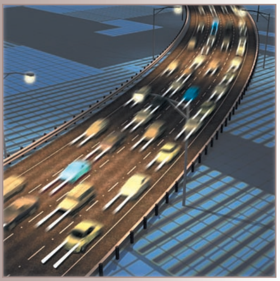


Editors: M. Brian Blake • mb7@georgetown.edu
Michael N. Huhns • huhns@sc.edu

Combining the Power of Taverna and caGrid

Scientific Workflows that Enable Web-Scale Collaboration

Service-oriented architecture represents a promising approach to integrating data and software across different institutional and disciplinary sources, thus facilitating Web-scale collaboration while avoiding the need to convert different data and software to common formats. The US National Cancer Institute's Biomedical Information Grid program seeks to create both a service-oriented infrastructure (caGrid) and a suite of data and analytic services. Workflow tools in caGrid facilitate both the use and creation of services by accelerating service discovery, composition, and orchestration tasks. The authors present caGrid's workflow requirements and explain how they met these requirements by adopting and extending the Taverna system.

Wei Tan, Ian Foster, and Ravi Madduri
University of Chicago

Service-oriented architecture (SOA) promises to evolve the Web from an information hub to a machine-to-machine collaboration platform.[1] In science, where rapid and accurate communication is often vital to progress, adopting SOA approaches could bring about "service-oriented science."[2] Biomedical research is one field that's benefitted from Web-scale collaboration using SOA. (See the "Wrapping Biomedical Data as Services" sidebar.)

The effort to virtualize resources as services in service-oriented science can foster an ecosystem that facilitates scientific investigation in a Web-scale manner. However, a healthy service ecosystem requires more than interoperability: it needs users willing to both use existing services and develop and publish a steady stream of new ones. Thus, we require tools that facilitate service development, publication, discovery, and composition. (We also require reward systems that encourage users to engage in these tasks, but that topic is beyond this article's scope.)

Scientific workflows that compose and orchestrate services are an approach for meeting these two requirements and thus sustain the service ecosystem. First, scientists usually achieve scientific explorations through complex and distributed procedures. If scientific workflow tools help them discover services that meet their needs

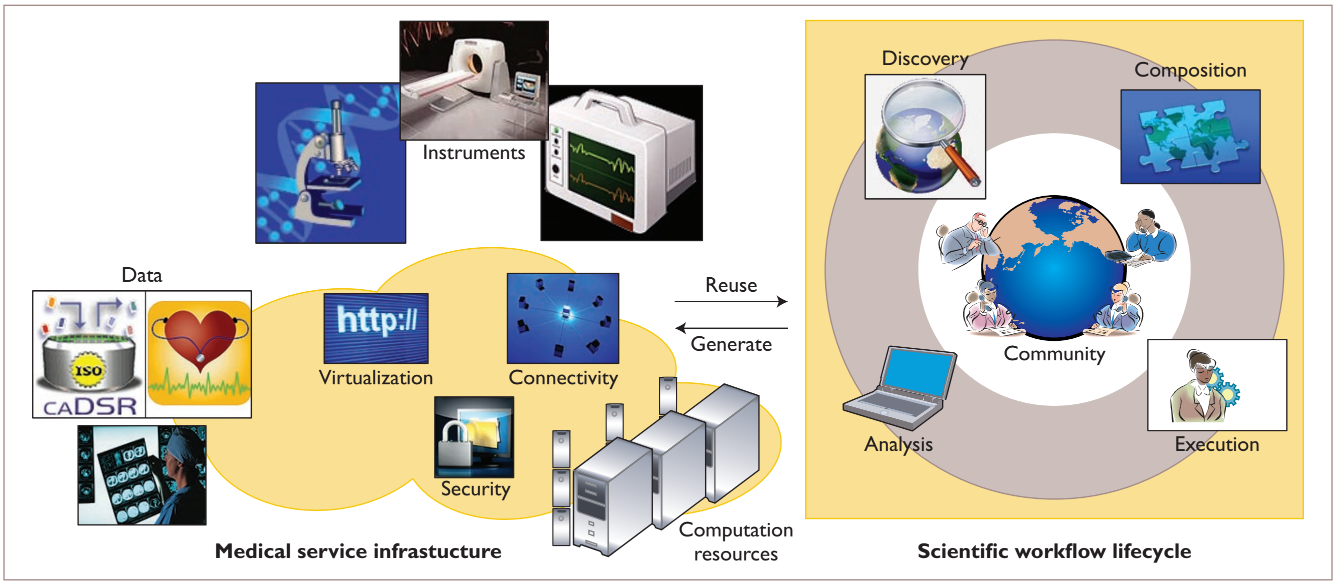


Figure 1. Service ecosystem and scientific workflow in biomedical research. A service ecosystem consists of a medical service infrastructure and data, instruments, and computation resources as services. Scientific workflow involves service discovery, workflow composition, workflow execution, and result analysis.

and compose those services in a desired sequence with a lighter programming burden, they'll be more willing to use this ecosystem. Second, user-created workflows that represent "best practices" for scientific experimentation can also be wrapped as services and published for others to use. This way, users not only consume but also contribute to the system.

