Cyber-Integrator Manual

Cyber-Integrator (CI) was created at the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign. We would like to acknowledge multiple funding agencies for the support including NCSA, NSF, NASA and NARA. The main creators of Cyber-Integrator are Rob Kooper, Luigi Marini and Peter Bajcsy with support from Barbara Minsker, Jim Myers, and Tim Nee. This document represents a current description of our on-going research and development of Cyber-Integrator and hence is updated on a regular basis.

Version 1.0

Revision History

Revision	Date	Author	Notes
0.1	4/04/2006	PB, RK,LM	Initial version of the document
1.0	2/8/2007	PB, TN	Initial release of the software

Table of Contents

Chapter 1 Introduction

1.1 Purpose of Cyber-Integrator

The motivation for developing Cyber-Integrator is to design a highly interactive scientific process management (workflow) system that aims at building complex problem-solving environments from heterogeneous tools. Driven by systems-science use cases and complex informatics problems, we identify the dimensions along which current process management technologies must grow to become a robust cyber-infrastructure capable of scaling to meet the national needs. There is an obvious need to be able to join workflows developed using modules from the multiple open source and commercial workflow systems in use in various sub-disciplines. Less obvious but also critically important are abilities to describe and share workflow fragments, to execute portions of workflows on different appropriate hosts, or to provide security, provenance and fault-tolerance features of software execution.

1.2 Why Meta-Workflow?

There are multiple definitions of workflow as described in our white paper[1]. One of the simplest definitions of a workflow is presented below:

A Workflow is a collection of Steps and data that define the paths that can be taken to complete a task. Workflows may contain activities such as displaying content to users, collecting information from users or computer systems, performing calculations, and sending messages to external computer systems[2].

In our current Cyber-Integrator prototype, we use the word meta for the following reason. Our aim is to design a workflow that works with descriptions (meta-data) of data, software tools and computational resources for easy integration and hierarchical organization. Thus, we focus on integration of heterogeneous software by a domain scientist (a computer programming novice) that would be achieved by adding an xml description of his/her tool rather than by a low level programming exercise. One can create distributed meta-data(a "registry") about data/tools/resources with a text or xml editor. The meta-workflow system will pull in all registry information and execute the workflow accordingly. The benefits of such meta-workflow for domain scientists are (a) the simplicity of software integration within the meta-workflow system and (b) the extra benefits of running, reusing, re-purposing and sharing workflows while receiving feedback from the system during workflow creation.

Thus, we labeled Cyber-Integrator as a meta-workflow system to refer to a workflow design that provides meta-data descriptions of data, software tools and computational resources for

---

[1] Bajcsy P., R. Kooper, L. Marini, B. Minsker and J. Myers, "A Meta-Workflow Cyber-infrastructure System Designed for Environmental Observatories," Technical Report: NCSA Cyber-environments Division, ISDA01-2005, December 30, 2005.
(available from http://isda.ncsa.uiuc.edu/peter/publications/techreports/2005/meta-workflow-approaches.pdf)

[2] www.scbos.com/Info/SCBOS-Site-Glossary.htm

easy integration and hierarchical organization, and meets the end-to-end needs outlined in Section 1.1.

1.3 Cyber-Integrator Functionality

The current implementation of Cyber-Integrator enables users (1) to browse registries of data, software tools and computational resources, (2) to create meta-workflows by example (step by step execution), (3) to re-use and re-purpose meta-workflows, (4) to execute meta-workflows locally or remotely, (5) to incorporate heterogeneous code executors and tools, and link them transparently, (6) to provide recommendations about workflow completion, (7) to search for data, tools and resources in registries, and (8) to support processing of streaming data and large size, out-of-core, data.

In this document, **executors** are understood as workflow environments (e.g., D2K or Kepler) or stand-alone application suites (e.g., Matlab, MS Excel or ArcGIS) while **tools** are specific solutions (functions) running using these executors. **Computational resources** refer to those machines that have installations of the Cyber-Integrator executors.

The current support of heterogeneous engines includes a Java executor, an external executor, MS Excel, a Monte Carlo simulation (Copano Bay demo), and GeoLearn. Java includes a set of Im2Learn tools. Both Java and external are generic executors that can run Java-based codes and the command-line based codes, respectively. The tools that use these executors are primarily for remote sensing analyses and water quality analyses (see Chapter 4 for Application Examples).

1.4 Functionality of the Released Stand-Alone Version

This release of Cyber-Integrator is intended as a desktop solution. As a result, remote functionality has been excluded from this release. This manual may mention these abilities. They have been developed in other extended versions of Cyber-Integrator, but this release only handles local functionality. *The application does not at any point contact remote resources*. The extended versions of Cyber-Integrator use the remote resources (a) for executing and monitoring processes on a remote server, (b) for storing and retrieving workflows and provenance information from a remote server, (c) for analyzing streaming data coming from remote locations, or (d) connecting to any remote services, i.e., recommendation services like CI-KNOW. See Appendix A for more information about remote functionality in extended versions of Cyber-Integrator.

1.5 Cyber-Integrator Architecture

We view Cyber-Integrator as a process management system, where a scientist would like to select a triplet with input data, tool and computational resource for each processing step within a sequence. One step of a workflow (understood as a sequence of steps) is shown in Figure 1-1.

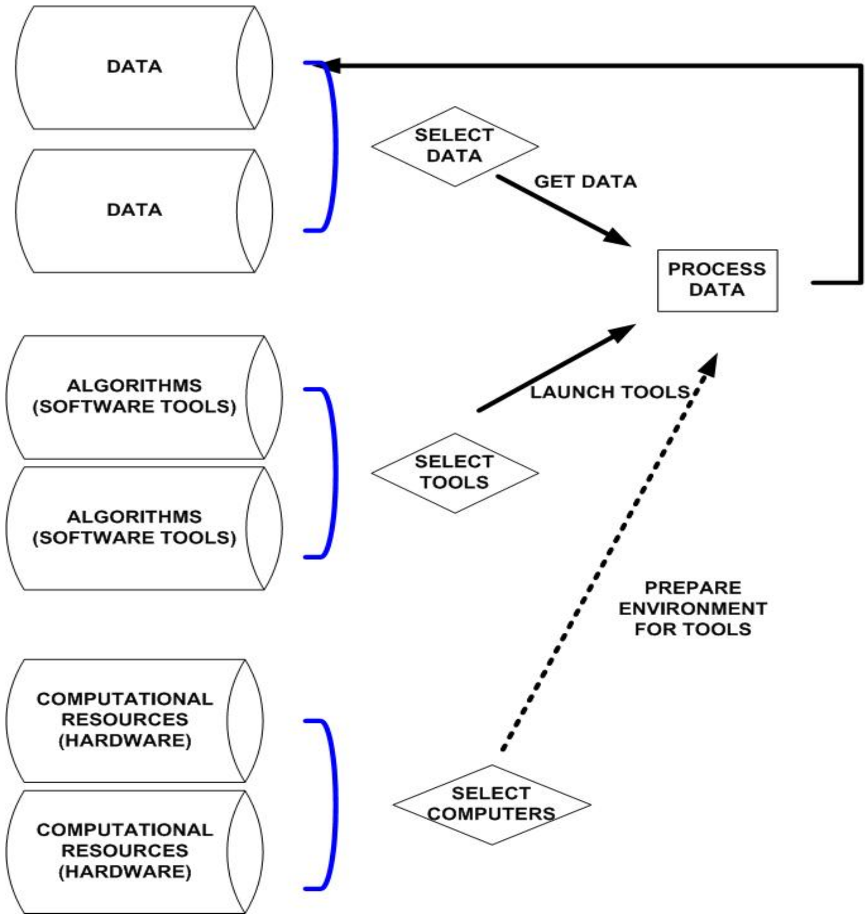

Figure 1-1: A typical sequence to execute one step of a scientific analysis. The output of the processing step either goes to the data repository or becomes an input to a new processing step.

From the process management perspective, a meta-workflow could be viewed as a system for (a) browsing and searching available data, tools and computational resources, (b) accessing available data sets, tools and computational resources, (c) bringing them together; (d) executing one tool at the time or a sequence of tools, (e) monitoring and controlling executions and (f) efficiently utilizing available data, tools and computational resources. These features are mapped into the design of Cyber-Integrator architecture.

The overall architecture of Cyber-Integrator is illustrated in Figure 1-2. The key components are the meta-workflow (MWF) editor: engine, executors, applications, collections of registries, and optional meta-data repository and event broker. They are all written in Java. The editor serves as a user interface to the gamut of Cyber-Integrator functionalities. The engine code coordinates the execution by multiple executors (remote or local). Each executor is a specific code to interact with applications and each application is the actual installation of the piece of code that has to be executed. The registries could be viewed as repositories of high-level descriptions of available data, tools and resources. The meta-workflow store provides a repository for gathered information about meta-workflow execution and it is based on a resource description framework (RDF) format. Finally, the event broker is a component for handling a stream of data or events, and it is based on Java Message Service (JMS) application programming interface (API) for sending messages between two or more clients

The arrows describe the communication among the key components of the meta-workflow system. For example, on the right side of the chart, the editor uses the data, tool and resource registries to create a triplet. The triplet forms a step in the meta-workflow. This step is passed from the editor to the MWF engine. The MWF engine then executes this step using an appropriate executor communicating with each application. The output of this step is sent by the MWF engine to the data registry.

As the MWF engine is executing a step, provenance information is generated, i.e., step 1 started execution at time X. The provenance information is sent to the meta-data store (see the left side of the chart). Provenance information gathering by the MWF engine could also be viewed as a sequence of events. These events could trigger other actions. For instance, an event "step is finished executing" reported by the MWF engine could trigger the execution of another step, i.e., "show image" step is being triggered after finishing "load image" step. Events are not only generated by the engine but also by the editor and hence the link between the editor and the event broker. For example, event broker could wait for an event "a new step has been added to the meta-workflow" sent by the editor and notify the MWF engine about it.

Finally, the link between the meta-data store and the editor refers to the system feedback during workflow creation. The editor can leverage provenance information, for example, by making recommendations based on past executions. The recommendation feature uses provenance information to recommend what tool to use next in a workflow sequence.

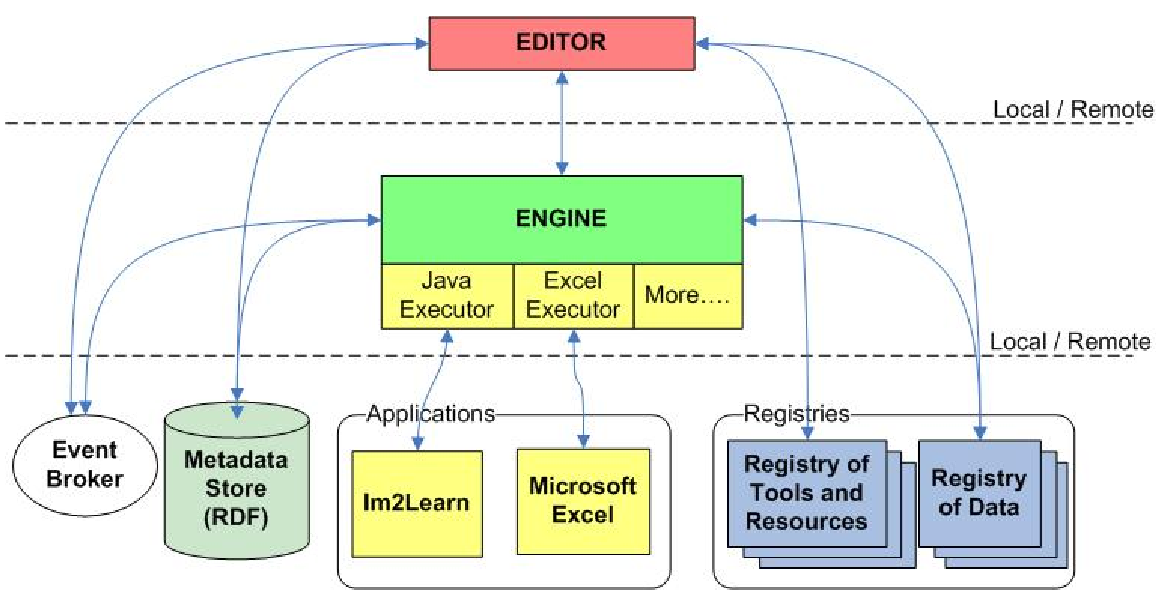

Figure 1-2. The overall architecture of Cyber-Integrator

.

Chapter 2 Installation Instructions

2.1 Hardware requirements

A computer with at least 512MB of RAM is recommended. Some processing steps might require significant computational resources. However, for most purposes, a regular desktop computer should be able to handle the processing. The execution duration depends on the hardware specification and the algorithmic computational needs.

2.2 Software requirements

- An Operating System capable of running Java
- Java 1.5

2.3 Installation steps

2.3.1 Windows

Extract cyberintegrator.zip with your tool of choice (e.g. winrar or winzip). To run the application, double click on cyberintegrator.bat.

Note: cyberintegrator.bat is simply a text file with several command line parameters. If you would ever experience out of memory issues (visible through the command line window or Help->Logging), then you might change the -Xmx512M parameter to something higher (e.g. -Xmx1024M) with Notepad.

2.3.2 Unix-based systems

The installation of Cyber-Integrator on a unix-based system is very similar to the Windows installation. From the command line, you can extract Cyber-Integrator with the command “unzip cyberintegrator.zip” and you may run it with the command “./cyberintegrator.sh”. sh files can be edited in the same manner as .bat files, with a text editor.

