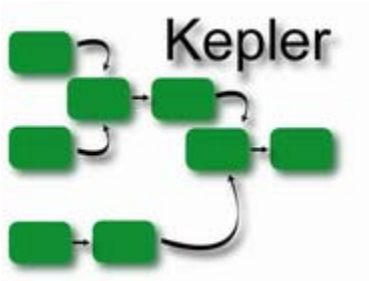
DRAFT

Kepler: An Extensible System for Design and Execution of Scientific Workflows



User Guide

* This document describes the Kepler workflow interface for design and execution of scientific workflows. It is based on Ptolemy II overview documentation, volume I.

TABLE OF CONTENTS

1 Introduction 1
2 Using Kepler. 2
2.1 Exploring and Executing Pre-built Models 3
2.1.1 GEON Mineral Classification Workflow 3
2.1.2 Running a Workflow 4
2.2 Creating a New Model. 6
2.2.1 The Workflow Editor 6
2.2.2 Creating a Workflow 6
2.2.3 Saving a Workflow 8
2.2.4 Actors, Ports and Relations. 8
2.2.4.1 Adding a Relation 8
2.2.4.2 Polymorphic Types 9
2.2.4.3 Adding Ports 9
2.2.5 Composite Actors 10
2.3 Prototyping Workflows. 11
3 Domains. 12
3.1 Process Networks (PN). 12
3.2 Synchronous Dataflow (SDF). 13
4 Actor Libraries. 13
4.1 Data Processing Actors 13
4.2 Display Actors 14
4.3 Various Execution Environments Actors 14
5 More Examples 15
5.1 GEON Map Information Integration Workflow. 15
5.2 Seismic Waveform Modeling Workflow 17
5.3 Database Query Model. 18

1 Introduction

The Kepler project [1] is a cross-project collaboration (see [2] for a list of participating members) to develop a framework for design, execution and deployment of Scientific Workflows. Kepler is built on top of Ptolemy II API for heterogeneous, concurrent modeling and design. Ptolemy II is an ongoing project at UC Berkeley [3].

Scientists in a variety of disciplines (e.g., biology, ecology, geology, astronomy) need flexible means for accessing scientific data and executing complex analysis on the retrieved data. Such analyses can be modeled as 'scientific workflows' in which the flow of data from one analytical step to another is described in a formal workflow language. The Kepler project's overall goal is to produce an open-source, extensible scientific workflow system that allows scientists to design scientific workflows and execute them efficiently using emerging Web and Grid-based technologies to distributed computation.

Ptolemy II is a system for modeling and executing workflows that provides a core infrastructure to Kepler. It provides support for dataflow-oriented models which is a very important characteristic of scientific workflows. The Ptolemy II system provides a convenient graphical user interface for designing and prototyping workflows. This graphical user interface is called “Vergil”. The system also comes with an excellent and comprehensive documentation and can be easily used and extended. The Ptolemy II system supports multiple, precisely specified execution models, called directors. Directors implement the execution semantics and are responsible for the scheduling of the execution of workflows at run-time. Workflows can be nested arbitrarily. The different hierarchical units (a.k.a. sub-workflows) in a workflow can have different directors. The sub-workflows can inherit their execution semantics from the parent, or introduce a new local director.

The focus of Kepler is on actor-oriented design. *Actors* are re-useable components that execute and communicate with other actors in a scientific workflow through input and output channels. Kepler provides a large variety of computational models, inherited from Ptolemy II. Scientific workflows are often dataflow-oriented and thus in most Kepler models either of two computational semantics are used, Process Networks and Synchronous Dataflow. The details of these two directors will be discussed later on.

Kepler inherits Ptolemy II’s Vergil graphical user interface. The Vergil interface is an intuitive tool for designing, prototyping and executing scientific workflows. The interface serves as an editor for Kepler’s workflow modeling language called Modeling Markup Language (MoML). MoML is an XML description of a Kepler workflow. Workflows in XML format can be easily shared with other scientists.

Kepler is an extensible system. Developers can easily add functionalities to the system according to scientists needs. End users can also extend the actor library by using the web service harvesting capabilities to import and execute new system components.

Kepler provides a large variety of scientific tools, such as:

- Access to local and remote data through web and grid services. Kepler provides a generic component that instantiates itself to perform remote calls as if they were running on the local machine.
- Generic tools that may be used in diverse scientific workflow belonging to any scientific discipline, e.g. math actors, conversion actors, etc.
- Various display, user interaction and communication tools such as browsers, graph plotters, email and SMS actors.
- Domain specific tools; actors that were developed to serve a specific field in science domain specific actors,
- A prototyping tool for designing workflows and sharing ideas with other scientists.

2 Using Kepler.

Kepler's Vergil interface allows the user to explore and execute pre-built models or construct new models. The application can be started by either typing $PTII/bin/vergil from the command line or selecting Ptolemy II -> Vergil-full, in the start menu. Once the system is up, you will see the "Main window" in figure 1.



Figure 1. Kepler welcome window

The examples in the welcome window show the cross projects collaboration (see [3]-[5] for participating projects) that constitute of the Kepler project. Kepler can be used for modeling scientific workflows assisting geologists, ecologists, biologists and many other scientists. The welcome window also refers to the Ptolemy II project main window, where you can view more examples under quick tour.

2.1 Exploring and Executing Pre-built Models

Selecting one of the example links will open a model in a Vergil graphical editor and would allow the user to view, edit and run the workflow. Another way to explore a pre-built model is by selecting Open File/Open URL from the File menu and pointing to a MoML (XML) file.

2.1.1 GEON Mineral Classification Workflow

Selecting *GEON Mineral Classifier* will open a GEON workflow for modal classification for naming rocks (figure 2). Mineral classification is being performed in several iterations of finer descriptive levels. At each iteration, a sample point is classified within an appropriate classification diagram using a point-in-polygon algorithm. According to the region of the point in the diagram and the sample mineral abundance, a next level diagram is chosen. The process terminates once the selected region represents a rock name.

The user needs to provide a sample id from a rock dataset along with a link to the classification diagrams dataset. The model then feeds a *Classifier* actor with mineral abundance, retrieved from the rock database, and a set of classification diagrams, and the *Classifier* outputs a rock name.

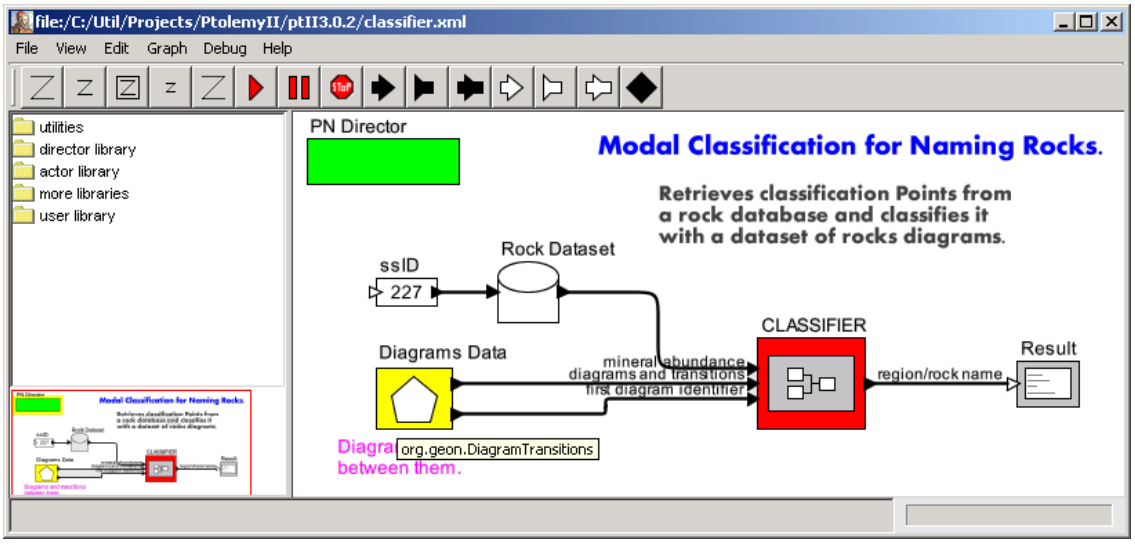


Figure 2. GEON mineral classifier workflow

The *Classifier* is the process that performs the iterative algorithm described above. It is a *composite* actor (an hierarchical component, indicated visually by a red outline), which means that its implementation is itself given as a block diagram. You can look inside the *Classifier* sub-workflow to reveal the implementation by right clicking the actor as shown in figure 3 (Composite actors are discussed in more details later).

