Kepler - Bug #1342

need R actor

02/10/2004 08:56 AM - Chad Berkley

Status:	Resolved	**Start date:**	02/10/2004
Priority:	Immediate	**Due date:**	
Assignee:	Dan Higgins	**% Done:**	0%
Category:	actors	**Estimated time:**	0.00 hour
Target version:	1.0.0alpha6	**Spent time:**	0.00 hour
Bugzilla-Id:	1342		

Description

create an actor to execute R jobs via kepler. Later, establish a GridService that can handle this for us.

Related issues:

Blocked by Kepler - Bug #2043: create suite of common statistical actors using R	**Resolved**	**03/11/2005**

History

#1 - 05/25/2004 09:39 AM - Matt Jones

STart with running it locally. Start with command line invocation, but it would be much better to directly call the R dynamic library via a JNI interface. Test by implementing the sub-sampling algorithms needed for the GARP pipeline. Need to determine how to dynamically change the I/O signature of the actor when a specific R script is bound to the actor -- this is a general problem with all scripted actors, and is similar to the problem in the EML200 parser and the WSDL parsing actors.

#3 - 01/21/2005 11:11 AM - Jing Tao

create seversl specific R examplss

#4 - 02/22/2005 03:32 PM - Matt Jones

A limitied R actor is now working on Windows and Linux and Mac (although limited graphical capabilities on Mac because of lack of jpg support) that uses the commandline actor to execute an R script. It however requires a custom workflow to stage the inputs and outputs properly for each script. We need to eliminate the need for this 'plumbing' in R to support the inputs and outputs that vary with each R script.

One approach is to have an interface def language for R scripts that basically provides a formal documentation of the R script inputs, outputs, and function. This could be very similar in nature to what Chad and I did in Monarch previously. Basically a 'WSDL' for R. The interface would list each of the inputs that the script expects, its formal type, and how it expects to access it (e.g., from a particular file, or through a particular stream). It would do the same for the outputs, listing each output with its type and how delivered (e.g., file with name '/tmp/graph1.jpg' as output). The 'R actor' would then parse this information and expose a dynamic set of prots with appropriate types, just as the WSDL actor does. A user could then use an R script just by having the R actor read this interface description. The interface description could even contain the R script itself, which would make it a self-contained unit that could be transported as an actor (only needing the 'R' actor to interpret and run it). This is basically the framework we had in Monarch.

This system also would enable us to have an 'R import dialog' in which someone can list the inputs and outputs of an R script, associate them with files or streams, and paste in the R script, which would then generate an R interface def and allow it to be stored in the library tree as a specialized R actor. This interface already partly exists in the 'New actor' functionality that chad developed for Kepler, in that it allows the i/o signature of an actor to be stubbed out.

In this scenario, the R actor would get re-written to only contain one parameter, which is a pointer to the R-script interface file. From that file it can get everything it needs to ingest the inputs, run the script (through

commandline or otherwise), and expose the outputs on the output ports with the proper types. This is exactly analogous to how the EML actor works and the WSDL actor works, so we have existing code that shows how this can be done.

#5 - 03/03/2005 11:41 AM - Dan Higgins

In his recent comments added to the R-bug in kepler/Bugzilla (#1342), Matt suggests that we need to create some sort of interface definition language for R scripts (a sort of WSDL for R). His suggestions triggered some additional thought(s) about this subject which I will attempt to document here.

Consider first R and just what data types it uses. In general, R is a functional language that operates on named data structures. Probably the most common data structure is the R vector which is a sequence of numbers, booleans, or strings. R is not strongly typed, meaning that content type of vectors need not be specified prior to use; types are determined by the values. All sorts of functions can be applied to these vectors and the ability to apply mathematical operations to a vector eliminates the need for many explicit looping operations required in lower level languages (C, Java, etc.). Of course, R also has other data types like the data frame, which is basically a collection of vectors (think of columns in a table). Often, analyses are started by importing a table from a file into a dataframe and individual columns are pulled from the dataframe as vectors and operated on with R functions. (Incidently, the R Reference Manual Base Package is a 700 page book just documenting functions in the R base package; R has a 'lot' of functions!) .

A common R input is a vector (or table) which is assigned a name when imported and the name is later manipulated by various functions. The actual import is typically a string entered from the keyboard or a datafile (table) read from the file system.

R output is usually text displayed on the screen or a plot created by R in a graphics device. Functions often display some summary results; plots are the results of specific functions and appear on 'graphic devices'. To export to other systems, the plots generally need to be written to graphic files.

So what type of ports would an R actor have if used in Kepler? Tables could of course be referenced as filenames or text streams. But R vectors might well be input as what are called 'arrays' in the Kepler/Ptolemy expression language. And output ports would presumable be text strings or file names, or perhaps kepler arrays/records.

Note that some of the Kepler token types are similar to R data types. The kepler token array corresponds to the R vector. A Kepler record can attach a name to kepler array, so an array of kepler records could be converted to/from an R dataframe.

