Generic Kepler Actors – USER MANUAL

(Please contact the author(s) for the usage of the domain specific actors when necessary.**)**

The Kepler actors folder is found under the "Actor Library". Currently, there is a folder for each individual or project effort. Below is a description of the generic actors categorized by the folder names.

The SPA Folder Actors

BrowserUI: Given a file path or URL including a CGI-based form, this actor can be used for injecting user control and input. It can also be used for efficient output of legacy applications anywhere in a workflow via the user's local web browser. The BrowserUI actor uses the default browser in the user's computer.

Figure1. The GUI for the BrowserUI actor

When not configured, as a result of the CGI form execution, the actor just outputs the (name, value) pairs in XML format and as separate arrays.

The actor can be configured using the configuration interface to allow for automatic CGI form generation. The configuration is made through a text file that imply specifies the name and type of the output ports that the user wants to configure the actor for. Please refer to BrowserUIConfigureTest.xml under $KEPLER/workflows/test for more information on the configuration of this actor.

Command Line: Given a command string and optionally one or few of arguments, switches, input and output files, the CommandLine actor generates a command and executes it using the java Runtime class.

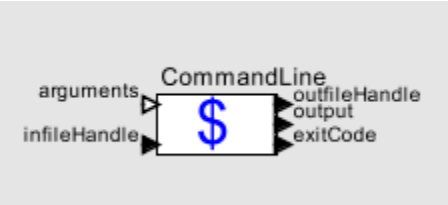

Figure2. The GUI for the CommandLine actor

As illustrated in Figure1, the CommandLine actor has the following ports:

arguments: --Input-- Arguments are independent values or references passed to a function, command or program, by the caller. The arguments can be basic string constants or can be passed into the port from other actors. They can involve the command switches that are attached to the argument values. They are listed in the order they were attached to the port when the command is being generated.

infileHandle: --Input-- This port is used if the file accepts an input file instead of a list of arguments.

trigger: --Input-- This is an optional port that is useful if the execution of a command should be after the successful execution of another actor. The output of the previous actor should be linked to the trigger port of the depending actor for scheduling purposes. It can be hidden if not used by turning off the hasTrigger radio button by selecting the configure box.

outfileHandle: --Output-- An output file can be used to store the results of the execution ibstead of the standard output. The handle is the full-path of the output file that is browsed/given by the output file parameter that can be changed/updated by double-clicking on the actor or selecting "configure" right-click context menu.

output: --Output-- The standard output of the command. Broadcasts only if no outfile is selected.

exitcode: --Output-- A true/false output that shows if the command was executed successfully.

All the ports of the CommandLine actor are optional depending on the usage of the actor.

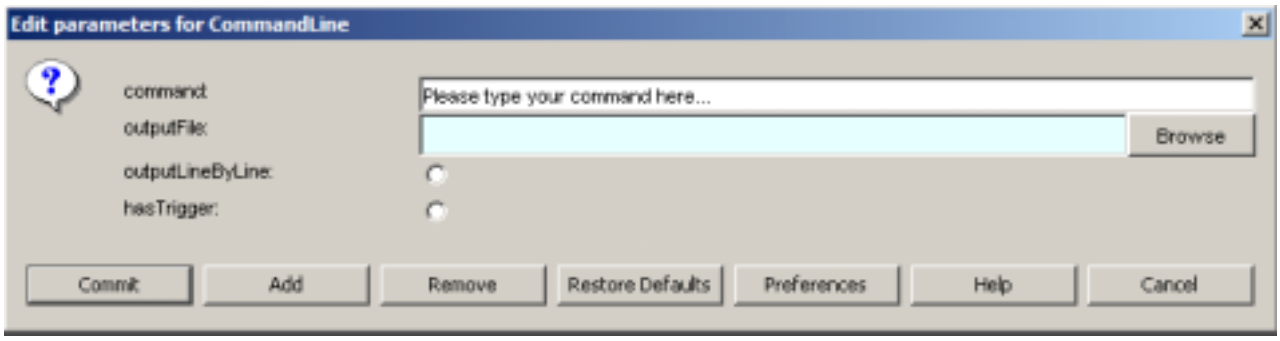

Figure3. The parameters for the CommandLine actor

The parameters of the CommandLine actor (see Figure2) are also used for generating the command as well as formatting the output and available ports.

Command: The main part of the command that all the arguments and infile are attached to. The full path to the command should be specified even if the command is in the PATH environment variable.

OutputFile: Used for browsing/typing the path of the output file that will be used instead of the standard output. The path of the selected file will be output from the 'outfileHandle' output port of the actor.

outputLineByLine: Formats the standard output of the actor if no outfile is selected. When selected, the 'output' output port will broadcast one token for each line in the standard output.

hasTrigger: Activates the 'trigger' input port of the actor when selected.

