Kepler - Bug #4483

Module dependencies in MoML files

10/21/2009 04:15 PM - Aaron Aaron

Status:	Resolved	**Start date:**	10/21/2009
Priority:	Immediate	**Due date:**	
Assignee:	David Welker	**% Done:**	0%
Category:	core	**Estimated time:**	0.00 hour
Target version:	2.3.0	**Spent time:**	0.00 hour
Bugzilla-Id:	4483		

Description

To reproduce:
Save a KAR from the WRP suite
change to the kepler suite
on startup you see the error:
Error encountered in: <property name="TOP Provenance Recorder" class="org.kepler.provenance.ProvenanceRecorder">
etc....

I used to think this was an issue with the startup building of the library and that dependencies in the KARs was the problem. But it turns out to be much more malicious. It is actually a dependency from the MoML file itself. Because the WRP suite has the Provenance recorder included in the moml, when the provenance module is not around the error is displayed and the workflow can not be used. So this means that any moml file generated by the WRP suite cannot be run in the Kepler suite. Unlike in Reporting where the reporting files are separate from the MoML itself and therefore the MoML can still run in the Kepler suite.

So the task here is to figure out how to add a dependency on the provenance module to all the workflows that are created from the WRP suite. The ability to do this I think is not built into the KAR design and I'll have to figure out how to do it.

Related issues:

Blocks Kepler - Bug #4516: Module Manager needs ability to install modules ne...	**Resolved**	**10/29/2009**

History

#1 - 10/21/2009 04:17 PM - ben leinfelder

You can still execute the workflow after skipping those class not found exceptions when you open the workflow - but it is inelegant

#2 - 10/21/2009 05:10 PM - Aaron Aaron

Definitely inelegant. Down the road could it be a serious problem though? Should we disallow even the attempt to open a workflow that has dependencies on modules that aren't there? Or is the ptolemy error popping up when the cache gets built OK?

#3 - 10/21/2009 05:15 PM - David Welker

This is the reason I believe we need to store the active module configuration inside the KAR is some manner. Then when a KAR is loaded, the user would have the opportunity to either change their configuration to match the configuration in the KAR or take their chances.

#4 - 10/22/2009 12:15 PM - Aaron Aaron

Storing the active module configuration in the kar would certainly do the trick. I think it may be too restrictive though. Imagine someone has a very insignificant additional module installed in their system, then anyone wanting to open any KAR that they saved would have to also install that insignificant module in order to use the KAR.

At the moment, a KAREntryHandler is responsible for returning a KAREntry that has a dependsOnModule attribute, so it is possible to individually specify module dependencies on each entry in the KAR. The problem becomes how does a module tell the ActorMetadataKAREntryHandler that it should add an additional dependency to the MoML file when it is time to save the MoML. The purpose of the EntryHandler design is to completely separate the saving and opening of individual entries in a KAR. So I see two solutions at the moment:

1) Use a common StringAttribute to designate module dependencies for NamedObjs. Here a module, such as provenance, when it adds a module specific entity to the workflow, would also need to add the common StringAttribute designating the module dependency. The ActorMetadataKAREntryHandler would then know to look for that StringAttribute and add the dependency to the KAREntry.

2) Actually parse out all of the class names from the xml of a named object and search for them in all of the modules and dynamically determine which modules the workflow (or any NamedObj) depends on. This I think would be very slow??? Maybe there could be a fast way to do it if we built an index of all the classes contained in a module when the module is built?

#5 - 10/22/2009 12:30 PM - David Welker

Good point. But this can be solved by simply making the current active configuration the default. If the workflow engineer has reason to think that not all the modules that are currently active are needed, they could simply turn some of them off. Obviously, it would be up to the workflow engineer to test his assumption that not all the modules are needed before distributing the workflow.

#6 - 10/22/2009 12:33 PM - ben leinfelder

a class-based decision about which modules are required won't work in cases where we are using class overrides...

#7 - 10/22/2009 12:56 PM - David Welker

Your point about overrides is not correct. If you are using a module that includes an override and your workflow uses that class, then you know there is a dependency on the module that contains the override.