Chapter 3 How to use Cyber-Integrator

3.1 Main window

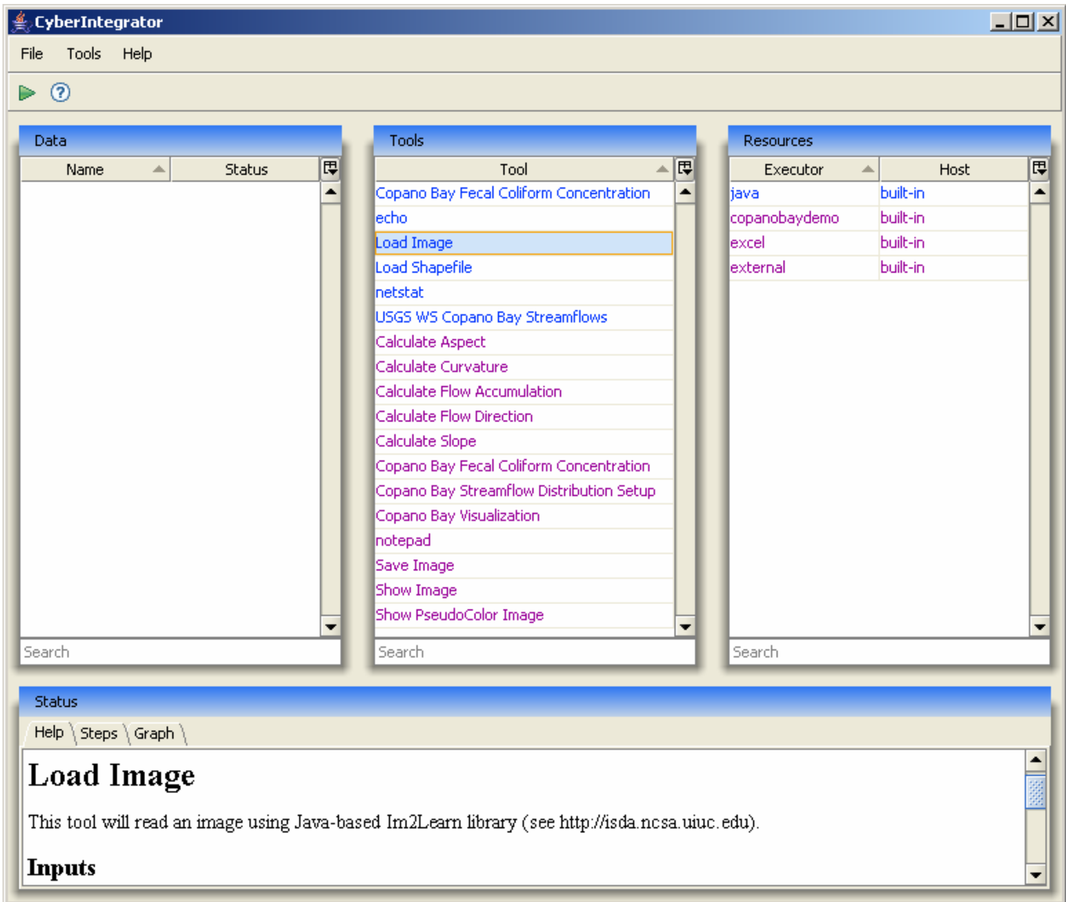

Figure 3-1: Cyber-Integrator editor.

The Cyber-Integrator user interface is shown in Figure 3-1. The current meta-workflow editor includes (1) three browsers of information registries (data - left, tools – middle, and executors - right) and (2) presentation of system information (bottom). At the top of the window there are buttons for a) executing the selected tool and b) recommending the next tool to be used in the workflow.

The Data browser lists all data sets loaded or generated so far for processing. The Tools browser lists the tools currently loaded from local and remote registries. The Resources browser lists computational resources (executors) available for a particular tool.

The bottom pane in Figure 3-1 contains several tabs with information about the Cyber-Integrator execution. The Help tab contains help text about the tool currently selected. The Steps tab contains a list of the tools run so far with information about each execution. The Graph tab includes a graphical representation of the sequence of steps. The current meta-workflow can also be saved and either reloaded or shared with others later.

Cyber-Integrator consists of a core set of functions and data structure conversions that represent solutions for specific tasks.

In order to fully understand how to use Cyber-Integrator, you should read through the entire user guide, particularly Chapter 4 Application Examples. *However, the usage of Cyber-Integrator can be summed up in two points*: 1) clicking on an item in a browser will update the other browsers to display the only valid choices to produce a triple. For example, clicking on a tool will update the Data and Resources browsers to show (in black text) what data and what executor will work with that tool. The unavailable data and resources will be grayed out. 2) The green "execute" button will be grayed out until the Data, Tool, and Executor is a valid triple. To see this visually, look at the figure below.

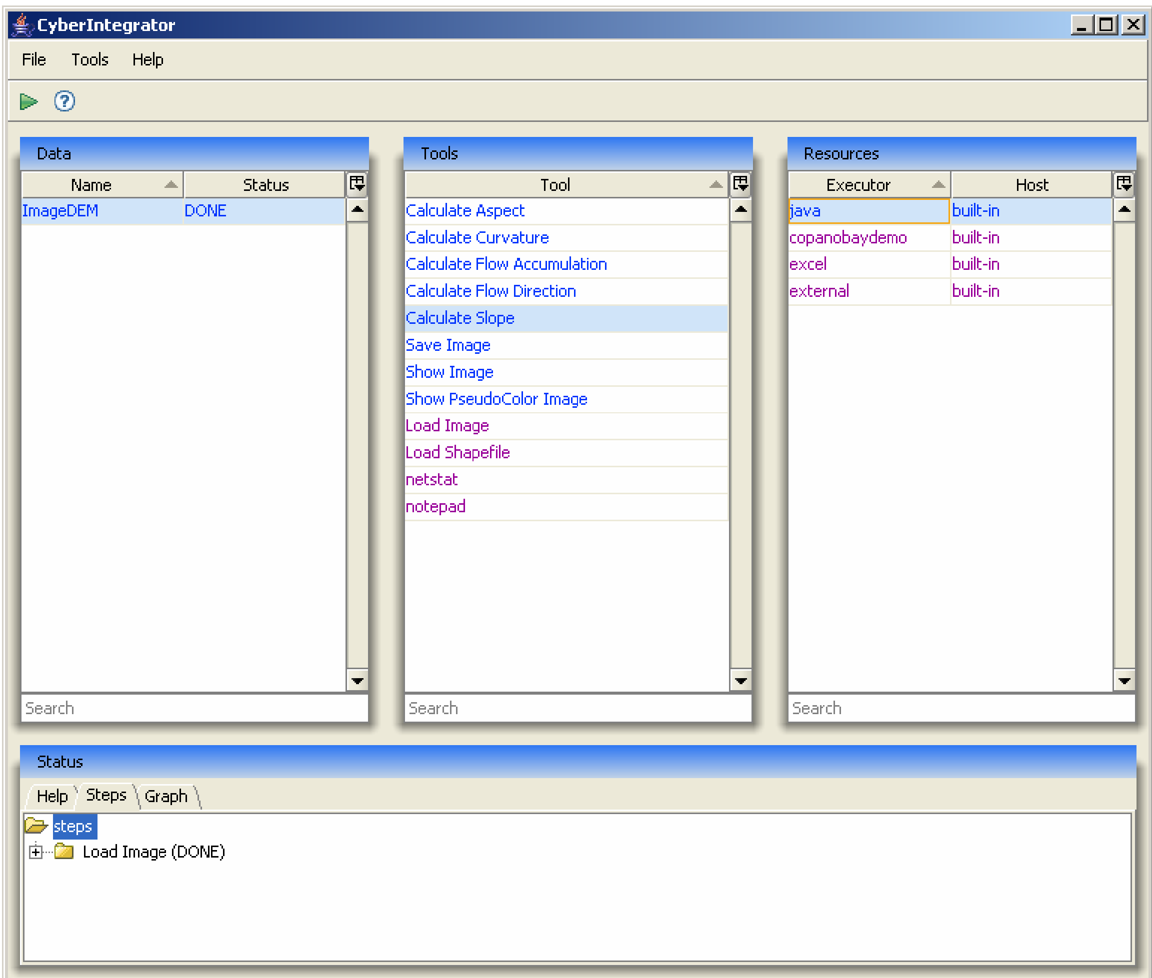

Figure 3-2 Valid triple ready to execute

3.2 Browsers

All browsers have search boxes at the bottom to allow for quick searching of the lists. They also have a common UI for changing the display of their columns.

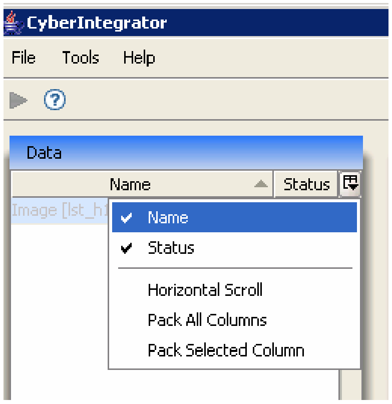

Figure 3-3: Column Options

Columns can be added or removed by clicking on them. **Horizontal Scroll** allows the columns to be wider than the browser box, and the **Pack** options resize the columns to be the appropriate width.

Data The data browser displays the name of the data and the status of the Execution. The name is defined in the tool registry. The status describes the state of the tool execution. **DONE** means that execution was successful, while **Failed** indicates that it was not. Items in this list can be right clicked on to display specific information about them. If you wish to select multiple pieces of data for a tool, you may use shift or control-click to do so. *Also, perhaps more importantly, you can deselect selected data by control-clicking on it.*

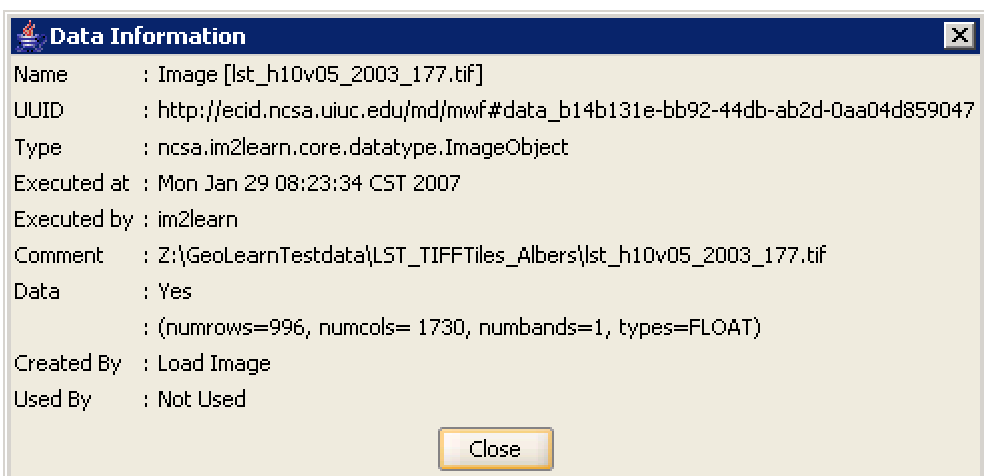

Figure 3-4 Data Information

Tools This browser displays the tool names and also dictates the contents of the Help box. The possible executors for a tool can be displayed by choosing **Executors** in the column options for that browser. *Note:* Not all possible executors may be installed (check the resources browser).

Resources This browser displays the executor name and its host. “Built-in” indicates that the executor is located on the local host.

Status This part of Cyber-Integrator displays Help, Steps, and a Graph section.

3.3 Menu Bar

3.3.1 File Menu

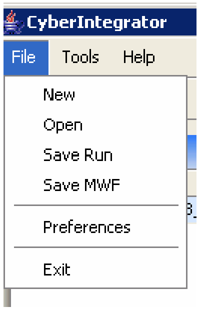

Figure 3-5: File Menu

Click on **New** to start a new blank meta-workflow.
Note: **New** will not give you the option to save the current meta-workflow. Please take care to save your Run or MWF (see below) if you do not wish to lose your work!
Open will produce a dialog to open .mwf files.
Save Run will save both the data and the meta-workflow (see Section 4.3).
Save MWF will save only the meta-workflow steps (see Section 4.3).
Preferences will bring up a preferences dialog. The default user interface colors can be changed in the resultant window (see Figure 3-6).
Exit will quit the application. Again, make sure to save your work before doing this since Cyber-Integrator will not give you the choice after quitting.

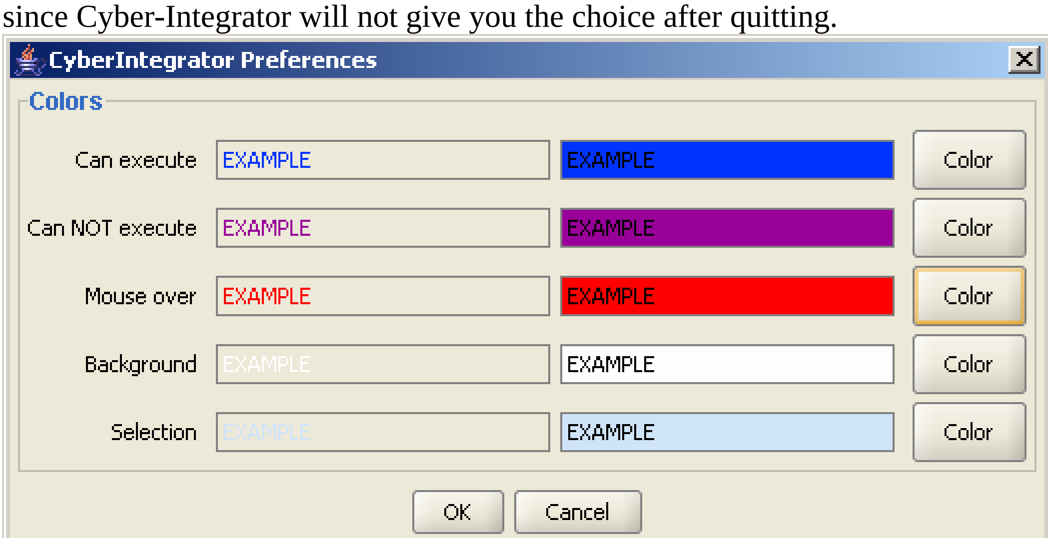

Figure 3-6: Preferences Dialog

3.3.2 Tools Menu

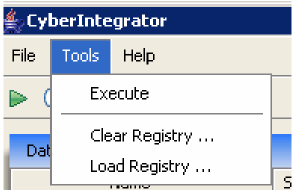

Figure 3-7: Tools Menu

Execute has the same functionality as the green arrow. It will run the tool if the Data, Tools, and Resources are a valid triple.
Clear Registry will clear the available tools and resources.
Load Registry allows you to load new tools and resources through an xml file called a registry. See Chapter 5 for more information about registries.