In the current version of the actor, the following command types are supported:

command (e.g. "C:/Program Files/Internet Explorer/IEXPLORE.EXE")
command < infile > outfile (e.g. "$HOME/myprog < inputFile.in > outputFile.out")
command > outfile (e.g. "C:/cygwin/bin/dir.exe > dirTemp.txt")
command < infile (e.g. "/usr/local/bin/myprog < inputFile.txt")
command [arg1..argn] > outfile (e.g. "C:/cygwin/bin/perl.exe c:/project/kepler/test/workflows/example.pl > c:/project/kepler/test/workflows/example.out";)
command [arg1..argn] (e.g. "/usr/bin/perl $HOME/perl_code/example.pl)

This actor will be extended to allow for the following commands:

command [arg1..argn] < infile > outfile
command1 | command 2 (This type of commands need to be able to give the output of all the commands instead of only the last one. Currently, only the output of the last command is broadcasted. A parameter to switch this on and off will be added.)

Email: Given the configuration parameters for the host SMTP server, to and from addressed, the Email actor sends the data that is linked to its 'messageBody' multi-port (see Figure4) as an output notification email from Kepler.

Figure4. The GUI for the Email actor

An example usage of the Email actor can be found at '$KEPLER/workflows/test/emailTest.xml'.

FileFetcher: Given a Globus authentication certificate and a ';' separated list (see Figure5) of the full paths of files, the FileFetcher actor copies the files to the localhost destination directory specifed by a configuration parameter.(see Figure6)

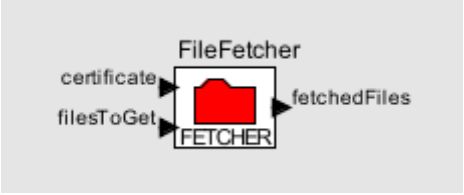

Figure5. The GUI for the FileFetcher actor

The actor outputs a ';' delimited list of full-paths of the fetched files. An example application for this actor can be found at $KEPLER/workflows/test/FileFetcherTest.xml.

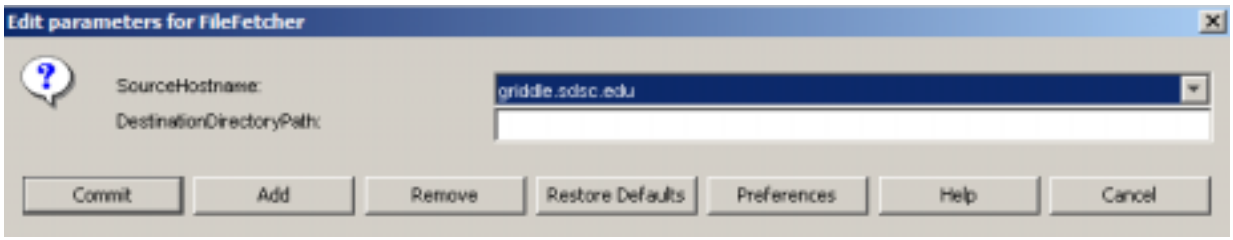

Figure6. The configuration parameters for the FileFetcher

FileStager: Given a Globus authentication certificate and a ';' separated list (see Figure7) of the full paths of local files, the GridFTP-based FileStager actor copies the given files from their localhost paths to a remote destination directory specified by a configuration parameter.(see Figure8)

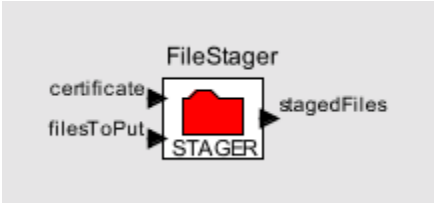

Figure7. The GUI for the FileStager actor

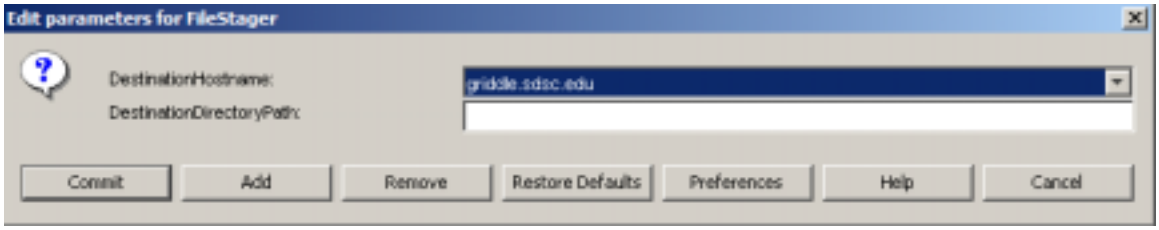

Figure8. The configuration parameters for the FileStager

GridFTP: Given a Globus-grid proxy certificate, and a set of configuration parameters (see Figure9), this actor copies a file from any remote Globus source or local directory, to any remote Globus host or local directory.