However, automatic dependency detection will not work in cases where reflection is used, unless we develop a specific means of detecting reflection within the code. Further, even if we solved this reflection problem, we could not detect module dependencies on modules that contain required non-java third party tools where the module does not also satisfy any java class dependencies.

I don't think there is any good way to do this automatically.

Let's just have the default be that all the modules in the active configuration in which the workflow was developed are required and let the workflow engineer specify that certain modules aren't in fact required if that is known to the workflow engineer.

If the workflow engineer developed comprehensive tests, it might be possible to determine dynamically what modules are really required using a sophisticated nightly build. But this sort of infrastructure would take time to develop and would imply a lot more work for workflow developers in developing testing mechanisms that exercise the workflow in a fairly comprehensive way. Even after all that, if the tests were not good, this sort of automatic determination would fail.

Overall, I don't think this is too bad. The worst case scenario is that someone downloads a module that is not really needed and has to restart Kepler in order to run the workflow. Most of the time, all of the modules in the current active configuration in fact will be required. In cases where that is not the case, inconvenience to the user can be avoided by workflow engineers indicating that certain dependencies are not in fact required.

#8 - 10/22/2009 01:00 PM - Aaron Aaron

David, what do you think about putting the responsibility not on an automated class/module detection system or on the workflow designer but on the developer. As solution 1) from comment #4 would do.

#9 - 10/22/2009 02:19 PM - David Welker

Hi Aaron,

I think it is an interesting idea.

So, you are basically saying that when an actor is added to the workflow, that actor should know all the modules it depends on and somehow register them, right?

But what if a dependency could actually be satisfied in the alternative by multiple modules? i.e. Either module X or module Y will satisfy the dependency. What if a specific version of a module is necessary to satisfy the dependency?

How does this fit into versioning and the concept of time? I can say that X is necessary, but it might be only X after version A but before version B that satisfies my dependency. It seems the developer needs to know not only that X satisfies his dependency. The developer needs to know which versions of X satisfy his dependency and which do not.

What is the advantage of your proposal? One advantage is that perhaps workflow developers have to know slightly less about their dependencies. In contrast, actor developers have to be constantly aware of precisely what they depend on. But it also seems less flexible. It is possible that the needs of the actor could be satisfied in multiple different ways and those solutions might exist in different modules that could be specified in the alternative.

Also, workflow developers are going to be very aware of modules in general. If they do not want to figure out what modules they minimally depend on, then they can just go with the default (i.e. the modules they were actually using when they developed the workflow) at the cost of minor inconvenience to the end user of those workflows.

So, what I see are some trade-offs. There are advantages and disadvantages to both approaches. I think your proposal has some interesting potential in terms of being automated, especially with some clever metadata. (i.e. if your module depends on another module because reflection or third-party libraries, then it needs to be specified explicitly.) These are both fairly low level approaches, however.

What do you think?

Yet another approach we might take is by abstracting this stuff out and using the concept of services. We could require modules to say they depend on service X, Y, or Z, which must be satisfied through well-defined contracts and then other modules will say whether they satisfy X, Y, or Z. So, a module that in one context depends on A to provide X, B to provide Y, and C to provide Z might in another context depend on only D, which actually provides all of X, Y, and Z. Implementing a service would be analogous to implementing a service, except that a service would actually consist of potentially multiple contracts or interfaces instead of just one. Also, things like the module manager could be aware of services, so that if you have a module that wants service X, but there is no module that provides service X, it freaks out. Also, services probably would need to be versioned so that you may need something that implements service X.5 or later and older versions such as X.3 will not work.

At this point, I would also be interested in considering how OSGi handles this problem, at least for inspiration. It turns out that JSR 277 is inactive and will probably die. Check out this post from Jeet Kaul, Vice President of Sun Microsystem's Client Software Group for more information here: http://blogs.sun.com/meetjeet/entry/osgi_vs_jsr_277. I think it is really lame that Sun abandoned this project. They seem to be very indecisive. But, if JSR 277 is dead (probably forever) then our alternatives are either OSGi or we solve our module problems with custom solutions (like we have been doing up to this point).