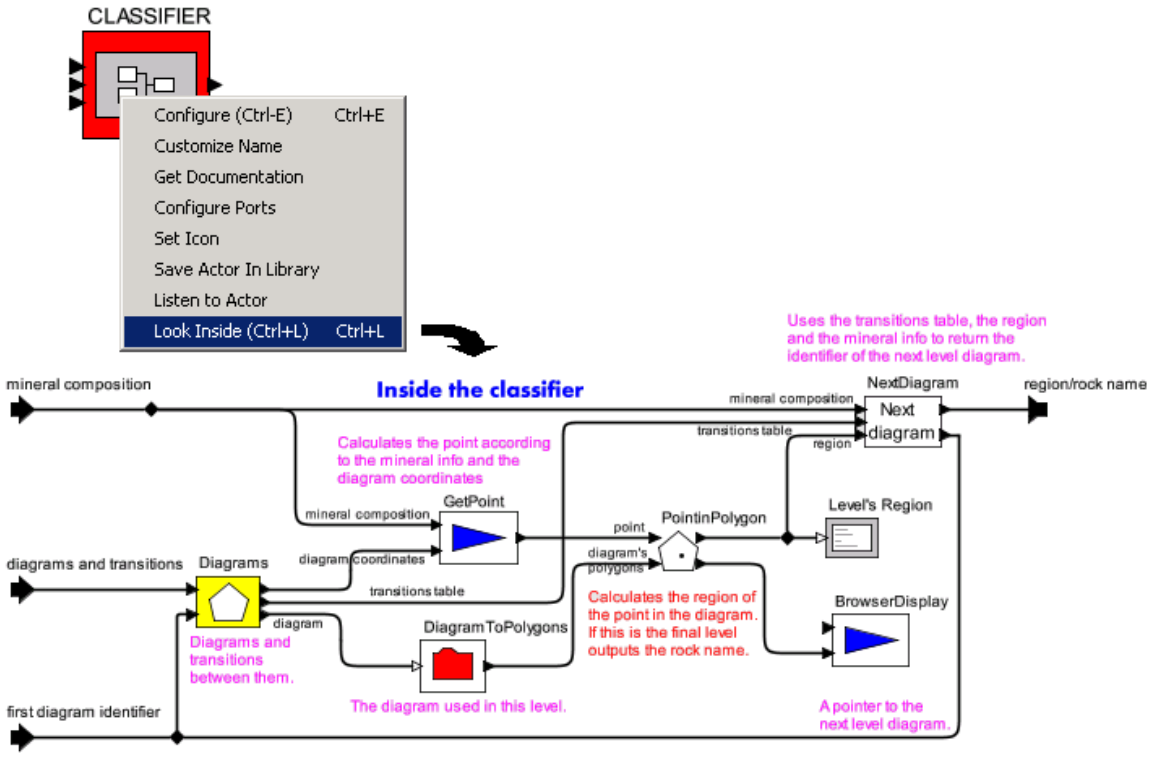


Figure 3. Looking inside the Classifier implementation

Inside the classifier. The *Diagrams* actor holds all the diagrams information (SVGs) and the transitions between them. It receives a diagram identifier and outputs the appropriate diagram to be used in the current iteration. The *DiagramToPolygons* actor converts an SVG diagram into polygon objects. The *GetPoint* actor receives the mineral abundance and the diagram's coordinates and calculates the sample point. Then a *PointInPolygon* actor is used. This actor consumes a classification point and a set of polygons and outputs the point's region along with a *BrowserDisplay* of the point within the diagram. The *NextDiagram* actor is then invoked to calculate the next iteration's diagram.

2.1.2 Running a Workflow

The Rock Classification workflow can be executed in two ways. You can click the run button in the toolbar, or select the "Run Window" in the View menu and then click on Go. The interface also provides other execution control buttons to stop, pause and resume

the execution. The result of the Classifier workflow execution is shown in figure 4. Notice the browser display of each iteration of the classification.

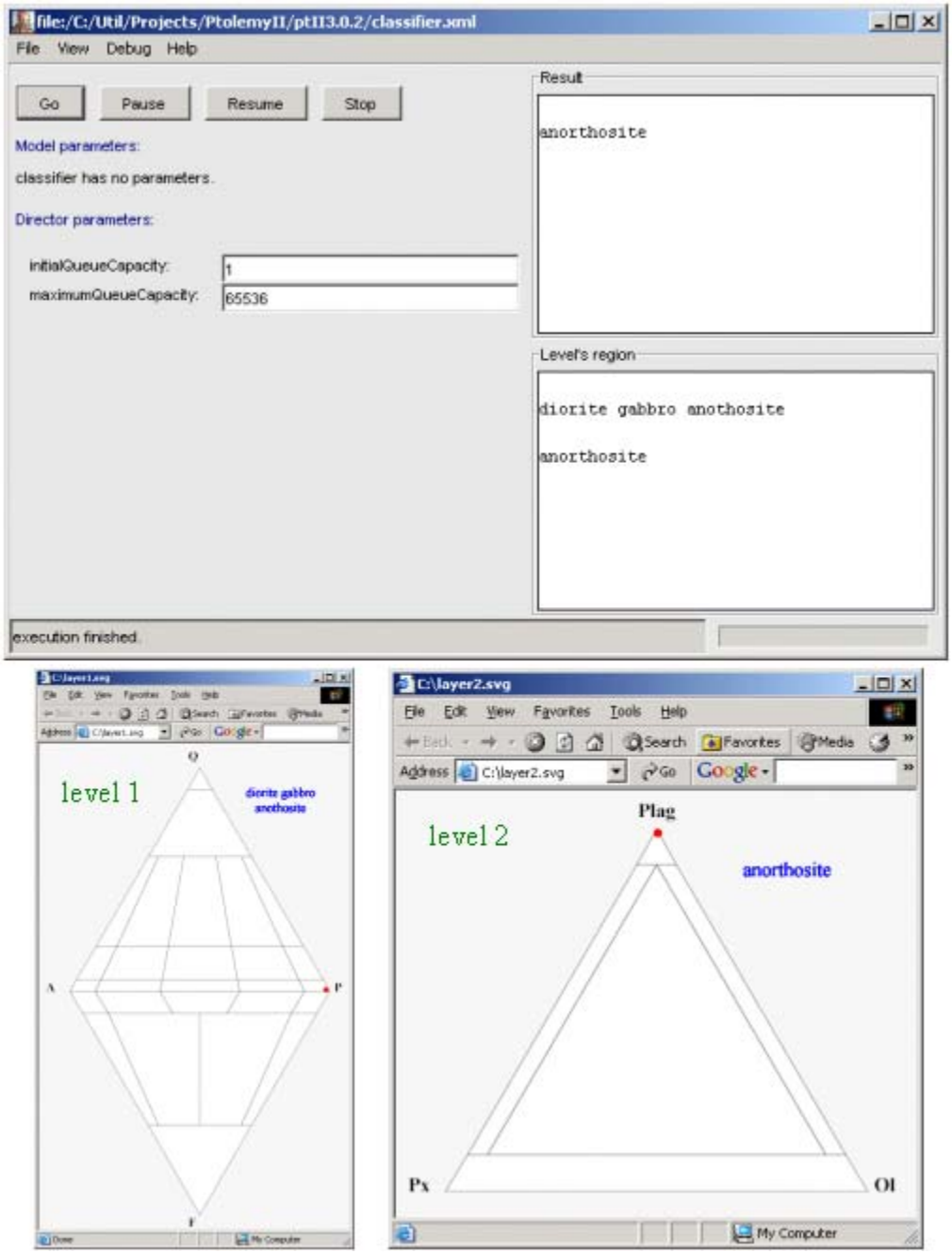


Figure 4. Classification result: The rock name displayed on the Kepler run window and a browser display of the classification steps

2.2 Creating a New Model.

Selecting File->New->Graph Editor in the welcome window, will open a fresh Vergil graph editor. You should see the window shown in figure 5.