The GridFTP actor can be thought of as a combination of the FileStager and the FileFetcher actors without having to copy to/from localhost. The actor has a choice-style selection of available hosts for ease of use. If you would like to set a new source or destination hostname, please click on the 'Preferences' button on the 'Configure' window for the actor and select 'Line' as the style for the parameter you would like to edit/type into.

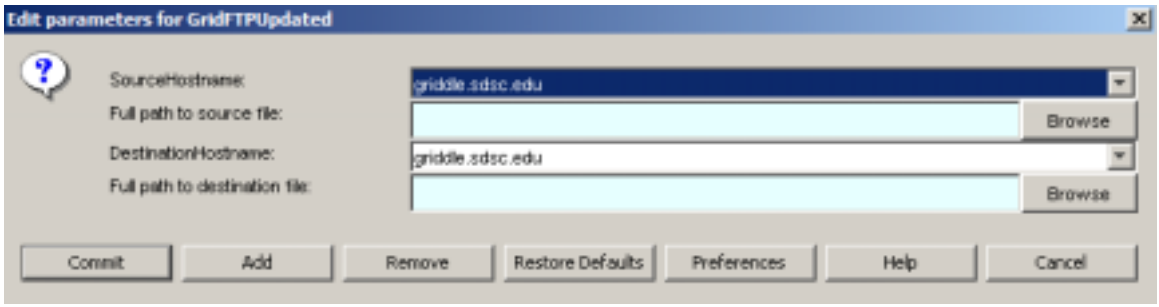

Figure9. The configuration parameters for the GridFTP-based file transfer actor

The GridFTP actor is under a major revision. The documentation for the actor will be finalized after the revision is complete.

Pause: This actor is used for putting an expected pause in the workflow specification to allow for execution to pause until the outputs until that time are reviewed and the workflow is paused. This actor is mainly useful for long-running jobs.

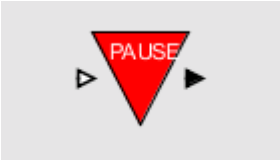

Figure10. The GUI for the Pause

RunJobGridClient: Given a Globus-grid proxy certificate, a list of input files passed from previous actors, and a parameter-specified host and program information, this actor generates the RSL string for a Globus job. It then executes the job and outputs the results and ';' separated list of the output file paths. (see Figures 11&12)

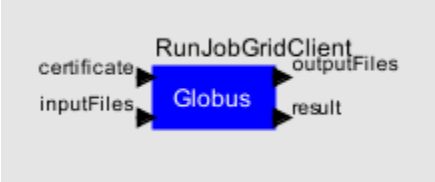

Figure11. The GUI for the RunJobGridClient

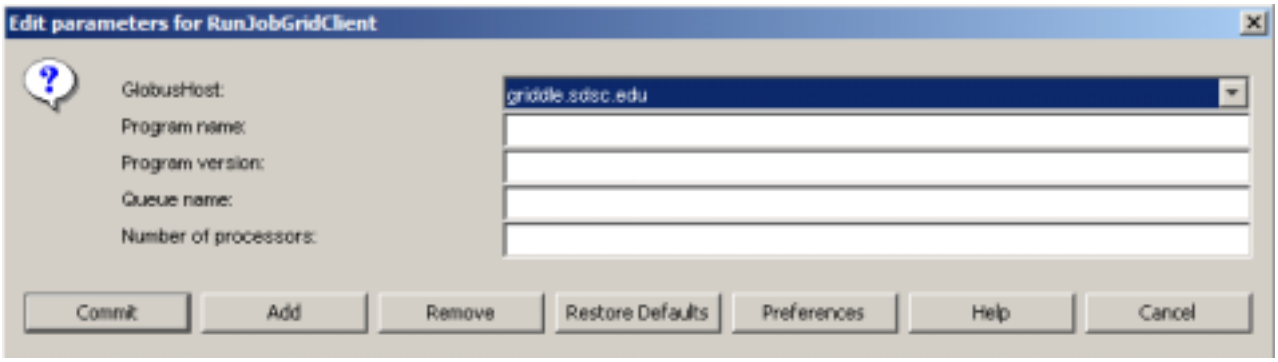

Figure12. The configuration parameters for the RunJobGridClient

StringConst: Given a string or a browsed file path, this actor outputs a string just once. This actor is used mainly in providing constant input to any actor in a workflow.

WebService: The WebService actor, as indicated in Figure13, provides the user with a plug-in interface to execute any WSDL-defined web service. Given a URL for the WSDL of a web service and an operation name that is included in the WSDL, this actor customizes itself to execute this web service operation.

WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints(services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. More information on WSDL and realted standard can be found at: http://www.w3.org/TR/wsdl.