3.3.3 Help Menu

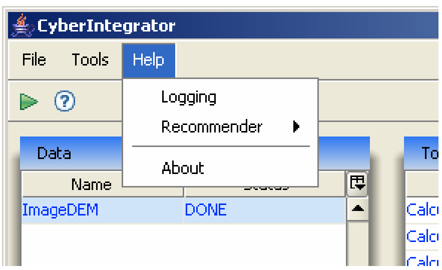

Figure 3-8: Help Menu

Recommender->what tool to use? has the same functionality as the question mark button. It will recommend the next tool to be used based on statistics gathered (metadata) from local Cyber-Integrator usage.
About gives credit to the creators of Cyber-Integrator as well as copyright information.

Chapter 4 Application Examples

Applications areas in which meta-workflows could be used are many. Some applications include environmental engineering, bioinformatics, hydrology, precision farming, plant biology, medicine or security. We present a couple of examples to illustrate workflow sequences for (1) feature extraction (slope calculation from elevation maps) and (2) water quality simulation (total maximum daily load simulations).

4.1 *Feature Extraction:* Slope Calculation from Elevation Maps

The goal is to construct a workflow for loading elevation maps, calculating slope from the elevation maps and display the original elevation map and the derived slope image. This type of calculation is very common in many GIS applications. It is included here as one of the simplest workflow sequences in order to familiarize the user with the Cyber-Integrator user interface.

4.1.1 Execution Sequence

This application example is primarily using Im2Learn functionality that is executed using the Java executor. The Im2Learn jar file is distributed with Cyber-Integrator and a java run time environment is assumed to be present.

1. Start the Cyber-Integrator
2. A user selects the tool "Load Image" first. An example elevation map can be found at data\test_data\DEMCD1AndCD2_ILsubSamp50.hdr
3. The loaded image can be visualized by selecting the input image in the left browser and the "Show PseudoColor Image" tool in the middle browser. The visualization is shown in Figure 4-1. The loaded image represents a digital elevation map (DEM) of Illinois.
4. Next, a slope of the DEM image can be computed using the java executor as shown in the right browser after selecting the input image (left browser) and the tool "Calculate Slope" (middle browser). Visualization of the slope image is shown in Figure 4-2.
5. The entire workflow process can be visualized by clicking on the bottom tab "Graph". The graph of the workflow is shown in Figure 4-3.

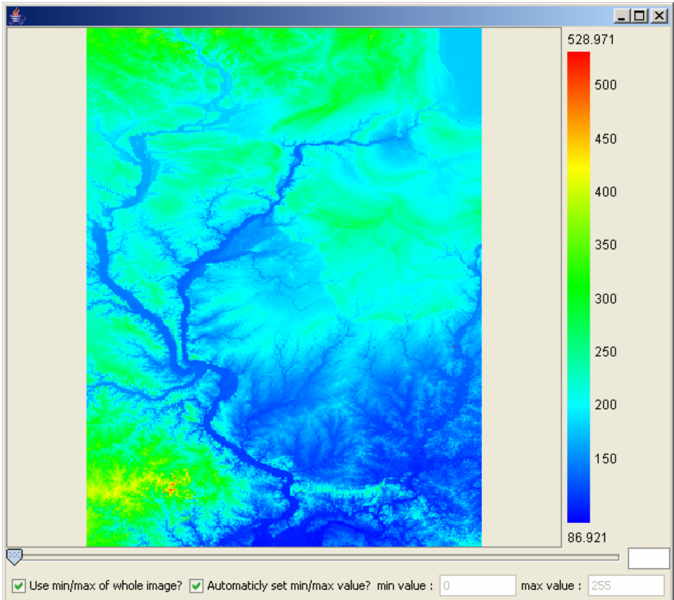

Figure 4-1: A DEM image loaded and visualized using "ShowPseudocolor" tool.

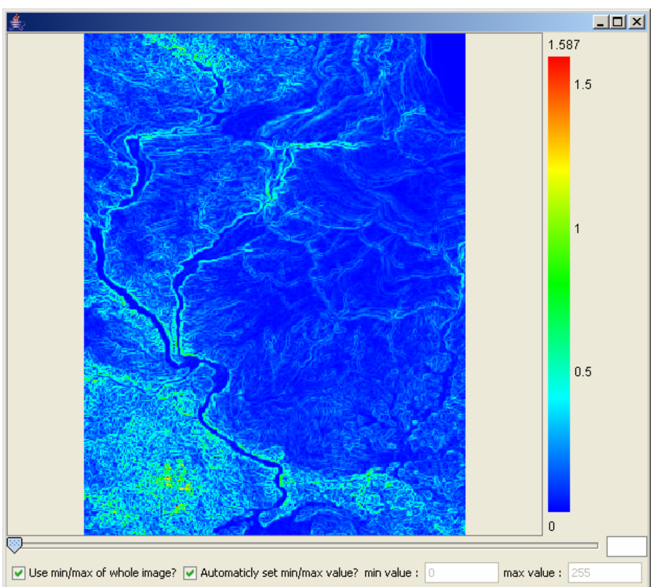

Figure 4-2: A slope image compute from the DEM image and visualized using "ShowPseudocolor" tool.

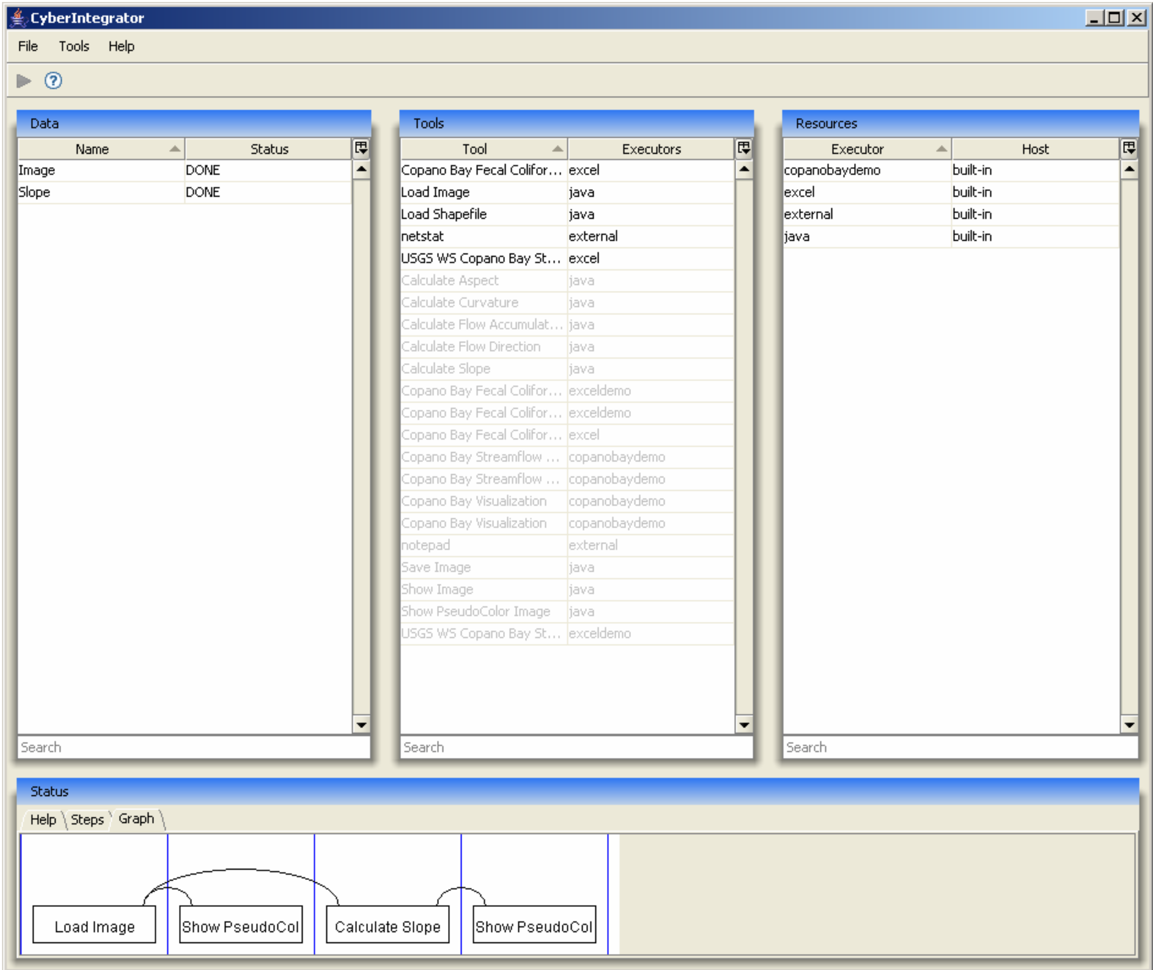

Figure 4-3: A DEM image loaded and visualized using "ShowPseudocolor" tool.

4.1.2 Extended Execution Sequence

During the workflow execution provenance, data are collected and stored in a local meta-data repository. A conceptual organization of the workflow provenance information is shown in Figure 4-4. The Cyber-Integrator can also save settings and outputs via the File menu (see Section 3.3.1) so that meta-workflows can be reproduced at a later time.

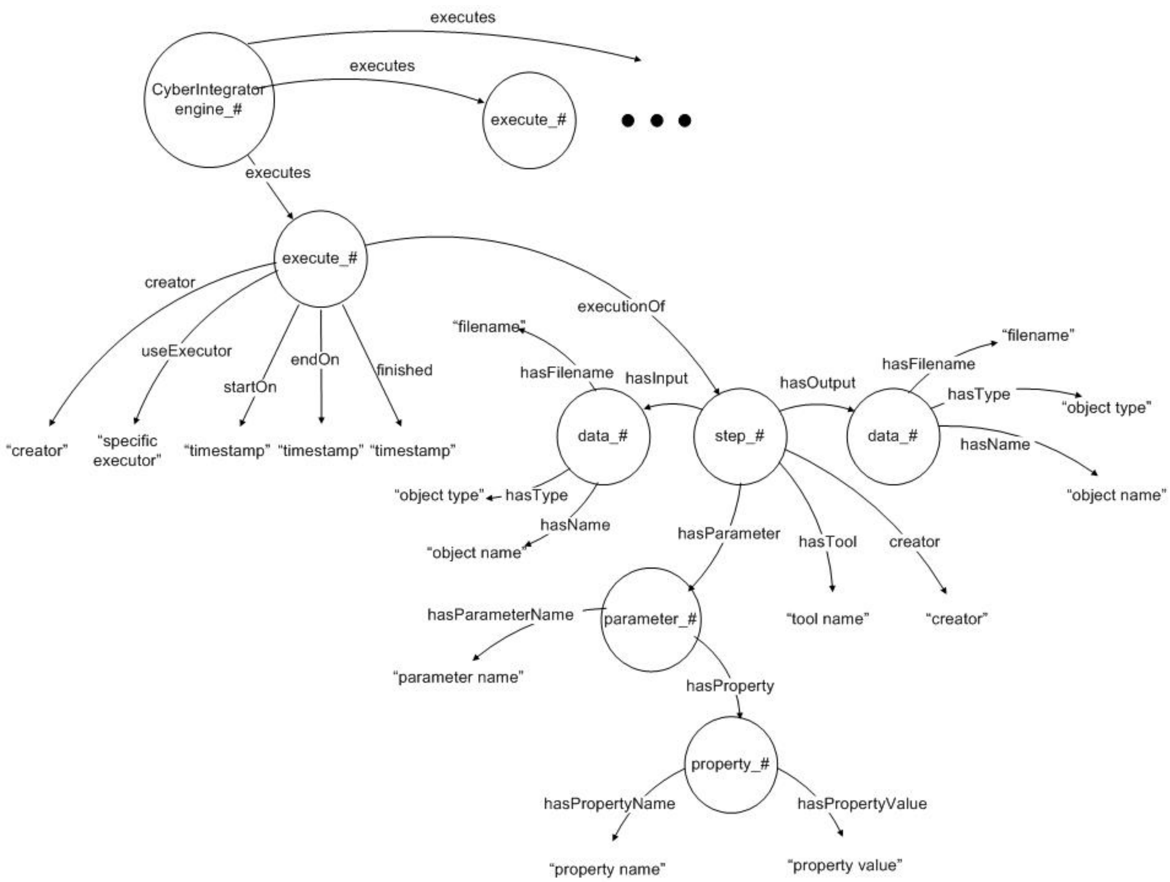

Figure 4-4: A conceptual organization of the workflow provenance information.