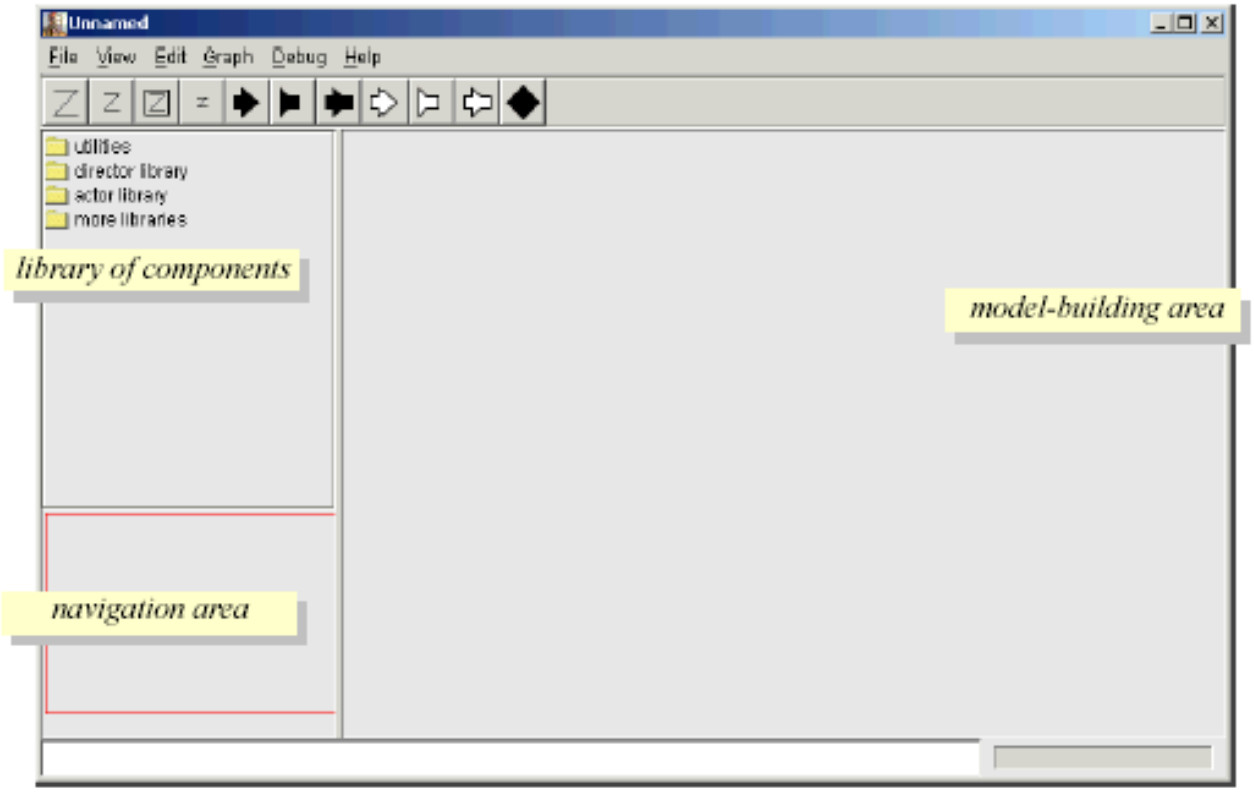


Figure 5. An empty Vergil graph editor

2.2.1 The Workflow Editor

The model-building area canvas in the right is the area where the block diagram is created. On the top left pane there are libraries of objects. Double clicking these libraries expands them and lists their objects. These objects can be dragged onto the model building area for use within a model. To begin with, the model-building area on the right is blank. The bottom left pane provides a smaller image of the whole model and allows the user to zoom and navigate into specific areas of the model-building canvas.

2.2.2 Creating a Workflow

Let's create a workflow for displaying a user expression. Double-click the actor library on the library of components. Select an *Expression* actor from the *math* library, drag-and-drop the actor onto the canvas on the right. Drag a *Display* actor, from the *sinks/generic sinks* library, onto the canvas. Drag a connection from the output port of the *Expression* actor to the input port of the *Display* actor. So far we created a graph of objects. Now let's give a meaning to the graph by assigning a director which provides the semantics of

the flow (directors are further discussed later on). Expand the director library and drag the *SDF Director* onto the canvas. Your workflow should look something like figure 6.

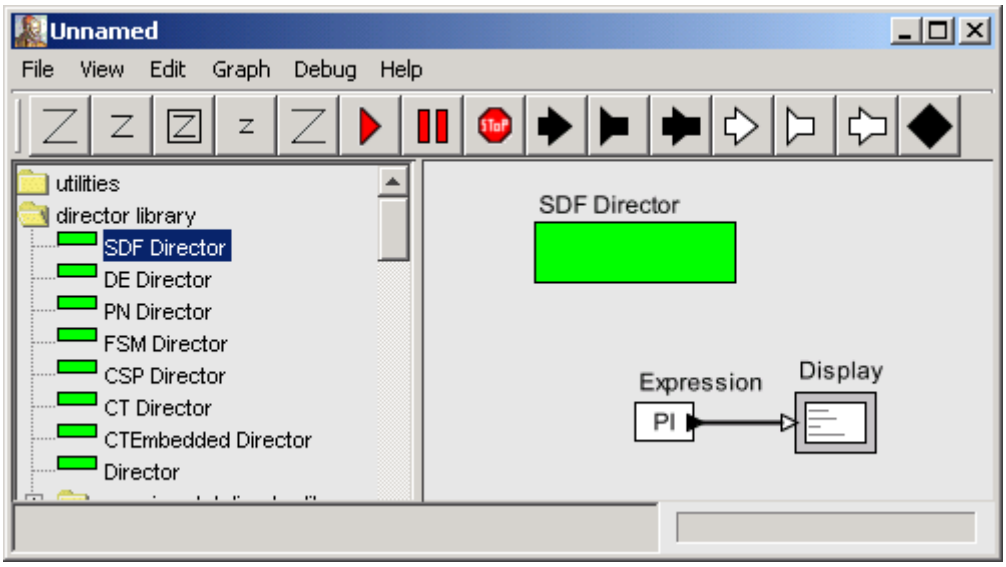


Figure 6. A simple workflow example

Assign the *Expression* actor with an expression by either double clicking the actor icon or right clicking the actor icon and selecting "Configure", as shown in figure 7. In the expression parameter, type an expression, e.g. PI, and commit. The model is now ready for execution. To run the model, either select "Run Window" in the View menu and click on Go, or click on the run button in the toolbar.

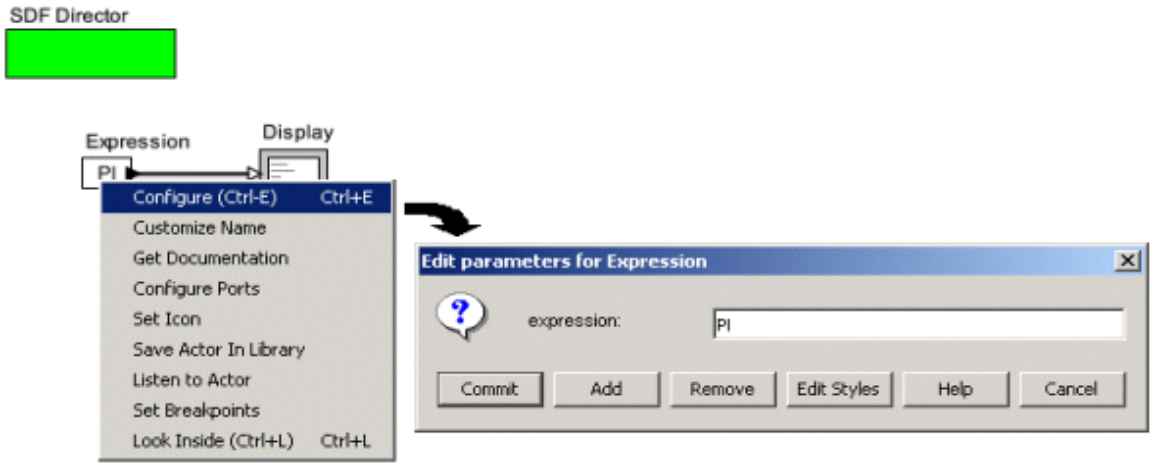


Figure 7. Configuring parameters

2.2.3 Saving a Workflow

You can save the model by selecting “Save” in the File menu. This option generates an XML description of the model called MoML (Model Markup Language). Now the model can be reopened and shared with other scientists.