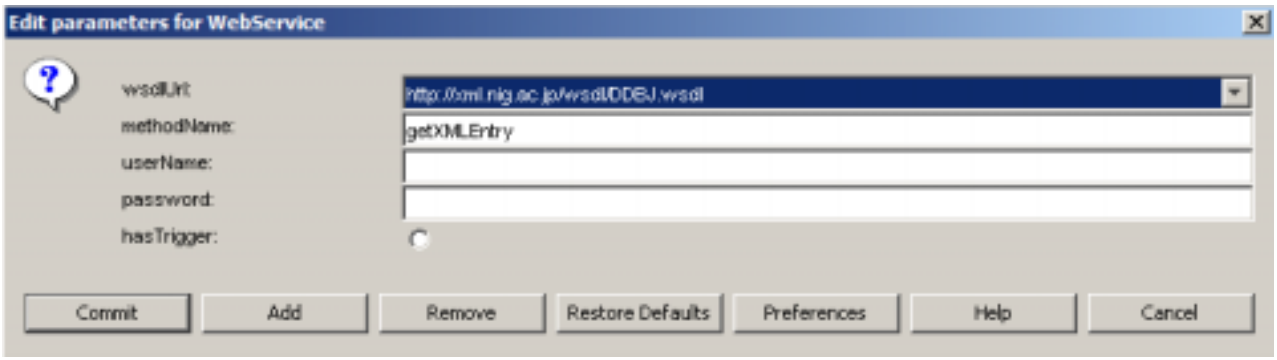

Figure13. The parameters for specialization of the WebService actor

The user can instantiate the generic web service actor by providing the WSDL URL and choosing the desired web service operation. The actor then automatically specializes itself and adds ports with the inputs and outputs as described by the WSDL. The so instantiated actor acts as a proxy for the web service being executed and links to the other actors through its ports.

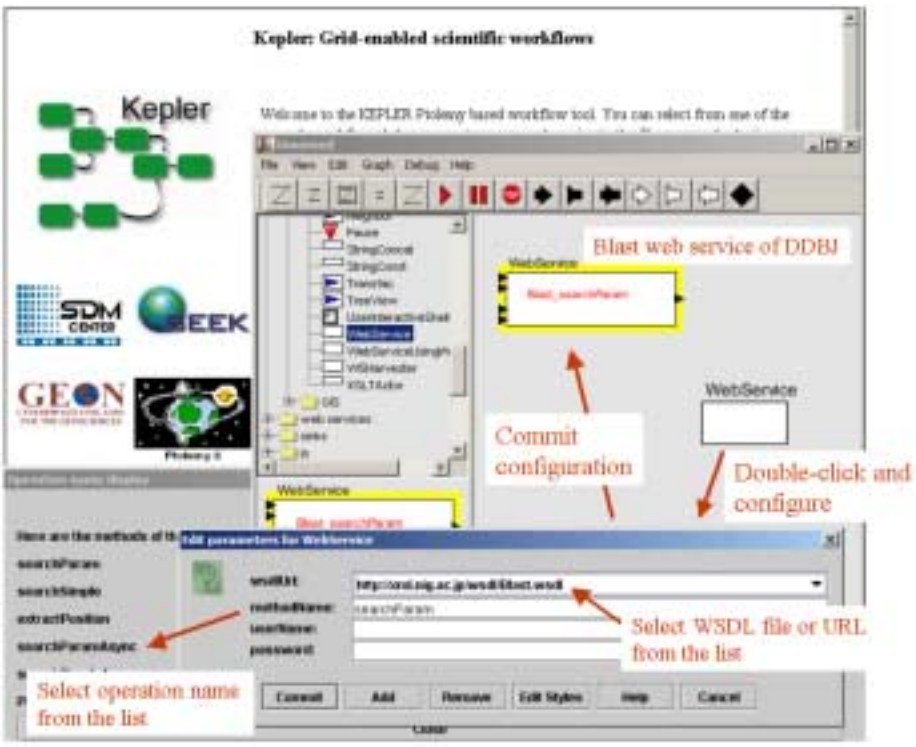

Figure14. An example instantiation of the WebService actor

The WSDL is parsed to get the input, output and binding information. It dynamically generates ports for each input and output of the operation. This customization happens at the configuration time of a model. When the actor is fired at run time, it gets the binding information and creates a call object to run the model. Using this call object, it invokes the web service and broadcasts the response to the output ports. Figure15. below shows two different instantiation of the actor for the Blast and DDBJ web services provided by the DDBJ. Example applications and test can be found under the workflows section of your Kepler directory.

