/persistent proc testcases?

Yes, the tests are OK.

0222b.zip was committed to bzr rev 10477
0222c.zip from #2228 (~~note 71 has a typo, 0222c.zip is the correct one~~) was committed to bzr rev 10478

#21 - 02/24/2014 12:11 PM - Greg Shah

- % Done changed from 90 to 100

- Status changed from Review to Closed

#22 - 11/16/2016 12:07 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features

Files

ca_upd20140218b.zip	178 KB	02/18/2014	Constantin Asofiei
ca_upd20140220d.zip	178 KB	02/20/2014	Constantin Asofiei
ca_upd20140222b.zip	178 KB	02/22/2014	Constantin Asofiei