The provenance meta-data are analyzed by the recommendation tool. The system provides real-time recommendations of tools and data sets that can be initialized from the Cyber-Integrator by clicking the question mark button at the top of the window. For example, before loading any data set, the recommendation is initialized and the pop-up message shown in Figure 4-5 indicates the past uses of the tools for loading data.

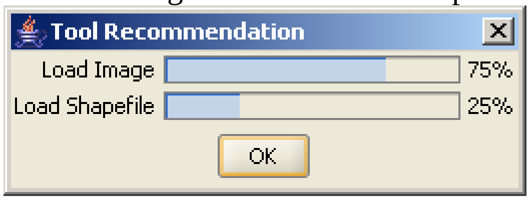

Figure 4-5: A recommendation dialog based on the provenance meta-data information.

4.2 Water Quality Simulation: Simulation of Bacterial Total Maximum Daily Loads (TMDL) for Corpus Christi Bay (CCBay)

This summary is adapted from Ernest To's description of his Monte Carlo simulation using Carrie Gibson's bacterial loadings model[3].

The Clean Water Act (CWA) requires that each State identify water bodies that do not meet the State's water quality standards. The study shown here focuses on Copano Bay, located off the Gulf of Mexico in Texas. The Texas Commission on Environmental Quality (TCEQ) has identified portions of the bay as impaired for contact recreation and oyster water use due to high bacteria levels. The Texas oyster water use standards require that the median fecal coliform concentration cannot exceed 14 cfu/100mL and that the 90th percentile value (i.e., the upper 10% of the samples) cannot exceed 43 cfu/100mL. In order to meet these standards, Total Maximum Daily Loads (TMDLs) from the various bacterial sources must be established, which are the maximum amount of pollution that a water body can receive each day and still retain its uses.

Carrie Gibson and Dr. David Maidment of the University of Texas at Austin have performed a significant amount of research for the Copano Bay TMDL and have identified important bacterial sources in the Copano Bay watershed and their associated loadings. Carrie developed a fecal coliform model in ARCGIS using Model Builder that predicts annual average fecal coliform concentrations in the bay. Ernest To converted the model to a macro in an Excel spreadsheet so that it could be run as a Monte Carlo simulation to predict median and 90th percentile values for the TMDL. He also created another spreadsheet that would download USGS data using a Web service and fit distributions to the data, which are then used in the Monte Carlo simulation.

This demonstration shows how the Cyber-Integrator can enable researchers to easily link a variety of tools and automatically pass data from one to another. Ernest's MS excel spreadsheet macros are run and, at the click of a button, the results and an ESRI Shapefile of the reservoir are fed into a visualization tool from Image2Learn (Java code) to create a visualization of the likelihood of exceeding the water quality standards at each schema node in the watershed. The nodes are locations in the watershed where bacteria levels are measured; nodes are spatially linked in the model as shown in the visualization. Bacterial loadings are predicted at each node, with the loadings originating from a variety of sources (e.g., livestock, wastewater treatment plants, water birds, and leaking septic systems) and being transformed via first-order bacterial decay kinetics.

4.2.1 Execution Sequence

[3] See https://webspace.utexas.edu/tosc/www/index.mht

Introduction

This application example is primarily using MS Excel and Im2Learn functionality. It is assumed that a user has a local installation of MS Excel prior to the MWF execution. The Im2Learn jar file is distributed with Cyber-Integrator and a java run time environment[4] is assumed to be installed.

1. Start the Cyber-Integrator
2. Retrieve USGA Streamflow Data
    - Execute the **USGS WS Copano Bay Streamflows** spreadsheet by selecting it from the list and clicking on execute. A dialogue box will pop up, asking the user to select a Gage Id (see Figure 4-6). Select any, click OK. The output will be collected on the left pane under "CBay Streamflow distribution parameters". This is an expensive operation and might take quite a while to finish.
    - This first step downloads daily streamflow data of a given USGS gage from the internet and performs statistical analyses on it using an Excel macro. It matches the flow distribution with a lognormal distribution and outputs the distribution parameters.
3. Create Streamflow Distribution for full Schematic Network
    - Select "CBay Streamflow distribution parameters" data and "Selected Cbay Streamflow Gage ID" (using control or shift-click) created in the previous step. Execute the **Copano Bay Streamflow Distribution Setup** tool. This setup does not require parameters to be set. It will output "Distribution Parameters" on the left pane.
    - This step takes the distribution parameters from the previous step and maps them to all of the nodes in the watershed for the Copano Bay schematic processor network. Each gage affects multiple schema nodes in the schematic processor model for Copano Bay. This step makes sure that the right streamflow distribution parameters are available for each node and creates a structure that can be used by the next tool (if run with an input).
4. Execute the **Copano Bay Fecal Coliform Concentration** spreadsheet model
    - Select the "Distribution Parameters" created in the previous step, and execute the Copano Bay Fecal Coliform Concentration on it.
    - Two dialogue boxes will pop up. One box is to specify a hydro node id in the network for which to run the simulation (see Figure 4-7) and one is to specify how many times the simulation will be run.
    - Click on "steps" to see the steps so far and current progress. Note that you can continue setting up later steps and the system will execute those steps when ready.
5. Execute the **Load Shapefile** Im2Learn(Java) tool.
    - Navigate to the data/CopanoBay directory and load the CopanoBayWatersheds.shp file, which is an ESRI Shapefile.

---

[4] According to http://www.objectinnovations.com/Guides/JavaVersions.html:
The Java Runtime Environment: this is the package of software that must be installed on a machine in order to run Java applications

6. Execute the **Copano Bay Visualization** tool, which uses Im2Learn.
    - Select the shapefile that was loaded, the CBay Fecal Coliform Concentrations, and the Node ID and Number of Iterations. Execute the "Copano Bay Visualization" tool to get a view of the values collected from the Monte Carlo simulation (see Figure 4-8). The visualization shows the watersheds with the schematic processor bacterial loadings model for Copano Bay that was used in the Monte Carlo simulation. The table with the loadings shows values in red if they are above the threshold and in blue if they are below.
7. Save your workflow when you finish.
    - You or someone else can load the workflow later and see all the steps you took, and continue from here. (Later versions of Cyber-Integrator will also allow you to delete or add steps to existing workflows.)
    - A unique feature of Cyber-Integrator, among other workflow tools, is its ability to allow you to create your analyses by example in this stepwise fashion, and replay them later. This could be set up in an automated fashion to allow real-time updating of your analyses as new data are received.

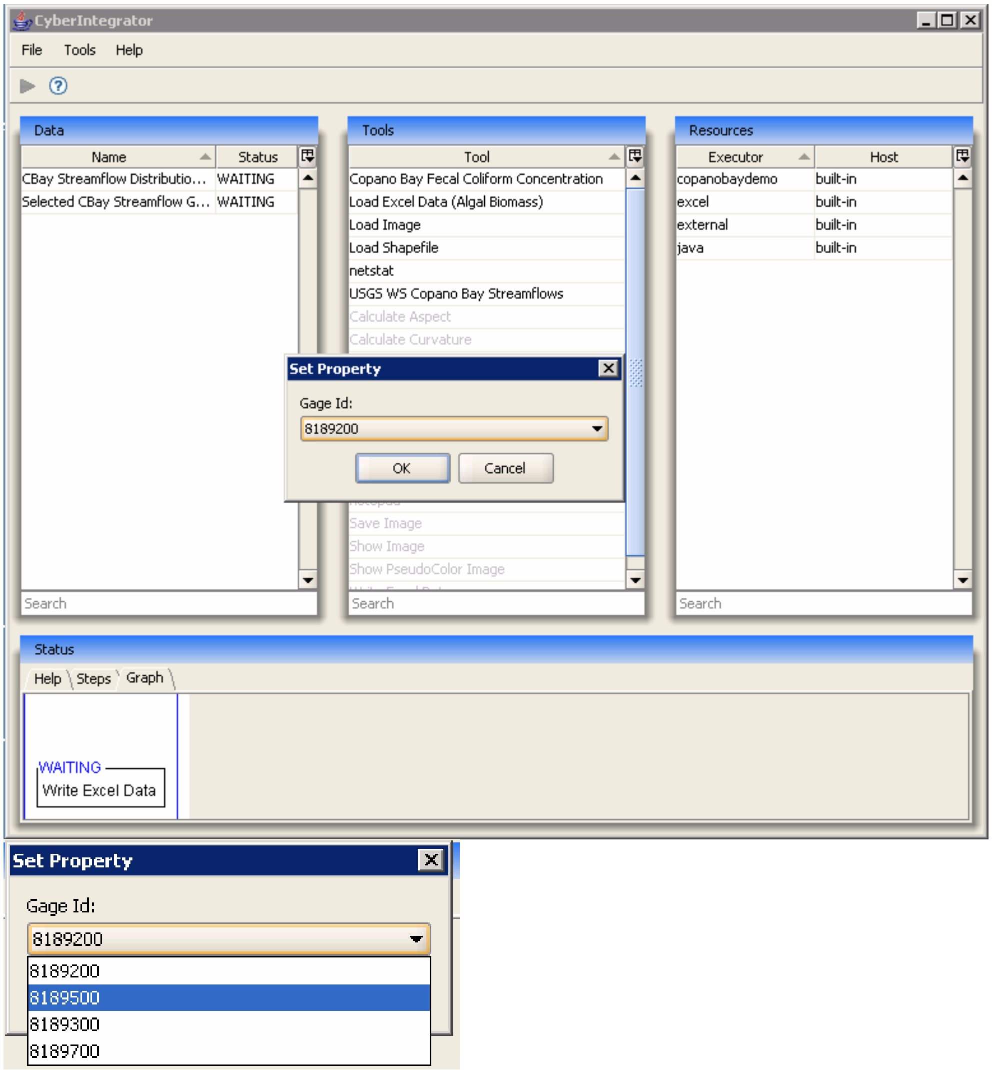

Figure 4-6: User input is collected using a dialog for selecting gage identification.

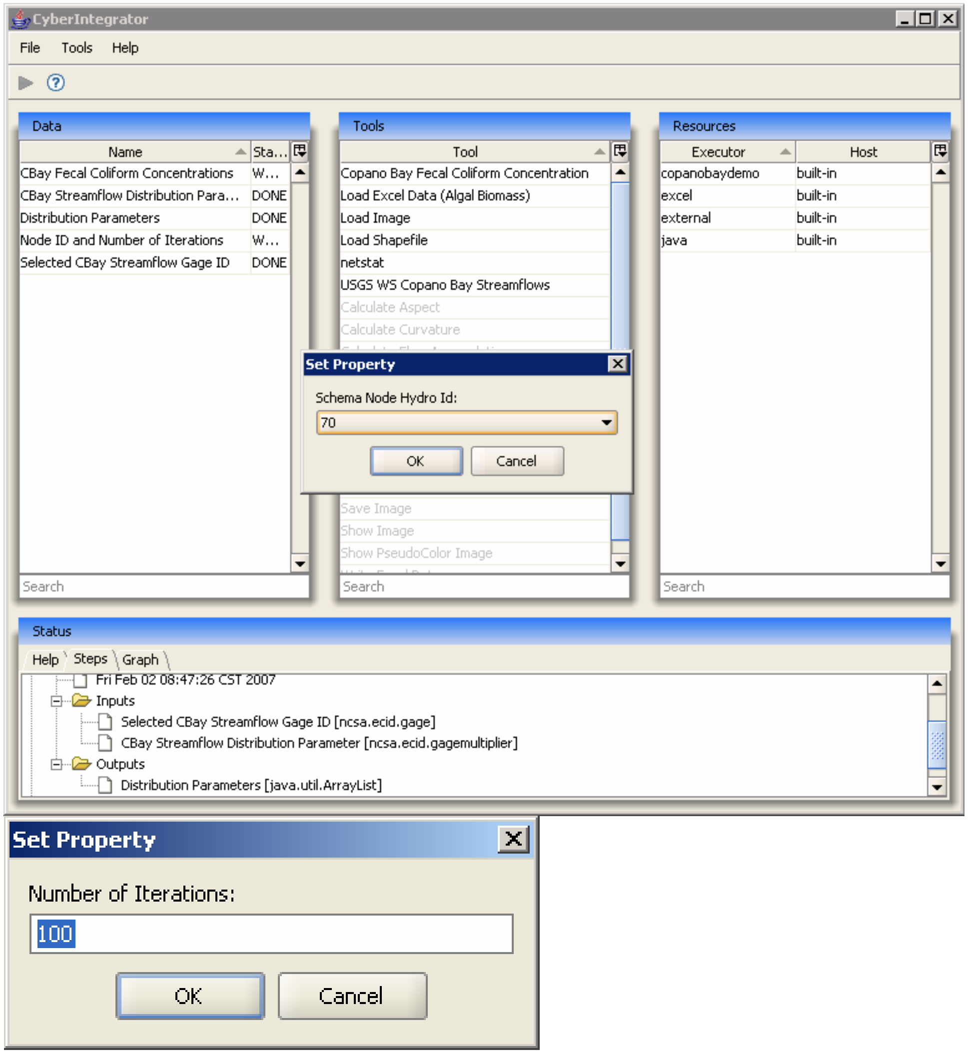

Figure 4-7: Dialogs for selecting schema node hydrological identification and the number of iterations of the Monte Carlo simulation

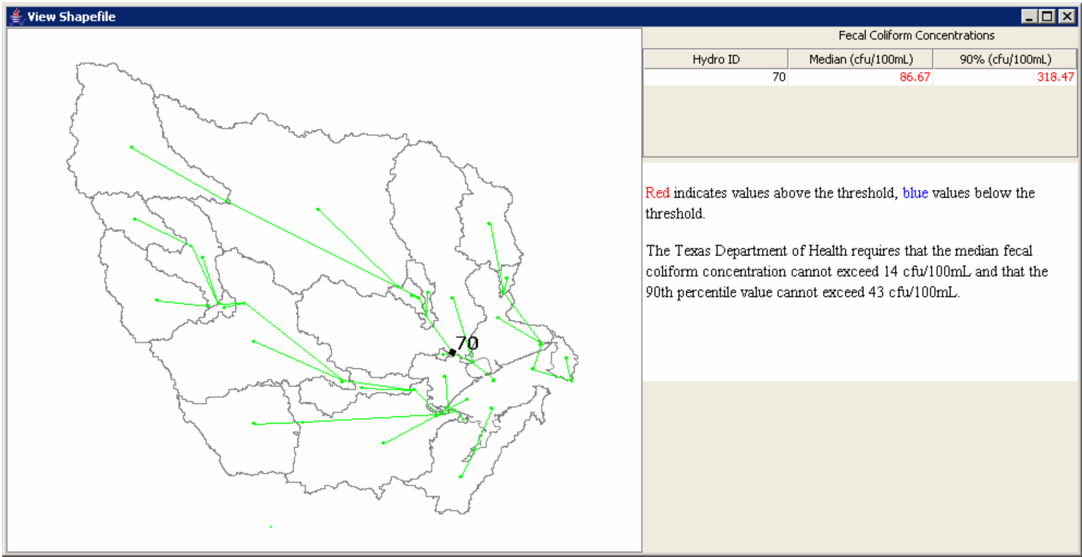

Figure 4-8: Visualization of the likelihood of exceeding the water quality standards at each schema node in one of the Corpus Christi watersheds.