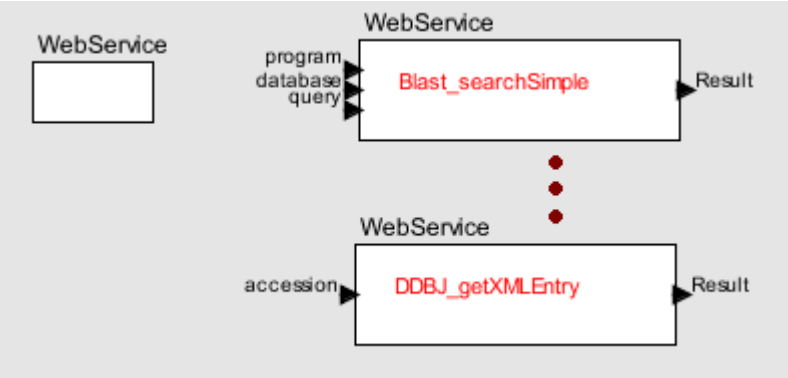

Figure15. The GUI for the WebService actor

WebServiceHarvester: Kepler provides a Web Service Harvester capability for importing web services from a repository. This feature was developed for conveniently plugging in a whole set of (possibly related) services. The web services to import can be searched on a web page or in a UDDI repository. Once imported, the web services are saved as actors. These actors can be reused in different scientific workflows.

The WebService actor is currently under rebuild. More information on the new features and usage of the actor will be given in this manual after the finalization of this actor. Please refer to the 'WebServiceHarvesterTest.xml' under your $KEPLER/workflows/test directory for an example usage of the actor.

XSLTTransformer: Given an xml stream as input, XSLTTransformer is used for for linking "almost but not quite fitting" output port and input port data formats together. The actor produces an html stream that can be viewed or queried using the BrowserUI actor. (see Figure)

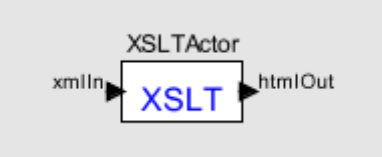

Figure16. The GUI for XSLTTransformer

The configuration window for the XSLTTransformer actor can be viewed by double-clicking on the actor or by selecting 'Configure' from the right-click context menu. The window displays a GUI for browsing the XSL script which will be used to perform the transformation.

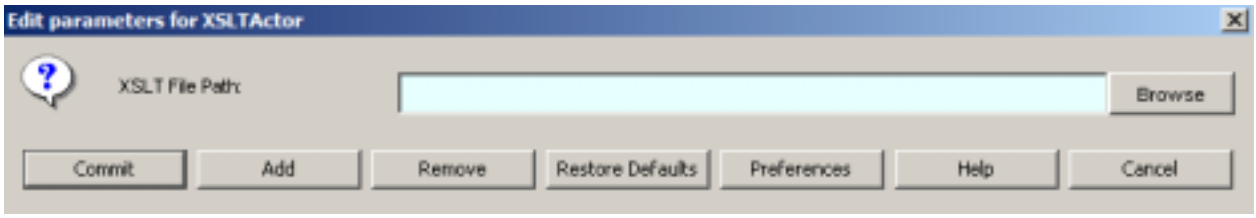

Figure17. The configuration of the XSL path parameter

Scientific Workflow Example

The current web service components of the Kepler/SPA system have been used in various scientific domains, including molecular biology, geosciences, chemistry and ecology.

One example is the "Geological Map Information Integration Workflow" depicted in Figure18. This workflow was designed by a geologist to integrate State Geologic Maps using rock and geologic age ontologies. This model demonstrates the use of distributed processes within a workflow. The details of this workflow can be reached from the SDSIC presentation at http://kbi.sdsc.edu/SciDAC-SDM/SDSIC-Integrated.ppt.