Now, an interesting 'feature' of Kepler (inherited from Ptolemy) is that one can add ports to any(?) actor by
simply using the 'Configure Ports' menu item of the context (popup) menu. In fact, adding ports to many actors is meaningless because the actor is not written to respond to new ports! [And it has been noted that adding ports, configuring the actor, changing the actor name, etc. should probably all be done in a single dialog.]

The one actor where adding a number of new input ports is very useful and meaningful is the Expression actor. When new input ports are added and named, the name can then be used in the 'expression' that appears inside the actor! The expression is just a function that operates on the named input parameters and creates a result. [Note that the Expression actor has a single resulting output (although this may be a vector, matrix, etc.), although one can add additional output ports to an Expression actor, they really are meaningless.]

One might think of an R script as an extension of a Kepler expression. Certainly the input data could be treated as in the Expression actor. Simply create input ports, and give these ports a name and (perhaps) a type. For example, Kepler arrays could be mapped directly to R vectors and the R vector given the port name. That port name would be used in the R script to operate on the input data. [Actually, the R actor could automatically examine the token type to determine how to convert to an R type. A string could just be entered as an R command; a fileParameter could be read as a dataframe; an array as an R vector.] Do we really need a special interface definition language to describe the Kepler inputs to an R actor? My thought is 'No' --- the MOML port specification is sufficient.

But what about output? Much of the output of R is just text which appears when a command (or script) is entered and executed. One output port could thus just transmit the R text output stream (either as a single token or as a series of tokens - see the ExpressionReader actor). Another type of output is an image (plot) created by R. This is probably in the form of an image file name (e.g. a jpeg file); R creates the image file and its name is returned to Kepler. (Most likely, Kepler then uses another actor to display the result.) In some cases, the desired output may be in the form of Kepler arrays, matrices, or records. These might be needed for further processing in a workflow. The actor thus needs a method for converting R objects (especially vectors) to Kepler arrays, etc. So, one could consider simply mapping R objects to Kepler ports. The port name would be the R object name and the port type would determine how the R object is converted to a Kepler token.

So I guess that I am not convinced that we need a 'formal' WSDL for R, at least for use only within Kepler. I think that any data needed to define I/O can be included in Port definitions, thus avoiding the needless complexity of another interface definition language. (We might need a better dialog for setting port parameters, however!). As I see it, the creator of a general purpose R actor understands R well enough to create an R script which is like the expression in an Expression actor. Input ports are added to represent inputs and port names can be used in the script. Output ports would be similarly defined, with perhaps a standard port for text output.

#6 - 03/11/2005 03:19 PM - Matt Jones

OK, to summarize the phone call from last week. We all agree an effiective R actor needs to provide the following capabilities:

1) Easily import existing R scripts by * letting user choose script * user defines ports for all script inputs and outputs * actor searches script and automagically provides plumbing to

2) Execute the R script in an existing workflow without any additional configuration on the part of the user. That is, if a workflow with an R actor in it works on one Kepler instance (e.g., Windows), it should work on all other Kepler instances without modification (ie, no path fixes, temp file renaming, etc)

The key here that we are trying to acheive is ease of use through transparent functionality. If someone binds an R script for Linear Regression to the R actor, they should be able to register that in the user library and from then on all other users should be able to use that regression actor without modification (just drop it in place, connect the ports, and run).

Handling (1) the import of actors will probably require a new GUI component, or at least a specialized configuration dialog that can take the R script as an attribute. The process might look like this for importing an R script:
a) User drags generic R actor onto canvas
b) user opens the configure dialog for that actor and
b1) pastes the R script of interest into the script field
b2) creates ports for all of the inputs expected by the R script

Ideally that would be it and the R actor would be able to map between the port name and the tokens in the script. For example, assume a script needed two vectors of floats as input named 'x' and 'y' in the script and produced a scatterplot in an image file of type image/gif called 'scatterplot'. When the user imports the actor, he would create two input ports named x and y of type array of float, and one output port named 'scatterplot' of type 'Object' (I'm not sure about that last type decision). The R actor would then automatically and without further configuration be able to take data coming in on the two input ports (x and y) and provide it to the exectuing R script, and take the out file and provide it on the output port (scatterplot). This means reading the temporary file that the R put generates using an InputStream and emitting the data on the port. In addition, becauser the R script used a temporary file for staging the output, ideally the R actor will 'know' to substitute a temporary file name instead of the default 'scatterplot' name in the script so that two or more instances of this actor in the same or different workflows don't overwrite each others output graphs.

With this functionality we will be able to import a suite of R actors that do common tasks that scientists can use, such as regression, anova, means by group, etc. These new actors are described in a new set of bugs (see bug #2043).

#7 - 06/27/2005 01:28 PM - Dan Higgins

Closing this bug because the RExpression actor has been created. This actor

allows basic R functionality to be easily inserted into a Kepler workflow and has conversion i/o routines for connecting to Kepler tokens as R objects.

Note that only a subset of all R object types are handled, but these include arrays and dataframes.

#8 - 03/27/2013 02:16 PM - Redmine Admin

Original Bugzilla ID was 1342

Files

ThoughtsOnR.txt	8.84 KB	06/14/2004	Dan Higgins