2.2.4 Actors, Ports and Relations.

Actors are components that execute and communicate with other actors in a model. Actors method of communication is through ports. Values transported through actors ports are encapsulated as *tokens*. A port can be either an input port, an output port or both, and it can either support a single or multiple connections (represented as black and white ports correspondingly). E.g., the *Expression* has a single-port output port; the *Display* has a multi-port input port. Each connection, which is a path from an output port to an input port, is treated as a separate channel. A single port can be linked to at most one relation.

2.2.4.1 Adding a Relation

Suppose we wish to route the output of a single port to more than a single relation. E.g., route the output of the *Ramp* actor to both the *Display* and the *SequencePlotter* actors (as shown in figure 8). If we simply attempt to make the connections, we would get an exception as shown in the figure. Exceptions are normal and common in Kepler. Their purpose is to assist the user constructing a model, by notifying of anomalies. Exceptions in Kepler are very clear and easy to understand and repair. Most common exceptions are: missing a director; trying to run a model without its semantics, type mismatch, e.g. connecting a port of type double to a port of type string, and number mismatch, as observed here, trying to make several connections to a single port without an explicit relation.

In order to direct the output of the *Ramp* to the other two actors, we need to add an explicit *relation* in the diagram. A relation can be used to broadcast an output from a single port. A relation is represented in the diagram by a black diamond and can be selected by clicking the black diamond button in the toolbar.

Select the connection between the *Ramp* and the *Display* and delete it (using the delete key), then connect the *Ramp* output and the *Display* and *SequencePlotter* inputs to the relation. Now the *Ramp* single port still has only one connection to it, a connection to a relation.

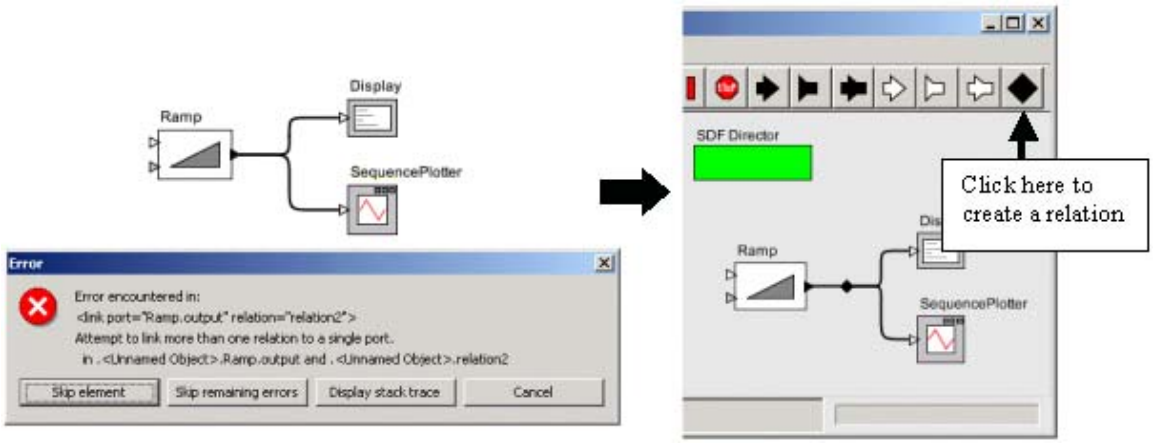


Figure 8. Adding explicit relations

2.2.4.2 Polymorphic Types

Ports can have a type to specify the tokens that can be passed through them. Most actors in Kepler are polymorphic, meaning that they can operate on or produce data with multiple types. In the example of figure 6, the *Expression* can produce several different types (e.g. string, int) and the *Display* can consume several different types. The behavior may even be different for different types. E.g. Multiplying matrices is not the same as multiplying integers, but both are accomplished by the *MultiplyDivide* actor in the math library. Kepler includes a sophisticated type system (inherited from PtolemyII) that allows this to be done efficiently and safely. It also includes a run-time type checking mechanism using a type lattice, which represents allowed data type conversions, to guarantee consistency of linked actors during program execution. In case of a type inconsistentency, an exception will occur.

2.2.4.3 Adding Ports

Ports can also be added or removed to/from actors. Right clicking an actor and selecting "Configure Ports" will bring up the dialog shown in figure 9. This dialog lists all of the actor's ports information. To add a port, click on "Add" and type a port name. Then specify whether it is an input port, output port or both. You can also set the type of the port, although it is not always necessary, the system can infer its type from its context. Ports can later be renamed by right clicking on them and selecting "Customize Name".

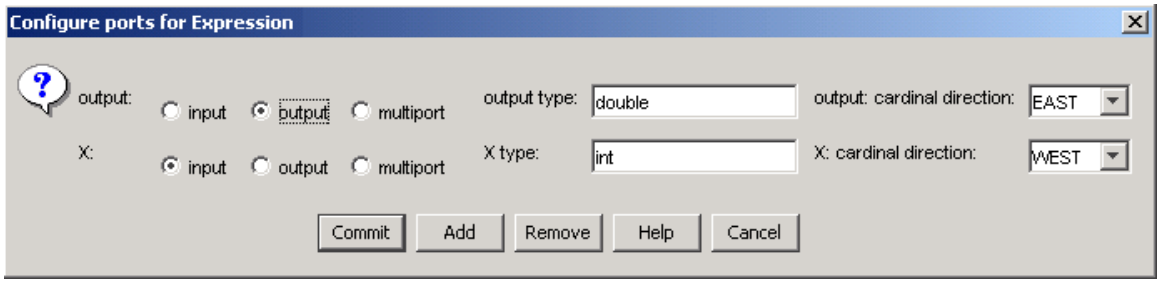


Figure 9. Ports configuration dialog

We modify the example of figure 6 by adding the *Expression* actor an input port, *X*, of type double, and changing the *Expression* parameter from "PI" to "X + PI". We also add an *ExpressionReader* actor (found under *io*) that reads and evaluates expressions from a specified file. Each line read and evaluated by *ExpressionReader* is fed into the *Expression* actor input, which adds a "PI" to it and outputs the result. The modified workflow is shown in figure 10.

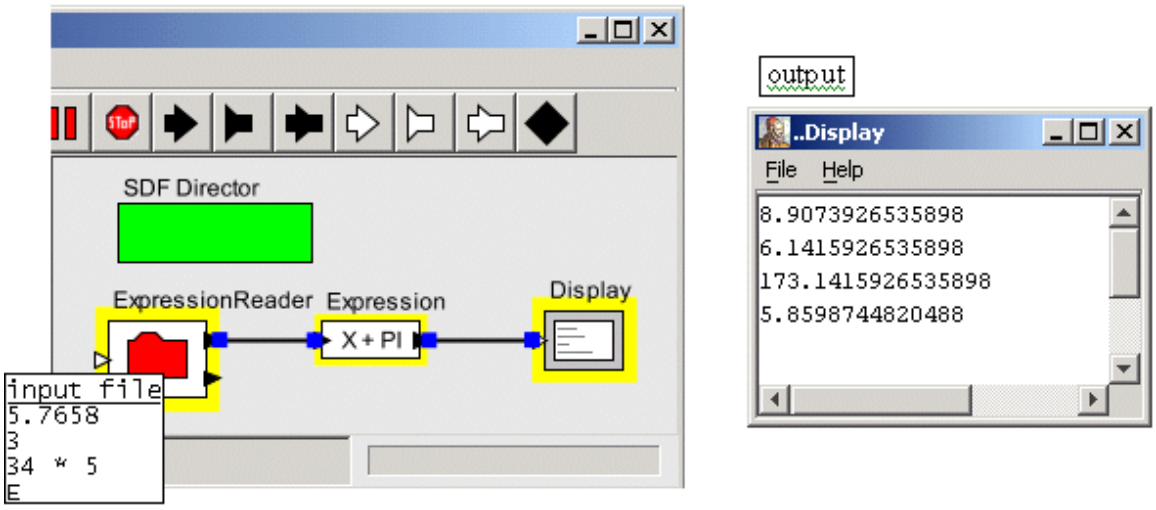


Figure 10. Adding ports

2.2.5 Composite Actors.

Kepler supports hierarchical models. These are models that are nested within other models. Such components are called composite actors. Composite actors are used for various purposes; to improve the visual appearance of a model, to avoid repetitive structures in a model, e.g. if a model contains several similar sub-models (that may differ only in parameters), the sub-model can be packaged as a composite actor and this actor can then be used instead of each sub-model, and to support multiple execution semantics in a single model, recall that Kepler allows nested models to have a different semantic then their parent model, thus using a composite actor this capability can be achieved.