The cancer Biomedical Informatics Grid (caBIG) program (https://cabig.nci.nih.gov/) is an example of this service-oriented effort in biomedical research. Sponsored by the US National Cancer Institute (NCI), it was established to let scientists more efficiently share data to accelerate cancer research. caGrid (www.cagrid.org), the service-based grid infrastructure for caBIG, is based on the Globus Toolkit 4.0 grid middleware.[3] The caGrid infrastructure allows users to wrap a wide variety of cancer-research-related data sources and analytical capabilities as services. Nearly 50 cancer centers and 30 other organizations are working collaboratively on caBIG, and caGrid provides more than 68 services. Thus, users must inevitably construct more complex experiment routines by composing multiple services as workflows.

Here, we present the requirements for developing scientific workflows from caGrid and explain how we fulfill these requirements by leveraging and enhancing software tools. The caGrid workflow encounters several challenges, so we selected a tool called Taverna (http://taverna.sourceforge.net/) as our solution. We've also created a caGrid plug-in that extends Taverna with the ability to orchestrate caGrid services.

Wrapping Biomedical Data as Services

In the main text, the left side of Figure 1 shows that not only data resources (such as a DNA database, large molecular entities, medical images, clinical trial data, and ontology data) but also computational functions (protein structural analysis, gene comparison, and data visualization) and even instruments (electron microscopes, oscillographs, CT scanners, and so on) can be wrapped as services that developers or users can then publish, discover, and access uniformly. Besides letting developers create these individual services, a cyberinfrastructure gives users common facilities, such as service virtualization, connectivity, and security. Thus, researchers can obtain data and computation from the Web instead of the lab and can undertake biomedical experiments at a Web scale. A UK-based survey identifies four major data centers and more than 20 smaller ones that collectively provide services implementing more than 3,000 data access and analysis operations (see www.mygrid.org.uk/wiki/Mygrid/BiologicalWebServices). One major data center, the DNA data bank of Japan (www.ddbj.nig.ac.jp/), now holds more than 87 million DNA entries and 91 billion nucleotide elements.

Challenges in Scientific Workflow

Service-oriented computing has brought traditional workflow technology to a Web-scale paradigm.[4] However, new challenges arise when we adopt workflows in scientific exploration, especially in a Web-scale platform such as caGrid. Scientific workflow has been the subject of considerable research, and researchers have developed and validated many systems.[5] A caBIG team performed a comprehensive investigation comparing these systems, languages, and tools (see http://gforge.nci.nih.gov/docman/view.php/332/7509/icr_workflow_tool_review_2007.doc). The team concluded that Taverna does a particularly good job of meeting caGrid's workflow requirements in terms of its full life cycle (see Figure 1). Other existing tools also have many desirable features, but here we'll focus on Taverna.

Taverna is an open source workflow workbench developed in the myGrid project (www.mygrid.org.uk). Its goal is to facilitate the use of workflows and distributed resources within the e-science community. Taverna provides both a workflow-authoring tool that uses a proprietary definition language called Scufl, and an execution engine compliant with this language. It has built-in support for accessing many biological databases and analytical services, such as Biomart (www.biomart.org) and Soaplab (www.ebi.ac.uk/soaplab/). It also provides an extensible framework that lets developers plug in additional applications.

To validate the decision to choose Taverna, let's look at some research challenges that occur in the life cycle of scientific workflows and the Taverna features that align with them. This life cycle has four stages: services discovery, workflow composition, workflow execution, and result analysis (Figure 1, right). We noticed that the service interaction process (discovery, engagement, and enactment) proposed in the Semantic Web Service Architecture (SWSA)[6] is a well-accepted one for Semantic Web services. The terms we use here come more from a scientific workflow perspective but can be aligned with SWSA terms.

During these four stages, scientist can greatly benefit from community experience through which they can share data, services, workflows, and the knowledge obtained by doing experiments. Taverna also has a sister project called myExperiment (www.myexperiment.org) — a Web 2.0 community for workflow sharing. For these reasons, we selected Taverna as the workflow "backbone" in caGrid.

Discovery

Choosing which service to use might be trivial in business workflows because most participant services are either homemade or outsourced via contracts. However, scientific workflows exist in a Web-scale ecosystem in which many organizations provide many services. Users might not know these services' URLs or even their functions and semantics: because the scientific community is too autonomous to share a common terminology, without domain knowledge it's impossible to determine a service's exact semantics using its syntax. A service ecosystem must establish an agreed-on vocabulary and use it to annotate services, so that users can discover them with less ambiguity. Taverna has an extension point (or *scavenger*) to aid customized service discovery. We'll discuss this more in the next section.

Composition

In this step, workflow developers compose individual services into a workflow that describes the experiment scientists will undertake. Developers might add data and control dependencies among services, or they might transform data if one service's output isn't in the format required for another service's input. Developers can perform composition using general-purpose programming languages such as Java, scripting notations such as Swift or DAGMan, or even GUIs more desirable to scientists in a given domain.

Whereas business processes consist of complex control logic, scientific workflows are more focused on parallel data flow. In a data flow, tasks and links represent data processing and data transport, respectively. Tasks without data dependencies between them can execute in parallel. Taverna's modeling style fits in this dataflow-oriented flavor, which the caGrid community welcomes.