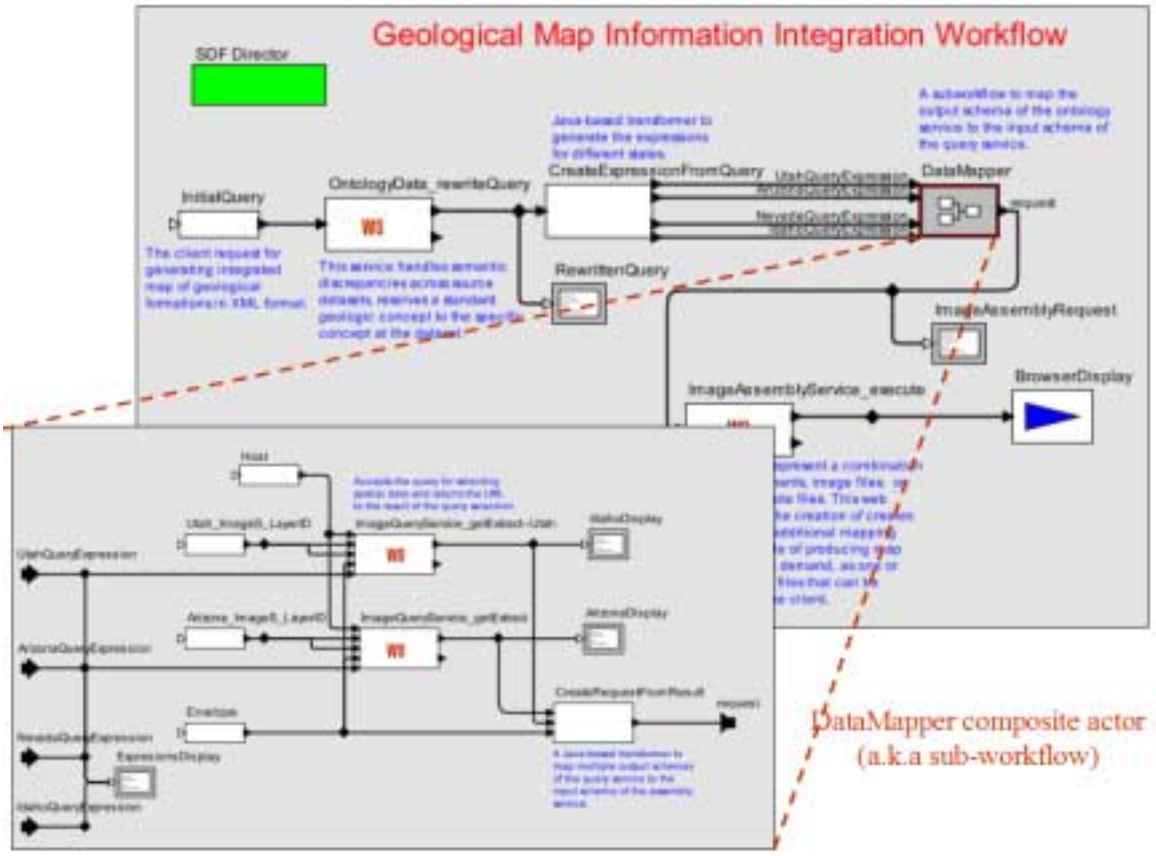

Figure18. Geological Map Information Integration Workflow

The application workflows show how to employ Kepler's web service components to compose distributed scientific workflows. Since web services are often not designed to fit, data transformations between the outputs of previous steps and inputs of subsequent steps are usually required. For this purpose, specialized data transformation actors (e.g. XSLT, XQuery) have been implemented. User interaction and workflow output are performed via a browser actor. Please open to '$KEPLER/workflows/geo/geonMapHierarchical.xml' under Kepler to execute this workflow.

Another example of the Kepler/SPA powered workflows is the 'Promoter Identification Workflow' which had been a running application in different forms for 3 years. It is a production workflow being used by a biologist to identify likely transcription factor binding sites in a series of genes. The process of identifying these sites in a single gene involves a series of tasks, such that performing the same series manually for each of a few dozen genes can be quite a repetitive and time-consuming process. The PIW workflow solves that problem by allowing the biologist to create the workflow once, and run it as many times as he or she desires for any set of different inputs. The details of this workflow can be reached at **http://kbi.sdsc.edu/SciDAC-SDM/piw-specification.ppt**. The workflow can be reached and executed from '$KEPLER/workflows/bio/PIW.xml'.

Another domain that utilized the Kepler/SPA actors is computational chemistry. The goal of this effort is to develop workflow tools specific to computational chemistry. Due to the current

explosion of the corresponding experimental data, there is a huge need for computational models, which combine various scientific methods, system sizes, and time scales on one hand and allow all this to be done in a high-throughput manner on the other hand. However, researchers in different fields often face the same problems: Their calculations need considerable computing power that fortunately nowadays can be provided by supercomputers and/or clusters which, although may need to be accessed remotely or via a grid, making connectivity and bookkeeping pretentious. The application programs applied are typically developed by groups of scientists over many years, and are highly specific and optimized, but difficult to adapt. Each program nearly always has its own proprietary input and output formats, often mixing data and keywords, making communication between different application codes difficult and ineffective.

As a first example of this type of workflows, we have developed the Nimrod/G-based Gamess execution workflow. This workflow utilizes the XSLTTransformer and CommandLine components of Kepler and combines them with some domain-specific knowledge using XSL scripts. (see Figure 19) Please refer to **http://www.sdsc.edu/~altintas/ieee_manish_apr30_04.doc** for more information on this effort.