Consider the GEON mineral classification workflow of figure 2. It uses a composite actor called *Classifier*. Without a composite actor, the workflow would look as complicated as the one in figure 11. In the classifier model the user needs only specify a row identifier from a Rock database, by setting the ssID parameter, and the dataset information, and the model outputs the rock name. Thus the user needs not be concerned with the actual implementation of the classifier algorithm in order to run the model and it can be considered as a "black box".

In order to simplify the workflow, we create a composite actor, *Classifier*, grouping all actors within the loop, by highlighting all of them (creating a bounding box with the mouse) and selecting "CreateHierarchy" from the Graph menu. We rename this composite actor *Classifier*, by right clicking on it and selecting "Customize Name". This actor can also be saved to the actor libraries for later use, by right clicking on it and

selecting “Save Actor In Library”. Thus a user, not familiar with the details of the classification algorithm can simply select the *Classifier* actor from the library and use it within a workflow.

Looking inside the *Classifier* will reveal its content in another graph editor. Notice that in this model the *Classifier* sub-workflow shares the same semantics/director as its hosting model. In order to introduce a new local semantic, simply drag a director from the library to the sub-workflow canvas.

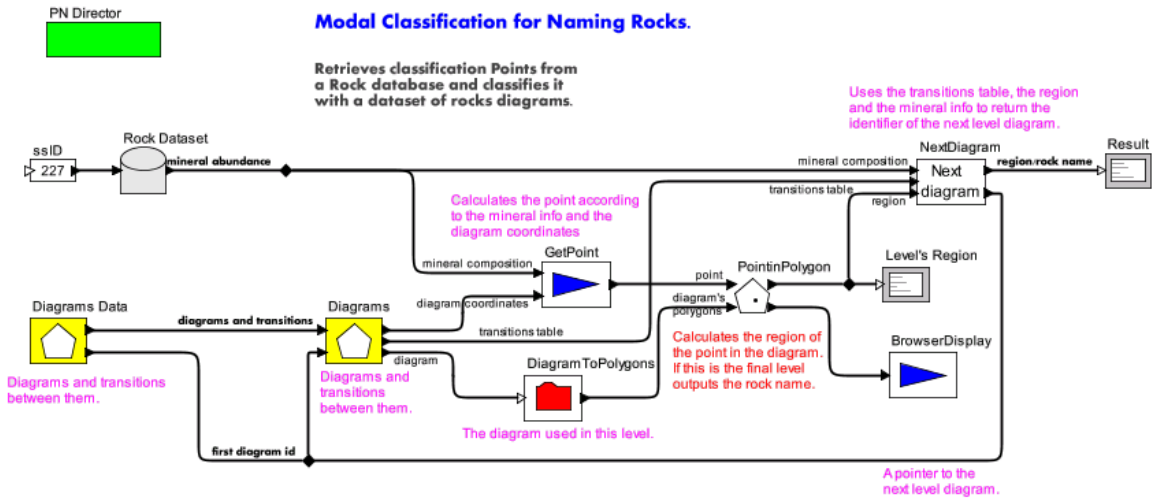


Figure 11a. The GEON mineral classification with no composite actors

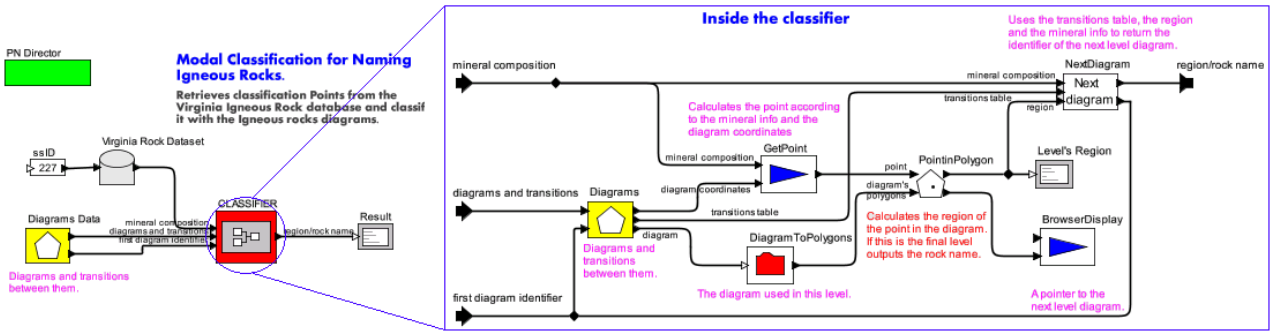


Figure 11b. The GEON mineral classification using a composite actor

2.3 Prototyping Workflows.

Kepler also serves as a convenient tool for designing and prototyping workflows. The system allows scientists to prototype a workflow before implementing the actual actor code needed for the workflow execution, by using a *design actor*. Thus scientists can share and discuss their models with colleagues and later generate the actual application.

Kepler's design actor is created by selecting "New Actor" from the View menu, which brings up the dialog shown in figure 12. The design actor can be seen as a "blank slate" which prompts the scientist for information about an actor, e.g., the actor's name, and the number, names, and types of its input and output ports. Once the user has prototyped an actor, a corresponding stub is added to the user's library. The user can then use this stub on the workflow canvas to prototype a workflow.

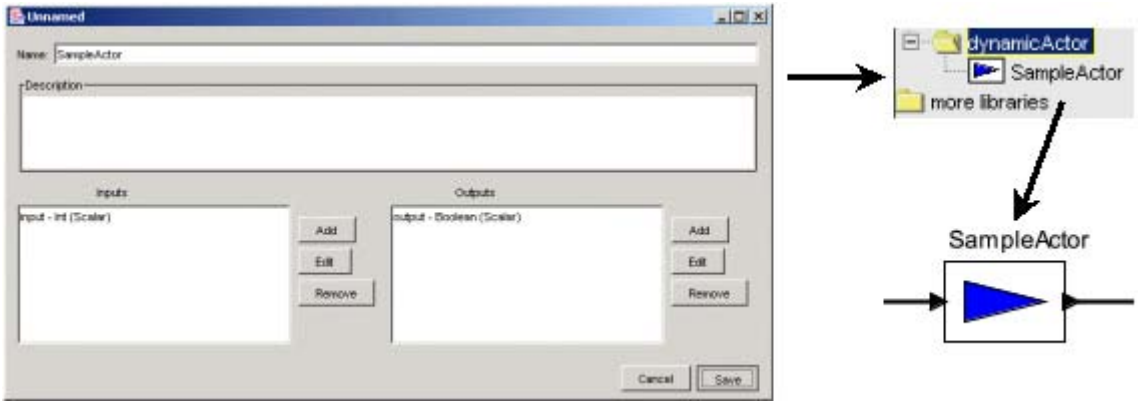


Figure 12. The design actor tool

3 Domains.

A key innovation in Kepler (inherited from Ptolemy II) is that, unlike other design and modeling environments, there are several available *models of computation* that define the meaning of a diagram. A diagram mainly specifies the connections between the model components, whereas a director provides a meaning (semantics) to a diagram. It specifies what a connection means, and how the diagram should be executed. In Kepler terminology, the director realizes a *domain*.

In Kepler most models are dataflow oriented. In a dataflow model, actors are invoked (fired) when their input data is available. Kepler supports two domains for handling dataflow models, Process Networks (PN) and Synchronous Dataflow (SDF). These domains are discussed below. For details on other domains see [7].