4.3 Rerunning, Modifying and Extending Application Examples

Let us suppose that one would like to rerun an already created meta-workflow and perhaps modify the parameters of multiple tools. This can be achieved by loading a .mwf file with the sequence of steps only and deciding whether input parameters of steps should or should not be modified before re-running the meta-workflow. It is also possible execute one tool at the time to inspect the execution one by one.

In another scenario, one would like to continue extending an already existing meta-workflow run that took several hours/days of computation. This could be achieved by loading a .mwf file with the data. Be aware that while the .mwf file in this case was generated from the File/SaveRun menu, the .mwf file described in the previous paragraph was generated from the File/SaveMWF menu.

We describe next how to operate a loaded meta-workflow that would contain either a sequence of steps only (meta-workflow only) or a sequence of steps with intermediate data sets (meta-workflow run).

4.3.1 Illustration Example

Meta-workflow only: The release contains a saved metaworkflow file in data/test_data/applicationExamplesMWF.mwf. The following sequence of steps can be executed in order to demonstrate the capabilities.

(1) Load the applicationExamplesMWF.mwf file via File/Open menu. The dialog shown in Figure 4-9 will appear:

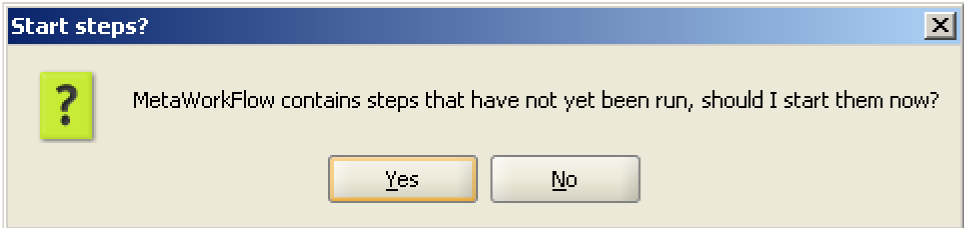

Figure 4-9: Dialog for choosing the meta-workflow to re-run or not.

(2a) If Yes is selected then the dialog with all meta-workflow parameters will appear as shown in Figure 4-10. One can modify any parameter before re-running the meta-workflow.

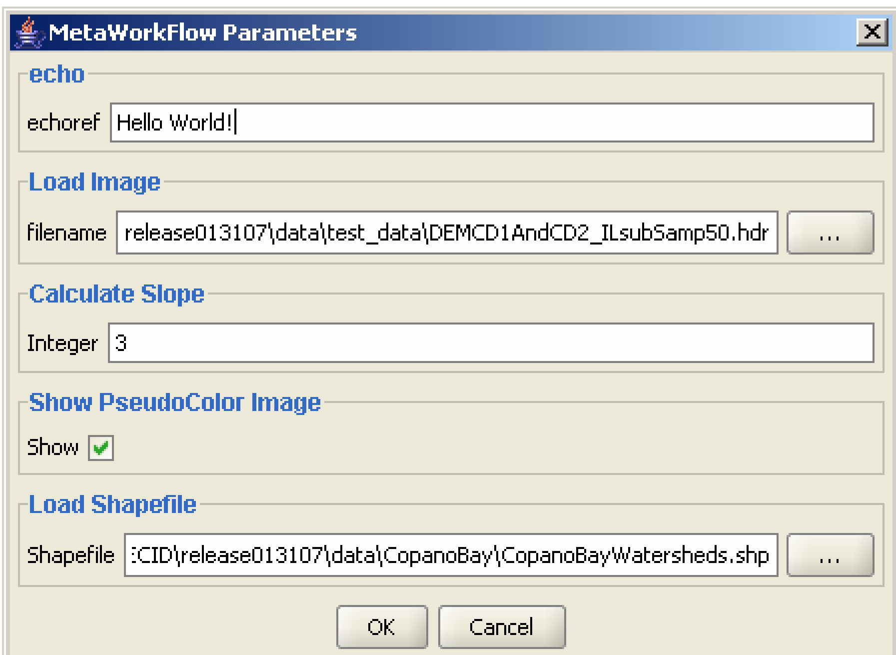

Figure 4-10: Dialog with meta-workflow parameters

(2b) If No is selected then the computation will not take place. A user would see a label "NOTSTARTED" next to data sets in the data pane, and next to steps in the graph or step view on the bottom of the Cyber-Integrator user interface (see Figure 4-11). The steps can be now re-executed one by one by right clicking on the graph (see Figure 4-11 bottom left) or step name in the step view (see Figure 4-12 bottom), and selecting edit/start or start. Figure 4-13 shows a case where some tools were re-executed, one is running and others are NOTSTARTED.

Introduction

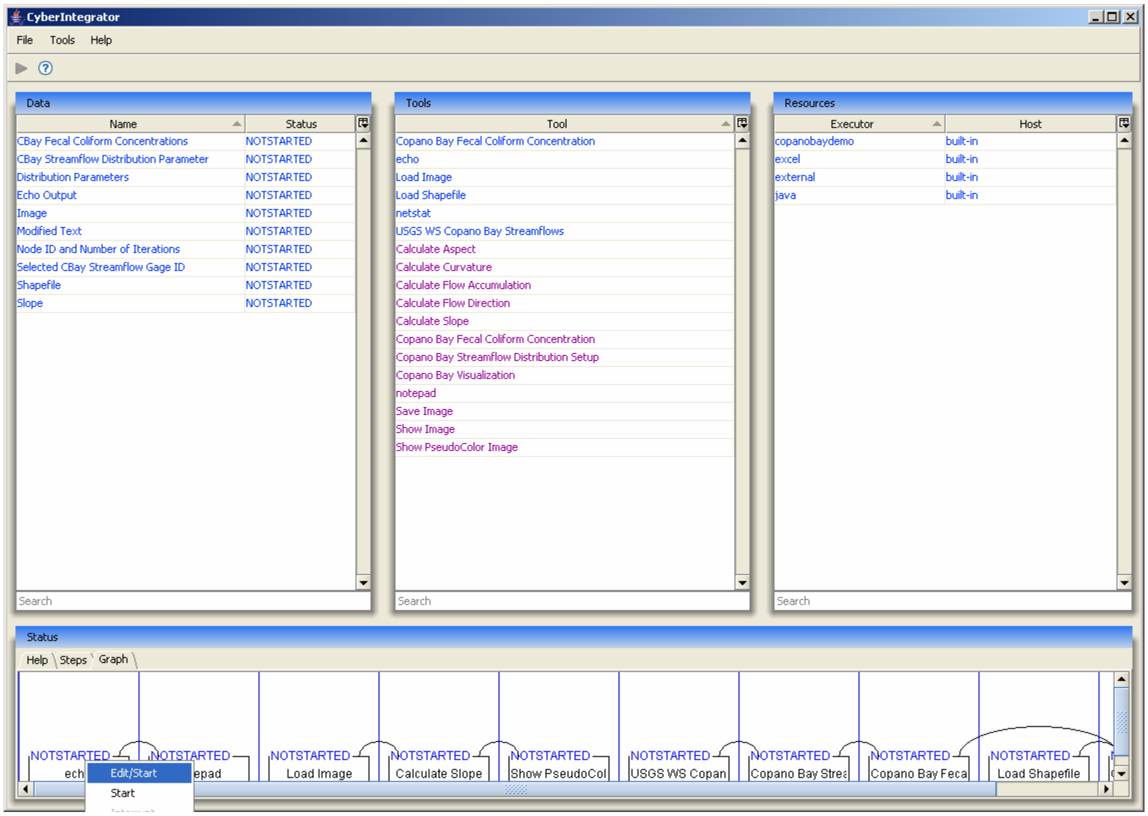

Figure 4-11: Loaded meta-workflow without re-execution.

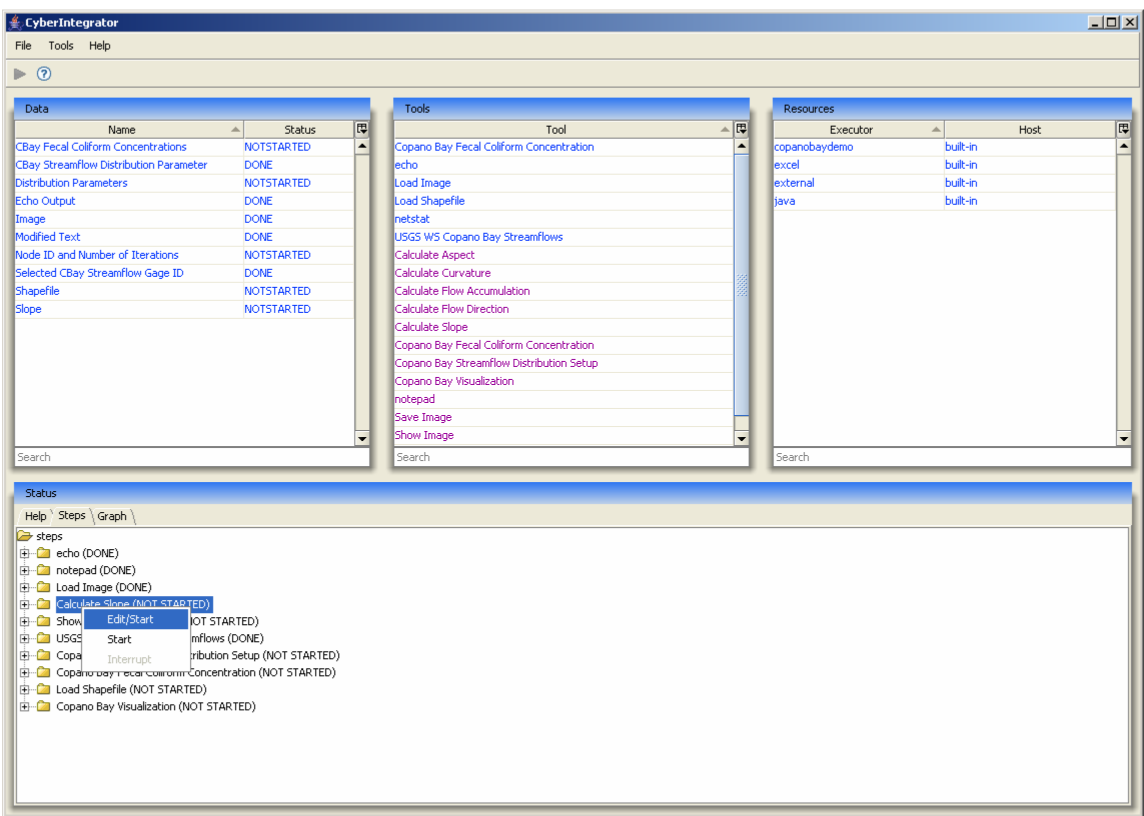

Figure 4-12: Re-execution of a tool “Calculate Slope” from the Step tab.