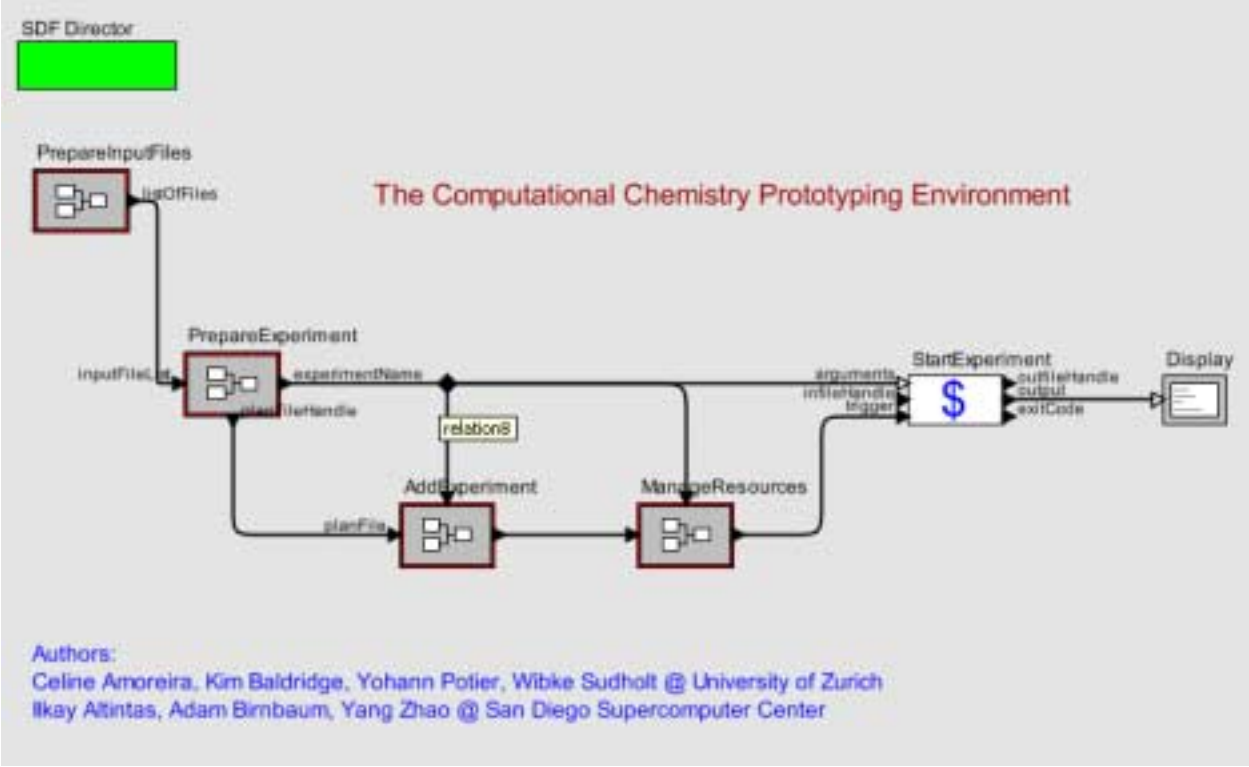

Figure19. Computational Chemistry Prototyping Environment

The generic components of the Kepler project are applicable to all scientific domains as well as non-scientific application domains. Figure20 illustrates an example workflow that utilizes the Globus Grid-based components and their usage.

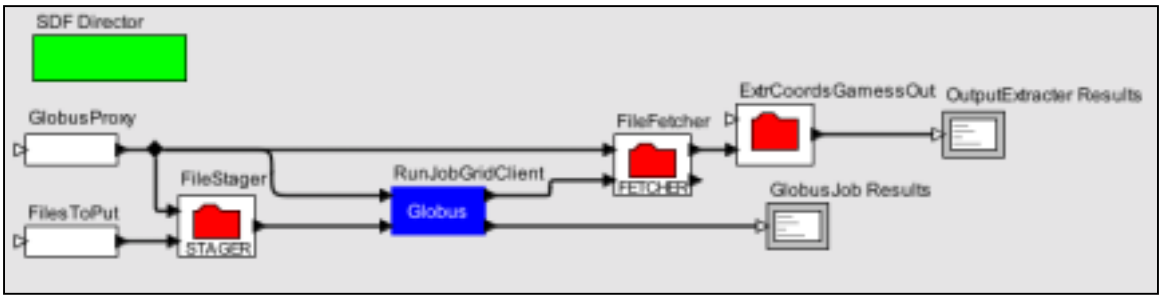

Figure20. An example Grid-based workflow

Please refer to the test workflows under your Kepler directory for interesting test-level applications utilizing these actors.

The GEON Folder Actors

BinaryFileReader: Reads a file or a URL and outputs its content as a sequence of byte arrays. The actor extends the Ptolemy FileReader, and can be used to read both ascii and binary file formats.

Figure21. The GUI for the BinaryFileReader actor

Inputs:
trigger:unknown; a trigger to invoke the actor.
Outputs:
Output:[byte]
endOfFile:Boolean
Parameters:
FileOrURL: FileParameter

BinaryFileWriter: Writes a sequence of byte arrays into a file and eventually outputs the file path. The actor is capable of writing both ascii and binary contents.