3.1 Process Networks (PN).

The process networks (PN) domain models a system as a network of processes that communicate by sending messages through channels that can buffer the messages. The sender of the message need not wait for the receiver to be ready to receive the message; however the receiver is blocked while trying to read from an empty channel until a message becomes available on it.

In the PN domain, the links represent sequences of data values (tokens), buffered as a FIFO queue, and the processes (actors) represent functions that map input sequences into output sequences. Multiple parallel processes can execute simultaneously. The sequences

of values are completely determined by the model and thus this model defines a determinate computation.

3.2 Synchronous Dataflow (SDF).

The synchronous dataflow (SDF) domain handles regular computations that operate on streams. SDF is a special case of process networks where the order of invocation of the actors (execution order) can be determined statically from the model; it does not depend on the data that is processed, i.e., the tokens that are passed between actors.

4 Actor Libraries.

Kepler contains libraries of reusable components, called actors, which are connected to other actors to construct a model. Actors can act as data sources, sinks (various displays), data transformers, analytical steps (e.g., Matlab scripts), or more generally any computation step which can be invoked as a (web) service. Originally aimed at signal processing, PtolemyII provides Kepler with a lot of *math* and *filtering* functionalities. In this section we give an overview of the actors provided in the actor libraries. We focus on the Kepler extensions to PtolemyII. More information is provided in [8].

4.1 Data Processing Actors

The first set of actors discussed below includes data processing actors. Kepler provides various actors for data processing tasks, such as database access and querying actors, file processing actors, and data transformation actors, transforming data from one format to another. More data processing actors are found in [8].

DBConnect. A database connection actor. Receives database connection information from the user, either by selecting a connection link from a DB 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 later be used by each actor accessing the specified database.

DBQuery. A database query actor. Takes as input a database connection reference, an SQL query, and a result-type parameter, indication whether the query result type; XML, record or string and whether to output the whole result set as a single token or as a sequence of token, each row individually. In the future we plan to add GUI-enable actors that support query formulation in a visual query-by-example style.

XSLT. An XSL Transformation actor. Takes as input an XML stream and a matching XSL file URL and transforms the XML content to HTML.

FileToString. A file processing actor. Takes as input a file URL and returns its content as a string token.

FileToArray. Another file processing actor. Returns a file content as an array.

SVGToPolygon. An actor that takes as input an SVG file URL, representing a diagram, and generates Polygon object out of the diagram components.

4.2 Display Actors

The next set of actors includes visualization and user interaction actors to provide a convenient interface for user control and input as well as output. Kepler also provides through PtolemyII other visual capabilities such as a text display and plotters, more information on these components is found in [8].

BrowserDisplay. An actor that takes as input a file or a URL and displays it on a web browser.

BrowserUI. Extends the BrowserDisplay actor. This actor supports a browser interface not only as a display method but also as a mean for convenient user interaction, for easily entering inputs and selections. The actor accepts a file, a URL or an HTML content and either displays the file (as in BrowserDisplay) or returns user selections in XML format.

UserInteractiveShell. An actor that provides a text interface for passing user inputs to the following component in the flow.

4.3 Various Execution Environments Actors

The following set of actors presents the several different execution environments provided by Kepler. Actors in Kepler can either execute locally (as seen so far), or on distributed environments, using web and grid services. Web and grid represent a “black box” functionality that can be reused without worrying about the service implementation.

Kepler also provides (through PtolemyII) support for native language interfaces such as a Matlab actor and a Python actor. All of these actors behave as if they were atomic steps in the workflow execution. Furthermore, any java application can be easily deployed as a new system component.

WebService. A generic actor that can be instantiated with any WSDL description of a web service. The user needs to provide the WSDL URL and select a desired operation, and the actor specializes itself to act as a proxy for the web service being executed. Using the *WebService* actor, any code that can be deployed as a web service, not necessarily a java code, can be used as an actor in Kepler. Kepler also provides a ***Web Service Harvester*** capability for importing web services found in a repository.

GridService. Similar to a web service, specialized for executing remote services on the grid.

GridFTP. An actor for performing grid-based data access.

ProxyInit. An actor for certificate-based authentication.

GlobusGridJob. An actor for grid job submission.

5 More Examples

5.1 GEON Map Information Integration Workflow.

The GEON map information integration workflow (figures 13 and 14) was designed for integrating State Geologic Maps using rock and geologic age ontologies. This model specifically demonstrates the use of web services within a workflow for using distributed processes without possessing the actual implementation.

The model uses the datasets from states covering rocky mountain region as a test bed. These datasets are very heterogeneous; Different representation formats: BLOB, Shape file, GML. Structural differences: different schemas table names, field names, etc. Semantics differences: different conventions as the Canadian GS report and the British report and use different operating systems. To integrate these heterogeneous dataset an ontology service is being used.

The user needs to provide the datasets to be integrated along with the ontologies to be used as the initial query. The query is sent to the ontology service, which is a *WebService* actor instantiated as an ontology data method. The ontology service resolves the datasets differences and returns a color schema for presenting the datasets. Then an image query service is used for accessing datasets from different systems. Eventually an image assembly service is invoked to integrate the results and expose them on a web browser using the *BrowserDisplay* actor (figure 15).