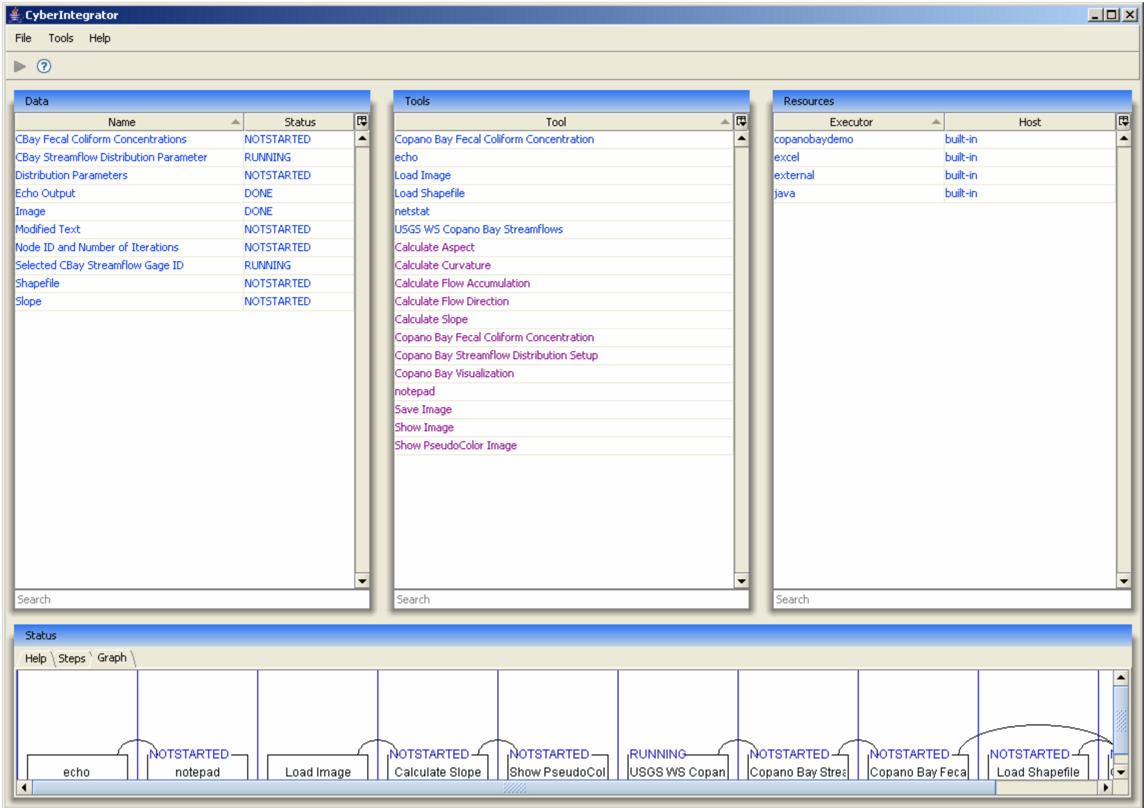

Figure 4-13: Loaded meta-workflow with two tools re-executed individually and one tool still running.

(3) If an execution of step takes too much time then it could be interrupted by right clicking on the step (graph or step view) and selecting the Interrupt option. In the current release, it is not possible to remove a step from the meta-workflow sequence.

Chapter 5 Advanced Operations: Adding Tools to Cyber-Integrator

This section describes one of the advanced operations with Cyber-Integrator such as adding new tools. In this section we assume that Cyber-Integrator already contains an executor for the new tool that should be added.

In order to add a tool, one has to become familiar with the XML structure of tool registries. It is also assumed that a user is familiar with the XML syntax and can also read the XML schemas provided with the registry files.

We provide a short description of where to find the registry files, how to modify the already existing tool settings and how to add a new tool by creating a tool description.

5.1 Registry Location and General Principles

The registries for all executors are currently located in the base Cyber-Integrator folder**data\registry** directory. For example, the following tool for viewing any text using MS notepad can be found in the external.xml file in the data/registry folder.

```
      <tool name="notepad" uuid="external-2">
        <input name="Original Text" type="text" id="0"/>
        <output name="Modified Text" type="text" id="0"/>
            <executor>
                  <external stderr="true">
                        <executable>notepad</executable>
                        <option>
                              <data input="0" output="0"/>
                        </option>
                  </external>
            </executor>
        <help>MS Windows text editor for editing the text that is
passed in as input.</help>
      </tool>
```

The structure of the tool descriptions in registries is provided in the base Cyber-Integrator folder**data\schema** directory. For instance, the corresponding schema for the external tools described in external.xml is in ExternalExecutor.xsd. The schema defines what high level syntax is available to describe tools. A small example of a high level tool description in ExternalExecutor.xsd is provided below:

```
<xs:element name="external">
   <xs:complexType>
      <xs:sequence>
        <xs:element name="executable" type="xs:string" minOccurs="1"
maxOccurs="1" />
```

```
        <xs:element name="option" type="cmdoption"  minOccurs="0"
maxOccurs="unbounded" />
        <xs:element name="exitcode" type="xs:integer" default="0"
minOccurs="0" maxOccurs="1" />
        <xs:element name="directory" type="xs:string" minOccurs="0"
maxOccurs="1" />
        <xs:element name="path" type="xs:string" minOccurs="0"
maxOccurs="1" />
        <xs:element name="env" type="envvar" minOccurs="0"
maxOccurs="unbounded" />
        <xs:element name="captureStdout" minOccurs="0" maxOccurs="1">
              <xs:complexType>
                  <xs:attribute name="refid" type="xs:string" />
              </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="stderr" type="xs:boolean" default="true" />
   </xs:complexType>
</xs:element>
```

The registries can be opened using a text editor (e.g. wordpad). The language defines a tool, its inputs and outputs, a help section for information about the tool and its creator, and its executor. The placement of a tool is up to a user. For example, several ccbay.xml tools use the excel executor but are placed inside of ccbay.xml file because the tools are aggregated into files according to the end application (rather than executors).

It might be advisable to view the “Help” and/or “Steps” tab for these tools in the Cyber-Integrator application while learning how to edit the registries, since they outline information that is defined in the tool registries. In general, one should become familiar with the schema before modifying or adding new tools. Once the registry modifications are finalized, the registry can be loaded during a run time via the Tools/Load Registry menu option.

5.2 Tool Descriptors

Let us consider a new tool saved in a file called myTools.xml.
An example of a generic tool in an xml tool registry could look like:

```
<tool name="A tool" uuid="mySymbol-0">
      <input name="My input" type="some.type" id="0"/>
      <output name="My output" type="some.type" id="0"/>
      <executor>
            ...
      </executor>
      <help> This tool does something.</help>
</tool>
```

5.2.1 Tool Tag

```
<tool name="A tool" uuid="mySymbol-0">
      ...
```

```
</tool>
```

The tool tag defines a tool **name** and a **uuid**[5], and encloses the other internal tags mentioned here. The Cyber-Integrator user interface displays the tool name to the user, while the uuid is used internally by the application to differentiate between tools. Each tool *must* have a unique uuid. For example, the next tool in our generic registry could have a uuid of mySymbol-1.

5.2.2 Input and Output Data Tags

```
<input name="My input" type="some.type" id="0"/>
<output name="My output" type="some.type" id="0"/>
```

If a tool is viewed as a black box then the **input** is what the tool ingests and processes, and the **output** is what the tool generates.

In the input and output tags, a **name**, a **type**, and an **id** are defined. The name is what is displayed in the Data browser, and the type is the object class that will be input or output. The class is usually specific to the executor, so check the executor section below for the correct type. However, the type can be something generic like int if that is what the tool inputs or outputs. If there are multiple inputs or multiple outputs, be sure to give each a unique id. If there is no input or no output, that tag can be omitted. Make sure that refid entries in the executor section match the intended input or output.

5.2.3 Help Tag

```
<help>This tool does something.</help>
<help><![CDATA[here's some help<BR><BR>copyright and point of contact
is provided here]]> </help>
```

The CDATA (character data) tag is used to indicate sections of text that the XML parser will ignore. You may add HTML to the help section, but only if it is contained within a CDATA tag. Otherwise, the HTML would be treated as XML and would not be read correctly. For example, the <BR> tag above is used to insert line breaks in that help section.

5.2.4 Executor Tag

```
<executor>

      …

</executor>
```

[5] According to http://en.wikipedia.org/wiki/UUID, a **Universally Unique Identifier** (UUID is an identifier standard used in software construction, standardized by the Open Software Foundation (OSF) as part of the Distributed Computing Environment (DCE).

Unlike the other tags, the executor tag contains terms specific to an executor.
Note: The **refid** must match the corresponding id from the input or output section.

The java and external executors are by their nature generic. As a result, we have included schema files that can be loaded into any XML editor. These schema files ensure that formatting rules are enforced and should make it easy to add or edit tools for those executors. Together, the java and external executors can run most of the tools you may choose to add.

Excel, copanobaydemo, and GeoLearn executors have also been included. You may wish to add/edit the excel tools in order to define custom .xls files, worksheets, ranges, macros, etc. The Copano Bay registry ccbay.xml often uses the excel executor, so you may want to see that registry for some advanced uses of excel. Editing the copanobaydemo or geolearn registry requires more advanced knowledge of their functionality.

5.3 Executor Specific Tags

5.3.1 Java Executor Tags

The Java executor by default includes tools for Im2Learn functionality. The java executor can implement tools from any jar file. The schema file is named JavaExecutor.xsd. The schema can be loaded into an XML editor to easily create and edit tools and will enforce correct formatting rules. The XML for the Java executor is very generic.

An example of a Java-based tool entry is provided below:

```
<java>
      <parameter name="filename" type="file" description="Filename of image"/>
      <jarLocation>data/applications/im2learn</jarLocation>
      <classesToInstantiate>
            <classToInstantiate className="ncsa.im2learn.core.io.ImageLoader">
                  <methods>
                        <method methodName="readImage">
                              <arguments>
                                    <argument className="java.lang.String"
parameter="true"                                    refid="0"/>
                              </arguments>
                              <return
className="ncsa.im2learn.core.datatype.ImageObject" refid="0"/>
                        </method>
                  </methods>
            </classToInstantiate>
      </classesToInstantiate>
</java>
```

A rudimentary understanding of java structure is required to understand Java tools. The **jarLocation** is where your java application is located. Basically you have the **className** and the **method** you will be calling. The **argument** section must correspond to the **input type** section and the **return** tag corresponds to the **output type** section. The provided schema should allow you to define jar locations and other parameters. See java.xml for more examples.

The JavaExecutor.xsd schema file outlines exactly how java tools can be constructed to an XML editor. However, it is fairly easy to understand if you wish to examine it manually.

5.3.2 External Executor Tags

This executor can run applications with parameters through the command line. Much like the java executor, the XML for the external executor is very generic. The schema file is named ExternalExecutor.xsd.

Let us assume that we would like to execute the netstat command and display the text returned by the external application. The following two tools are available for running the netstat command and showing the resulting text using Notepad:

```
<external stderr="false">
      <executable>netstat</executable>
      <captureStdout refid="0"/>
      <option>
            <value flag="-n"/>
      </option>
</external>


<external stderr="true">
      <executable>notepad</executable>
      <option>
            <data input="0" output="0"/>
      </option>
</external>
```

A rudimentary understanding of the command line is required. If you run these two tools in order you will be able to understand what they do easily. The **executable** tag indicates the location of the program, much like typing something into the Start->Run box. In the **option** section, the **value flag** indicates that the command “netstat –n” will be run. **captureStdout** takes the output from from netstat –n and puts it as text as defined in the **output** tag. Then, notepad can take text as input and output it as modified text.

The ExternalExecutor.xsd schema file outlines exactly how command-line tools can be constructed to an XML editor. However, it is fairly easy to understand if you wish to examine it manually. You may define elements like **path** or **env** to set the PATH or environment variables, respectively, but that is not required (minOccurs="0").

5.3.3 Excel Executor Tags

This executor is working on Windows only and requires installation of Microsoft Excel. Examples of reading from and writing to an excel file are provided below:

```
<excel workbookpath=" data/test_data/Test.xls">
      <query>
```

```
            <output sheet="Sheet1" range="B2:E4, L10:P15"
            refid="0"/>
      </query>
</excel>

<excel workbookpath=" data/test_data/Test.xls">
      <query>
            <input sheet="Sheet1" range="B2:E4, L10:P15" refid="0"/>
      </query>
</excel>
```

The **workbookpath** defines the location of the xls. Then, **sheet** defines the worksheet number in the xls file and **range** the section of the table. **Input**, **output**, and **refid** have corresponding entries in the data section. Remember, **output** generates data that will appear in the data browser. This means that output is used to read excel files. **Input**, which takes input from the data specified in the data browser, writes excel files. Since the worksheet location must be hardcoded, please experiment with entering the tools into myTools.xml file and editing the file name.

Another example of a tool for executing an excel embedded macro is provide below.

```
<excel workbookpath="data/CopanoBay/
USGS_TX_daily_streamflow_downloader_20060919_1807.xls">
      <macro name="import_daily_flow">
            <output sheet="Control_sheet" range="B3" refid="0"/>
            <output sheet="optimization_sheet" range="K10" refid="1"/>
            <parameter sheet="Control_sheet" range="B3" value="8189200"
                  prettyName="Gage Id"
possibleValues="8189200,8189500,8189300,8189700"/>
      /macro>
</excel>
```

Macro name defines the name of a macro that is part of the .xls file in the **workbookpath**. **Parameter sheet** defines the combo box that Cyber-Integrator displays. If you wish to define macros, several more examples are in ccbay.xml

5.3.4 CopanoBay Executor Tags

Types are specific to the function. For copanobaydemo, they are ncsa.ecid types.
The Copano Bay registry defines a Monte Carlo simulation by using excel macros and copanobaydemo functions. The excel macros are outlined in the excel section above. See ccbay.xml for examples. An example of one of the Copano Bay tools is provided below.

```
<copanobaydemo function="createDistributionMultipliers2">
      <input name="gage" refid="0"/>
      <input name="multiplier" refid="0"/>
      <output name="Distribution Multipliers" refid="0"/>
</copanobaydemo>
```

A **copanobaydemo function** is defined. The input and output behave like the other executors.

5.4 Illustration Example

Let us assume that we would like to add an external tool that is one of the functions available from the MS DOS command environment. Particularly, our goal is to include the echo function from the MS DOS command environment into Cyber-Integrator and display the text passed as an argument to the echo function. This could be achieved by invoking the command environment from Start/Run/ menu, typing cmd and then executing “echo text” in the console window as illustrated below in Figure 5-1.

Figure 5-1: Example of echo execution in the DOS command environment.

Steps for adding the echo tool:

(1) The tool can be added using the external executor since it is a command line tool. Thus, one would read the XML schema for the external executor first.