Figure22. The GUI for the BinaryFileWriter actor

Inputs:
input: [byte]
Outputs:
filePath:string
Parameters:
FileOrURL: FileParameter

Database Query: The DatabaseQuery actor takes as input a database connection reference, an SQL query, and a result-type parameter, indication of the query result type; XML, record or string and the broadcast rate; whether to output the complete result set as a single token or as a sequence of tokens, each row individually.

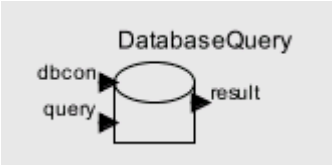

Figure23. The GUI for the DatabaseQuery actor

Inputs:
- dbcon: DBConnectionType
- query: string

Outputs:
- result: XML string / record / string

Parameters:
- OutputType: choice
- outputEachRowSeparately: boolean

OpenDBConnection: A database connection actor. Receives database connection information from the user, either by selecting a connection link from a database driver repository, or by providing the database URL, user-name and password. The actor returns a reference to the database connection (wrapped as a database connection token). The connection can then be propagated to all actors accessing the specified database.

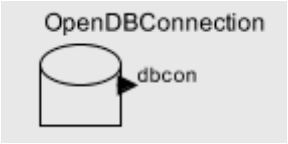

Figure24. The GUI for the DatabaseConnect actor

Outputs:
- dbcon: DBConnectionType

Parameters:
- driverName: string
- databaseURL: string
- username: string
- password: string

SRBConnect: Connects to the SRB and returns a reference to the SRB file system. The user needs to specify the following connection parameters: srbHost, srbPort, srbUserName, srbPasswd, srbHomeCollection, srbMdasDomainHome and srbDefaultResource. The connection reference can then be propagated to all actors accessing the SRB workspace.

Figure25. The GUI for the SRBConnect actor

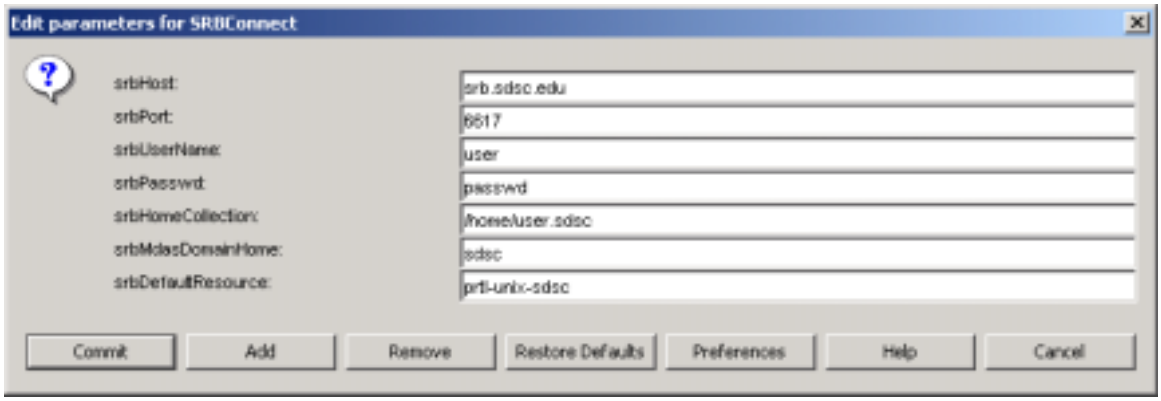

Figure26. SRBConnection parameters

SRBDisconnect: Disconnects from the SRB File system once it gets an confirmation that all actors accessing the specified file system have terminated.

Figure27. The GUI for the SRBConnect actor

SRBReader: Accepts a reference to the SRB files system and a file name, reads the file from the SRB and outputs its content as a sequence of bytes arrays.

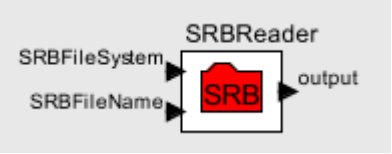

Figure28. The GUI for the SRBReader actor

Inputs:
SRBFileSystem: object
SRBFileName: string
Outputs:
output: [byte]

SRBWriter: Accepts a reference to the SRB files system, an SRB remote file name and a sequence of bytes array as input. The SRBWriter actor writes the byte arrays to the remote file on the SRB and sends a trigger once it is done.

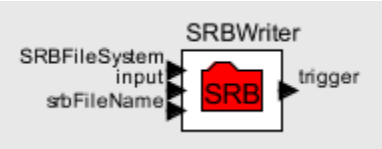

Figure29. The GUI for the SRBWriter actor

Inputs:

SRBFileSystem: object
Input: [byte]
SRBFileName: string

Outputs:

trigger: boolean