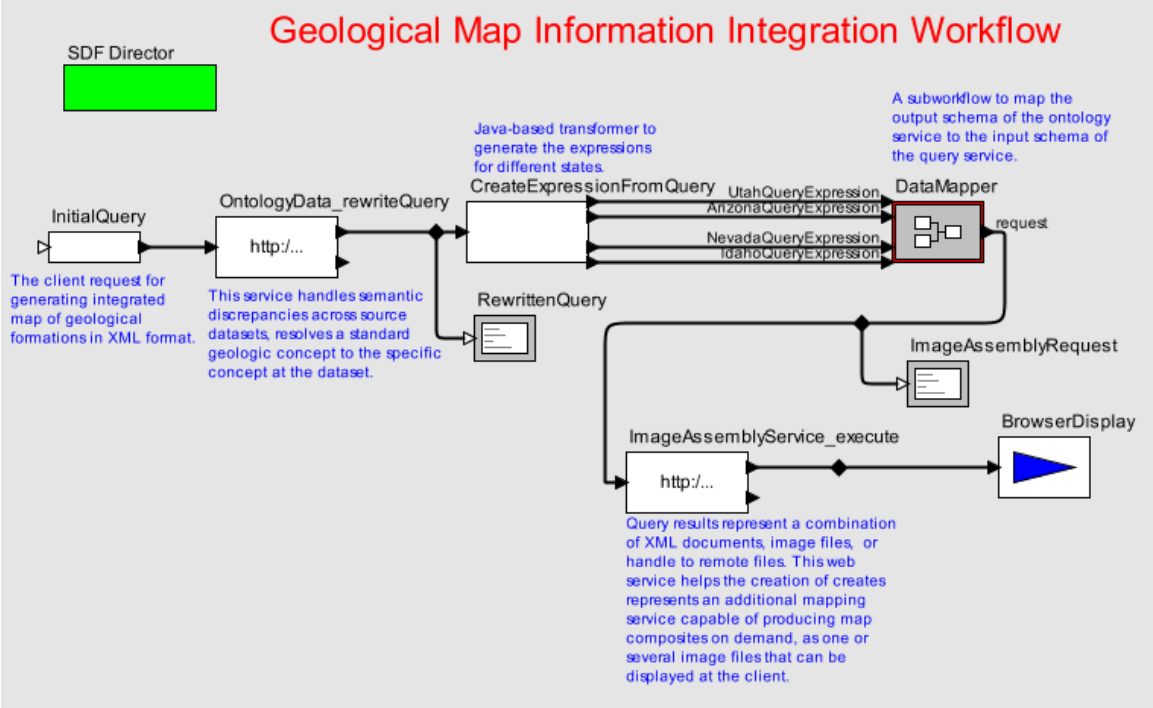


Figure 13. Geologic map information integration workflow

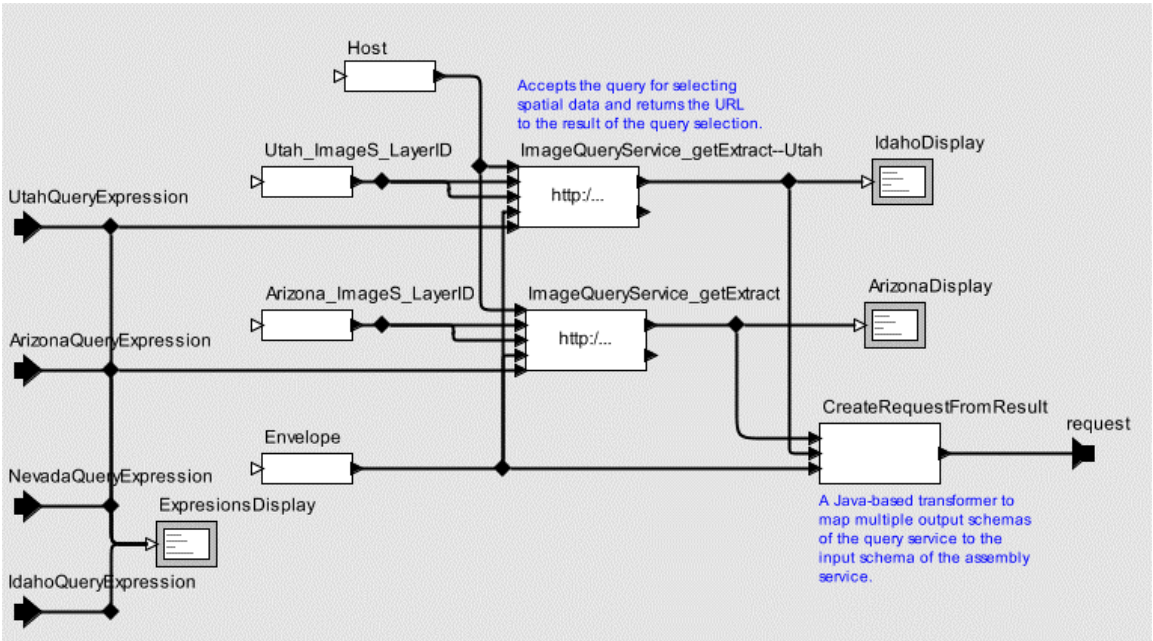


Figure 14. Inside the data mapper

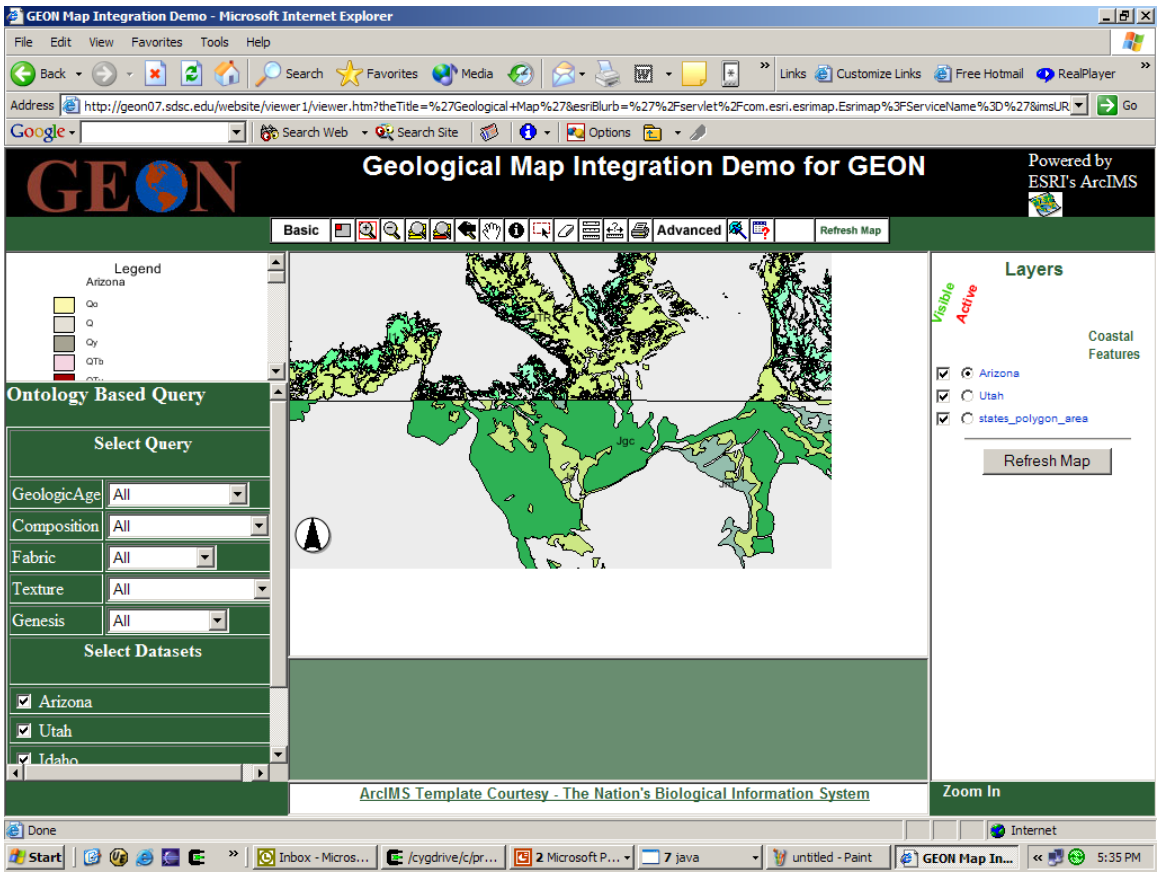


Figure 15. Map integration result using a BrowserDisplay actor

5.2 Seismic Waveform Modeling Workflow

The following model (figure 16) shows an initiative in deployment of a GEON Seismic analysis tool (GSAT). The workflow has not yet been implemented but uses the prototyping tool, discussed above, as its design method.

The goal of the seismic waveform modeling workflow is to analyze and simulate any observed regional seismic waveform recorded from any given region on Earth. Using existing 3D earth models, observed synthetic seismograms (simulations running on distributed supercomputers) would be generated for different earthquake parameters such as earthquake depth and focal parameters. These synthetic records are compared with observed ones to obtain better knowledge about the Earth structure as well as the earthquake parameters.

InvokeApplication, loads the system. The user selects a map and a desired region envelope using *RenderMap* and reads the events and stations within the envelope. The user can then either select to immediately plot an observed waveform, or to analyze a synthetic seismogram by running a simulation over a bounding box. Each of the user selection is captured by a *UserEvents* actor which processes the requests and directs the flow accordingly. Both the observed and analyzed seismograms are plotted with a *SeismicDisplay* actor allowing the user to compare between them.

The synthetic analysis is done as follows; first the user needs to set the simulation parameters. Kepler provides a convenient browser interface for entering user selections, using the *BrowserUI* actor. The simulation is running on distributed machines using a *Grid* actor. The resulted seismogram can then be filtered, using the various signal processing filtering techniques provided by Kepler, and then sent to the display.

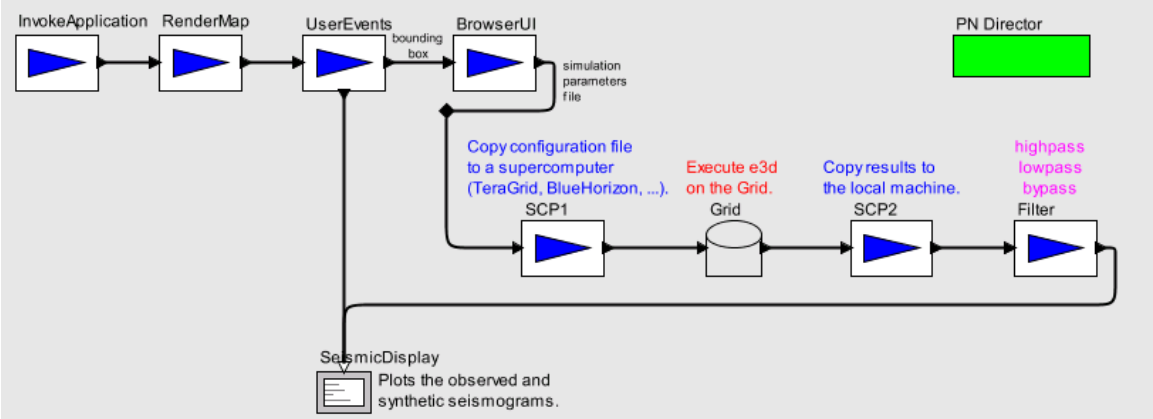


Figure 16. GEON Seismic waveform modeling workflow

5.3 Database Query Model.

The following model (shown in figure 17) demonstrates some of Kepler data processing capabilities. It queries a database and outputs the result on a web browser. The *DBConnect* actor (OpenDBConnection) connects to an Igneous rocks database, as specified by the parameter dialog shown in figure 18. The query is represented by a *Const* actor (found under *Source/Generic Sources*), which is an actor that produces a constant output set by its parameter value. The *DatabaseQuery* actor is fed with both database connection reference and the SQL query string. It outputs an XML stream of the result set, as configured by the dialog box of figure 19. Then an *XSLT* actor is used to transform the query result to HTML format to be displayed using the *BrowserUI* actor (the result is shown in figure 20).