(2) Open a registry file and start creating tool tags as illustrated below. The tool name “echo” will be executed by calling the external executable called cmd with arguments “/C echo” followed by the echo parameters described by echoref parameter of type String and value of `="Hello World!"`. In this case, the execution would be equivalent to “cmd /C echo “Hello World!””

```
      <tool name="echo" uuid="external-3">
        <output name="Echo Output" type="text" id="0"/>
            <executor>
                  <external stderr="false">
                  <parameter name="echoref" type="String"
description="Text to echo" value="Hello World!"/>
                        <executable>cmd</executable>
                        <captureStdout refid="0"/>
                        <option>
                              <parameter flag="/C echo" refid =
"echoref"/>
                        </option>
                  </external>
            </executor>
        <help> Echo - Displays messages, or turns command-echoing on or
off. </help>
      </tool>
```

(3) In the Tools menu of Cyber-Integrator, load registry with the added echo tool.

(4) Select and execute the echo tool. A meta-workflow parameter dialog with “Hello World!" will appear, and the value could be modified.

(5) Select the output of the echo tool with the notepad tool to display the string value. The workflow is shown in Figure 5-2.

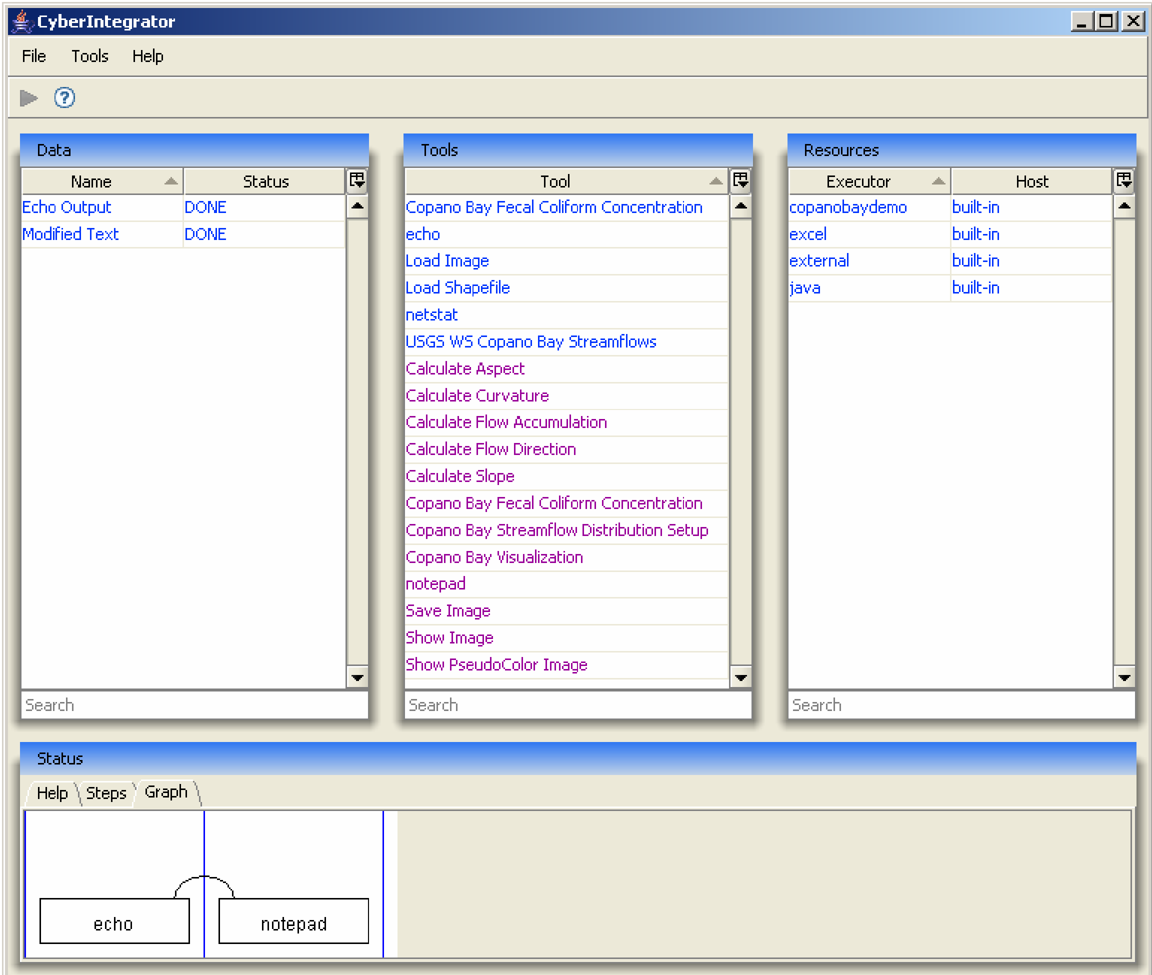

Figure 5-2: Execution of echo command in the DOS command environment followed by notepad visualization of the echo string.

Chapter 6 Advanced Operations: Adding Executors to Cyber-Integrator

This section describes another advanced operation with Cyber-Integrator such as adding a new executor. The purpose of adding a new executor is to support adding and launching a set of tools with minimum programming knowledge while benefiting from the Cyber-Integrator system.

In order to add an executor, one has to become familiar with Java programming to some degree. We provide a set of general principles for adding an executor, a motivation example for adding a new executor and an illustration example to walk through the steps of supporting MS DOS commands by adding a new Cyber-Integrator executor

6.1 General Principles

In general, there are three principles to follow. Create a java class that implements the interface class package ncsa.enviroci.metaworkflow.engine.Executor provided with the software release. The interface is shown below.

```
package ncsa.enviroci.metaworkflow.engine;

import java.util.Collection;
import org.w3c.dom.Element;
import ncsa.enviroci.metaworkflow.*;

/**
 * An executor is a wrapper around a specific engine. Each executor will need to
 * implement these functions. The getName() function will return the name of the
 * executor and will be used to match the executors for each tool.
 *
 * The function initialize() is called when the executor is about to be used to
 * execute a tool. This function is called after a new instance of the executor
 * is created. The function is given the XML code that is used with the tool
 * section. This XML code can be used to initialize the executor and prepare it
 * for the execute command.
 *
 * The execute command is will be used to run a tool on the engine that this
 * executor encapsulates. It is assumed that the executor is set up properly in
 * the initialize function.
 *
 * @author kooper
 * @version $Revision: 1.31 $
```

```
 */
public interface Executor {
    public String getName();

    public void initialize(Element executor) throws Exception;

    public void execute(String execute, Engine engine, MetaWorkFlowStep step,
Collection<MetaWorkFlowParameter> parameters) throws Exception;
}
```

Second, create a directory META-INF/services and place inside a file with the entry specifying the full package of your executor implementation class, for example, `ncsa.enviroci.executor.myExecutor.myDOSCMDExecutor`

After compiling the executor java file with the jar files located in the Cyber-Integrator release downloaded, you will need to create a jar file of your executor. This jar file should contain the class files generated, as well as the META-INF/services directory created in the second step.

To execute the Cyber-Integrator with the newly created executor, place the jar file in the directory where the Cyber-Integrator is installed, and modify the cyberintegrator.bat or cyberintegrator.sh file. You will need to add the jar file with your exector to the classpath used to start the Cyber-Integrator. For example if your jar file is called myexample.jar the bat file will read “java -Xmx512M -classpath "**myexample.jar;**commons-logging.jar;…” where the bold part is the inserted text.

The tools for your executor are kept in a registry (an XML file). To automatically load this new registry, in addition to the existing registries, place your registry file in the directory data/registry inside of the Cyber-Integrator release. Otherwise you can load the registry manually from Tools/Load Registry menu.

6.2 Motivation Example

Let us suppose that one does a lot of work in MS DOS command line environment. A user executes a series of commands in the MS DOS window and would like to create/modify/re-execute scripts consisting of a sequence of commands and forming a workflow. While typically this would be done using a batch file, a Cyber-Integrator’ executor supporting an execution of any MS DOS command would enable a user to benefit from all process management features.

6.3 Illustration Example

Let us design a MS DOS command executor and then execute the command “echo text”.

Introduction

Steps for adding the MS DOS executor and launching the echo tool using this executor:

(1) Implement the Executor interface class. The implementation is provided below with comments

```
package ncsa.enviroci.executor.myExecutor;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Collection;

import ncsa.enviroci.metaworkflow.MetaWorkFlowParameter;
import ncsa.enviroci.metaworkflow.MetaWorkFlowStep;
import ncsa.enviroci.metaworkflow.engine.Engine;
import ncsa.enviroci.metaworkflow.engine.Executor;

import org.w3c.dom.Element;

/**
 * This is an example MS DOS command executor
 *
 * @author pbajcsy
 * @version 1.0
 *
 */

public class myDOSCMDExecutor implements Executor {


    public myDOSCMDExecutor() {
    }

    public String getName(){
      return  "dos-cmd";
    }

    public void initialize(Element executor) throws Exception{
        System.out.println("TEST: initialize myDOSCMDExecutor");
    }

    public void execute(String execute, Engine engine, MetaWorkFlowStep step,
Collection<MetaWorkFlowParameter> parameters) throws Exception{
        // notify listeners we have started execution
        engine.fireExecuteStart(execute, step, parameters);

        // retrieve the tag that will define the MS DOS command
```

```
        Element executor = step.getExecutor();
        String func = executor.getAttribute("command");

        // verify that a command was specified in a tool registry
        if (func == null) {
            throw(new Exception("need a command"));
        }

        // implement what to do a command
        String output = new String();
        if (func.equals("echo")) {
            // retrieve the text parameter labeled as echoref
            MetaWorkFlowParameter param =
MetaWorkFlowParameter.findParameter("echoref", step, parameters);
            // execute the echo command
            Process p = Runtime.getRuntime().exec("cmd /C echo " + param.getValue());
            // retrieve the command output and put it into the output string
            BufferedReader input = new BufferedReader(new
InputStreamReader(p.getInputStream()));
            String line = input.readLine();
            while(line != null){
                output += line;
                line = input.readLine();
            }

        }
        step.getOutput("0").setData(output);

        // notify listeners we have finished execution
        engine.fireExecuteDone(execute, step, parameters);

    }

}
```

(2) Create the META-INF/Services/ ncsa.enviroci.metaworkflow.engine.Executor file and put there one line:

```
ncsa.enviroci.executor.myExecutor.myDOSCMDExecutor
```

(3) Create a registry.xml file:

```
<registry>
      <tool name="myEcho" uuid="dos-cmd-1">
        <output name="Echo Output" type="text" id="0"/>
            <executor>
                  <dos-cmd command="echo">
                  <parameter name="echoref" type="String"
description="Text to echo" value="Hello World!"/>
                  </dos-cmd>
```

```
            </executor>
        <help> Echo - Displays messages, or turns command-echoing on or
off. </help>
      </tool>

</registry>
```

(4) Compile the project and run Cyber-Integrator. The new executor should appear in the right pane and after loading the new registry, the tool can be executed with the notepad viewing of the output as shown in Figure below.

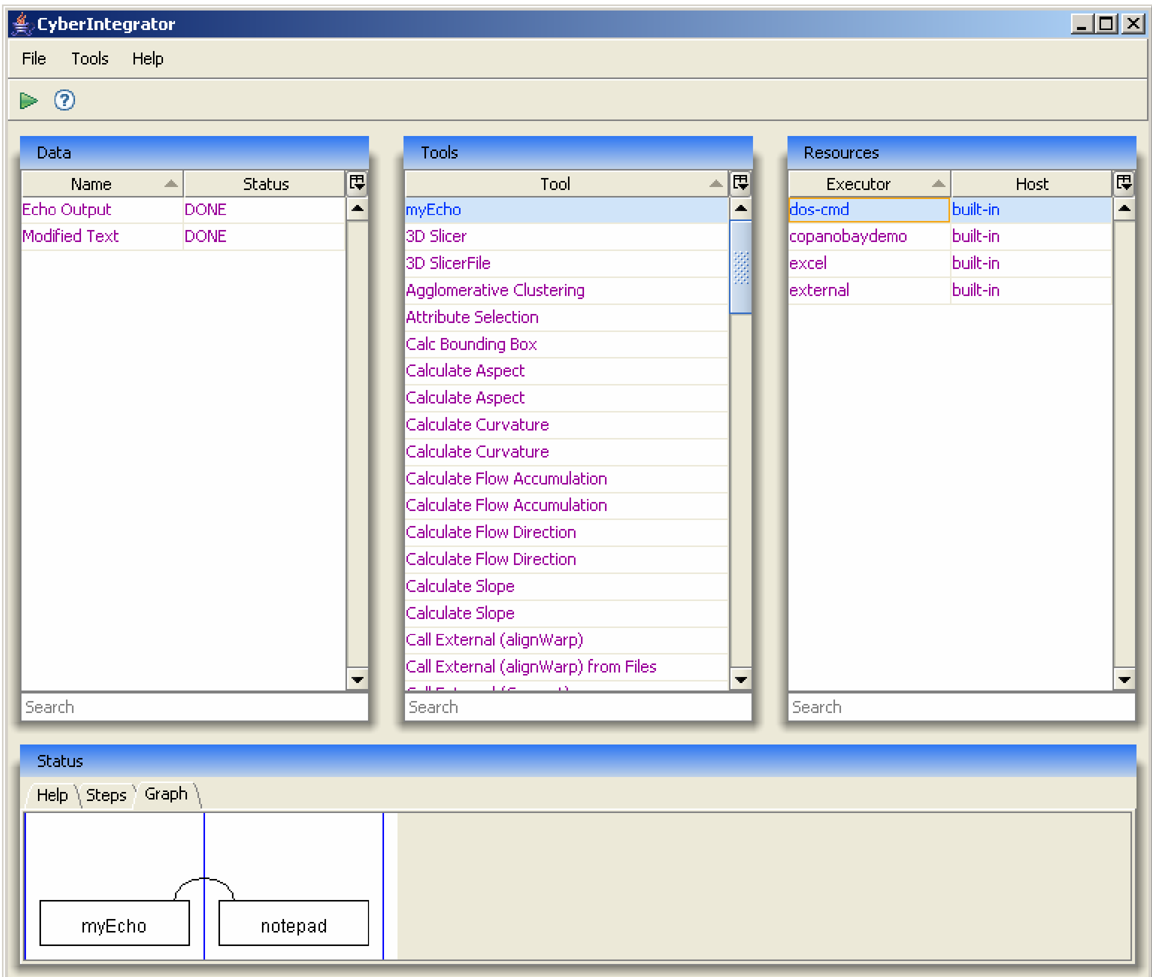

Figure 6-1: Example dos-cmd executor

Appendix A Excluded Remote Functionality

Publishing: Cyber-Integrator has the ability to publish meta-workflows to remote servers. They can then be accessed by others and run remotely or locally.