Would OSGi help us solve this problem? If so, how? It is an interesting question that I hope to research in the days ahead.

#10 - 10/22/2009 02:39 PM - David Welker

Besides OSGi, another technology to consider for inspiration is Project Jigsaw. Check out this link here for more information: http://openjdk.java.net/projects/jigsaw/

#11 - 10/22/2009 02:47 PM - Aaron Aaron

Yes, OSGi allows dependencies to span multiple versions. Definitely read the OSGi spec if you haven't already. Would be interesting to see how Jigsaw and JSR 277 handle it as well. We certainly need that capability. For now I will implement solution 1) using Strings as the transport mechanism, this will make it easy for us to add versions and version spanning later on (which are specified in the syntax of the module-version string in the case of OSGi).

#12 - 10/22/2009 03:27 PM - David Welker

If you are going to implement a temporary solution, I think you should instead
just save the current active configuration in the kar file. That is much
simpler for developers and will always work.

Also, I have been thinking about this, and I think part of the problem is
basically Kepler restart. My concern with a module specifying incorrect
dependencies is if those incorrect dependencies lead to an unnecessary restart
of Kepler. You could imagine a situation where two workflows are in fact
compatible, but when you load one or another you find that you need to restart.

I think you should hold off on the idea of having a String transport mechanism because then we have to think about how we are going to specify dependencies and I don't think we should be specifying them at such a low level without thinking about the implications. Saving the active configuration to the kar, by contrast, is both very simple to implement, requires not extra work for developers, and is very simple to back out of if and when we come up with a better solution.

#13 - 10/22/2009 04:13 PM - Aaron Aaron

I've implemented and checked in a prototype of solution 1. At the moment, it only solves the original bug if you clean your cache before restarting the kepler suite after saving a kar with the wrp suite. The next step is to recognize the dependencies without cleaning the cache and to add a visual aid to the user in the component library so they can see a KAR that has broken dependencies and right click on it to see or even get the needed modules.

#14 - 01/13/2010 03:35 PM - Chad Berkley

This visual icon is in the tree. Downloading now works when you right click on the item and tell it to get the dependencies. Just need to hook up a system to reload the actor tree after the dependencies have been met.

#15 - 01/28/2010 12:28 PM - Aaron Aaron

Chad I've added code to ImportModuleDependenciesAction that rebuilds the library after new modules are downloaded. The action still doesn't seem to work but the code I added to refresh the library works well in other situations so I'm lead to believe there is still trouble with loading the new modules. I also do not notice any change to the gui that should be visible if the new modules were loaded even after I restart Kepler. So I'm passing the buck back to you on this one...

#16 - 02/22/2010 03:57 PM - Chad Berkley

pushing to post 2.0 release.

#17 - 02/02/2011 11:57 AM - Derik Barseghian

We no longer write any module dependency info into the moml, but certain things from modules may still be written into the moml, e.g. provenance recorder and reporting listener, as mentioned in the original bug.
I think one acceptable way to close this bug is to simply add a warning to the export action (the way you write moml now) stating that export is lossy, that module dependency info is not kept -- e.g. if you export to moml from within the reporting suite, opening that xml file in another suite that lacks reporting will give errors.

#18 - 05/02/2011 03:59 PM - Matt Jones

Moving dependency tracking bug to 2.3 as an issue to be resolved for this release.

#19 - 05/02/2011 04:52 PM - Derik Barseghian

Dan made some changes (~r27498 to r27503 i think) so the provenance recorder and reporting listener are no longer added to the MoML, so this can be closed; afaik no non-vanilla modules are modifying the MoML for their own purposes now.

This bug, along with bug#5099, provide useful background info should we try to find a better solution to dealing with module dependencies; our current solution promotes downloading modules when they may not truly be necessary. Do note that as of 2.2 the user is allowed to dismiss the import module dependency dialog when attempting to open a kar saved beneath different modules than are currently active by selecting the "force open" option.

#20 - 03/27/2013 02:27 PM - Redmine Admin

Original Bugzilla ID was 4483