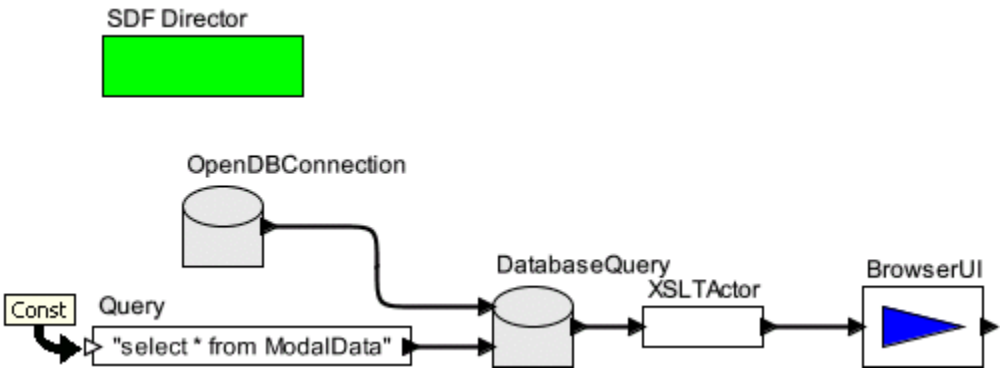


Figure 17. A model for display database query results on a web browser

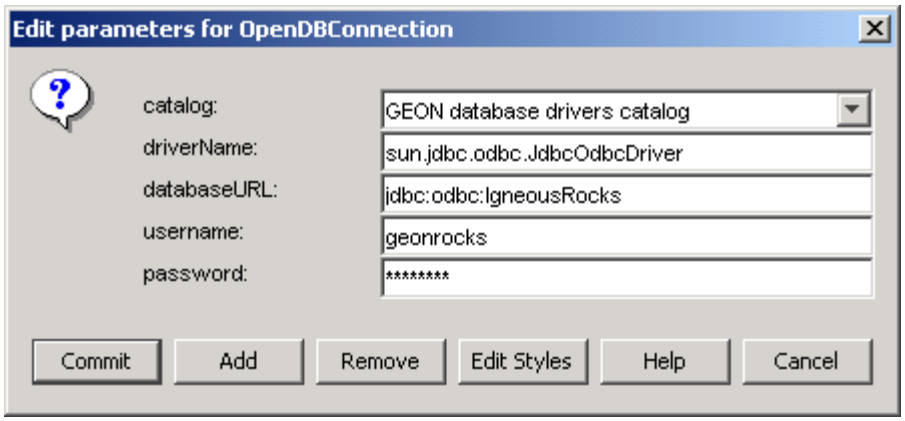


Figure 18. Configuration parameters for the DBConnet actor

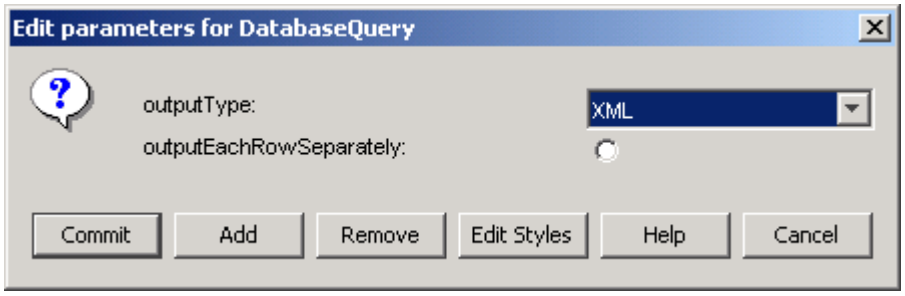
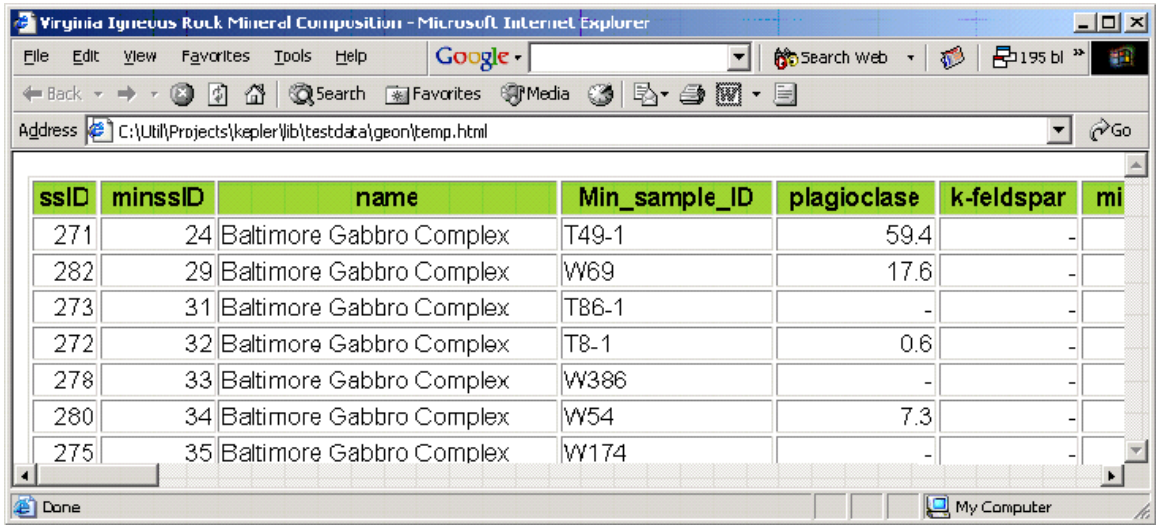


Figure 19. Configuration parameters for the DBQuery actor



ssID	minssID	name	Min_sample_ID	plagioclase	k-feldspar	mi
271	24	Baltimore Gabbro Complex	T49-1	59.4	-	
282	29	Baltimore Gabbro Complex	W69	17.6	-	
273	31	Baltimore Gabbro Complex	T86-1	-	-	
272	32	Baltimore Gabbro Complex	T8-1	0.6	-	
278	33	Baltimore Gabbro Complex	W386	-	-	
280	34	Baltimore Gabbro Complex	W54	7.3	-	
275	35	Baltimore Gabbro Complex	W174	-	-	

Figure 20. Database query result transformed into HTML using an XSLT actor

REFERENCES

[1] Kepler: An Extensible System for Scientific Workflows, http://kepler.ecoinformatics.org

[2] Kepler team members, http://kepler.ecoinformatics.org/members.html

[3] Ptolemy II project, http://ptolemy.eecs.berkeley.edu/ptolemyII

[4] GEON: Cyberinfrastructure for the Geosciences, http://www.geongrid.org

[5] SPA: http://kepler.ecoinformatics.org/spa.html

[6] SEEK: Science Environment for Ecological Knowledge, http://seek.ecoinformatics.org

[7] Ptolemy II domains, http://ptolemy.eecs.berkeley.edu/papers/03/ptIIDesignDomains/ptIIdesign3-domains.pdf

[8] Introduction to Ptolemy II, chapter 4, http://ptolemy.eecs.berkeley.edu/papers/03/ptIIDesignIntro/ptIIdesign1-intro.pdf