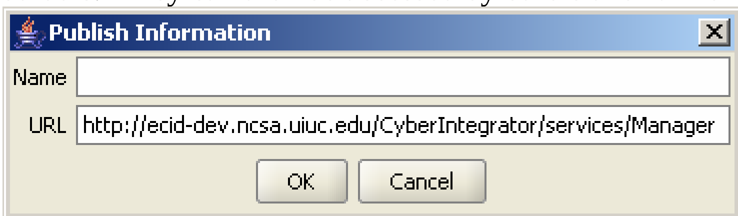

Figure 6-2: Publish dialog

CI-KNOW: Cyber-Integrator has an option of automatically collecting data and processing provenance information and storing the information as Resource Description Framework (RDF) triples in a local or remote meta-data database. The provenance information is used for providing recommendations about the use of tools by calling an external service called CI-KNOW. CI-KNOW recommendation dialog is shown below.

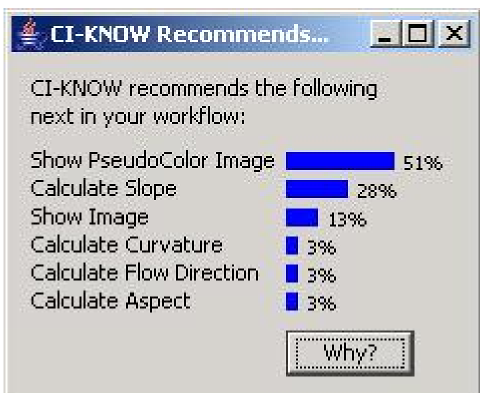

Figure 6-3: CI-Know recommendation dialog after the step "load image".

Remote Registry: Cyber-Integrator can load remote tool registries directly into the application from the tool menu.

Add Executor: It is also possible to add an executor to the resource pane. This process is more involved and is not currently documented. The executors could be local or remote specified by URLs of remote executors. This means that remote resources can be used directly from the desktop application.

Streaming data: Events can be streamed real-time using Java Message Service (JMS). Thus, remote servers receive events in real time and can be programmed to act on certain events. Data input can be continuous as well as discrete.

Login screen: Since users may choose to implement proprietary resources or keep meta-workflow information private, a login screen provides security.

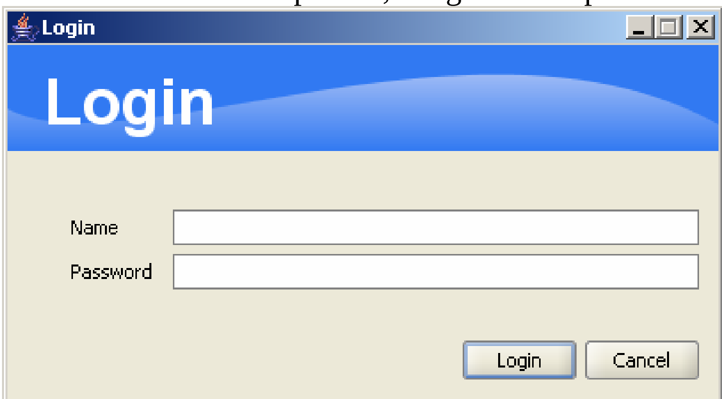

Figure 6-4: Login screen

Cyber-Integrator Server:

The server is the basis for the remote functionality.

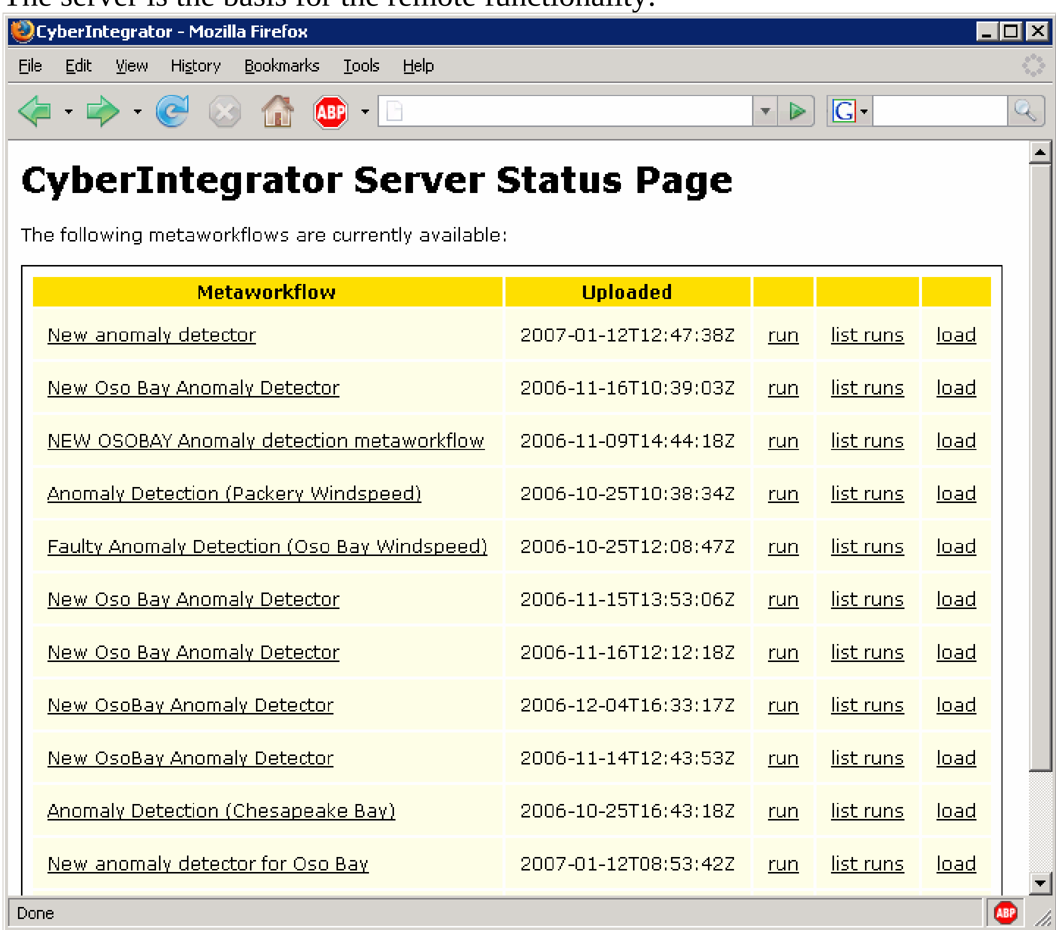

Figure 6-5 Cyber-Integrator Server

Java applet:

The Java applet can be used to execute meta-workflows exclusively on the remote machine.

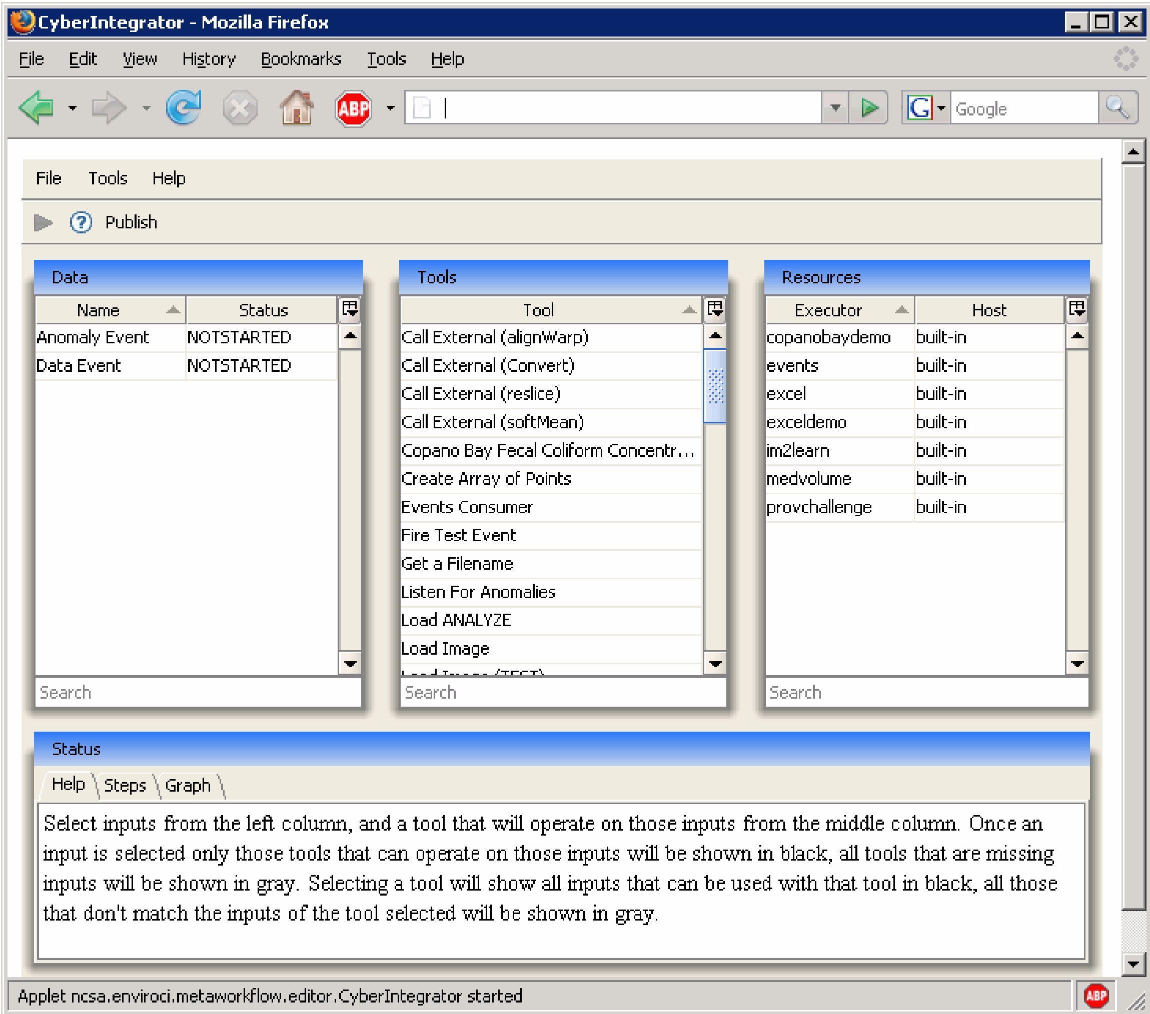

Figure 6-6 Cyber-Integrator Java Applet

Appendix B Optional Application Example

Cyber-Integrator has also been developed with tools for data-driven modeling using remote sensing imagery (exploring causes and consequences of hydrologic variables)

The goal is to construct a workflow for loading multiple raster files (images), integrating images in terms of their spatial resolution and geographic projection, masking image pixels of interest, extracting the values of pixels over the mask, selecting input and output variables for data-driven modeling, obtaining a data-driven model and displaying the results of modeling. This workflow is pretty complex and can be applied to a class of discovery problems when phenomena are not well understood or modeled analytically.

In this application example, the input files are remotely sensed images from NASA and the investigations focus on establishing relationships among the variables represented by the image data. More information about the background of this application example can be found at http://isda.ncsa.uiuc.edu/geolearn/.

Execution Sequence

This application example is using GeoLearn functionality. GeoLearn has to be installed before the execution of this sequence. It can be downloaded from http://isda.ncsa.uiuc.edu/download/. Please be aware that the full execution of GeoLearn requires also installations of other software packages as listed in the GeoLearn user manual (For data-driven modeling of small data sets, D2K must be installed - separate license required for non-educational institutions; ArcGIS – separate license required; MODIS Reprojection Tool (MRT)). Example data sets are also available with the download. We will include this application example in our next release.

Appendix C Software License

6.3.1 Illinois Open Source License

http://www.otm.uiuc.edu/faculty/forms/opensource.asp

University of Illinois/NCSA
Open Source License

Developed by:

Names of Development Groups:
Image Spatial Data Analysis (ISDA) group
http://isda.ncsa.uiuc.edu/

Cyber-Environments and Technologies (CET) Division
http://www.ncsa.uiuc.edu/AboutUs/Directorates/CET.html

Names of Institutions:
National Center for Supercomputing Applications (NCSA)
http://www.ncsa.uiuc.edu/

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal with the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.
- Neither the names of **University of Illinois/NCSA**, nor the names of its contributors may be used to endorse or promote products derived from this Software without specific prior written permission.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,

Software License