Execution

At runtime, a workflow engine invokes services in a predefined order, providing their input and retrieving their output during the ordered execution. An engine for scientific workflows should be aware of the data and computation resources that it can leverage and, ideally, will plan the execution to optimize some performance index, such as execution time or throughput.

Taverna provides a functional model, called *implicit iteration*, to ease parallel execution. At runtime, implicit iteration occurs if a processor receives more inputs than it expects. This capability is useful when workflow designers

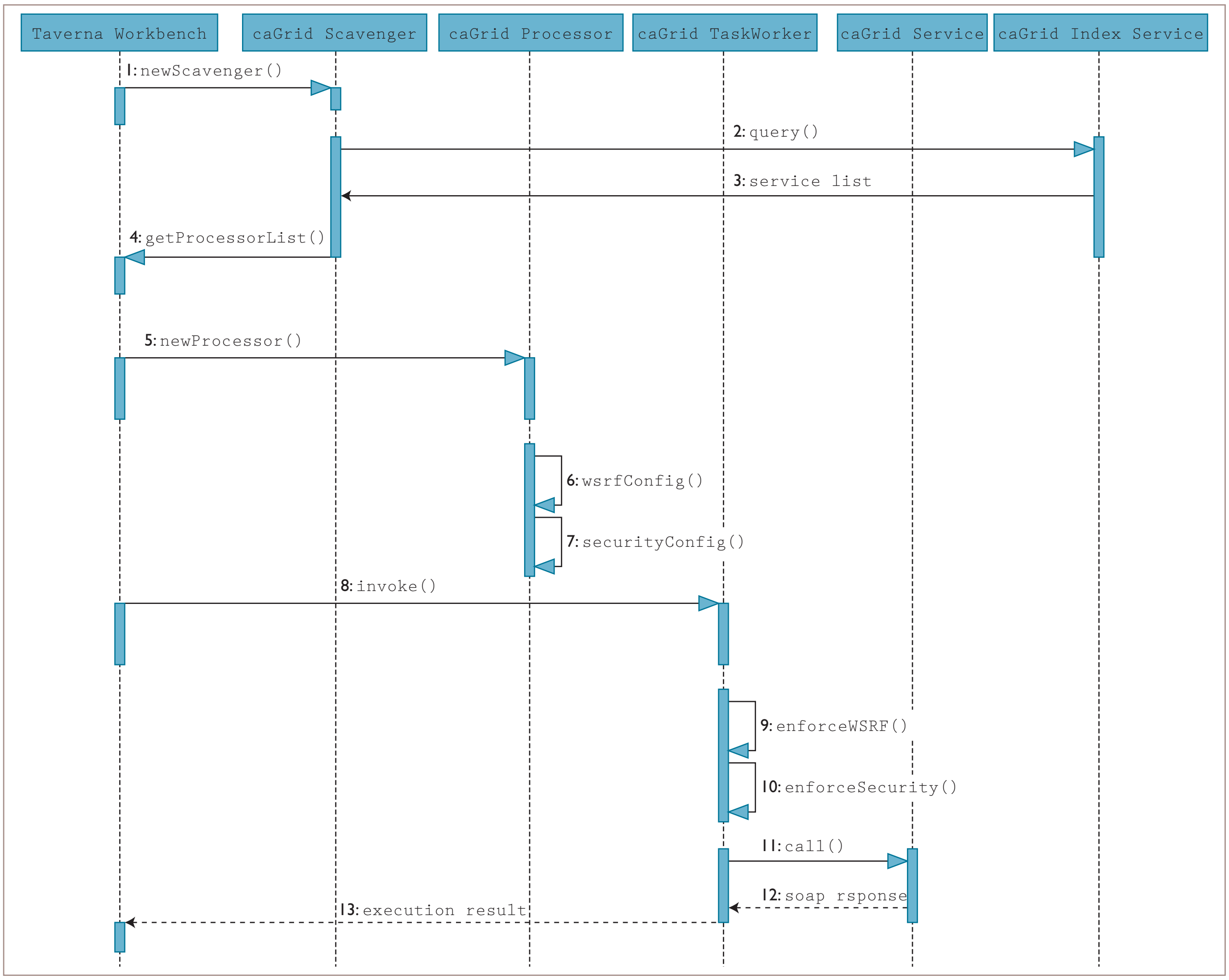


Figure 2. Interaction between Taverna and caGrid plug-in components. The Taverna workbench queries services from Scavenger, configures Globus extensions in the Processor, and invokes services through TaskWorker.

can't estimate inputs' cardinality at build-time, which often happens in caGrid workflows. In contrast, in an imperative language such as the Web Service Business Process Execution Language (WS-BPEL),[7] you'd have to add `<while>` or `<forEach>` constructs explicitly for iteration.

Analysis

Scientists create and execute workflows in support of exploratory research. Component services generate intermediate results, and the workflow yields final results; these results are of great value and should be stored for later analysis. Taverna gives each intermediate and final data item a unique identifier and stores them properly. Its extension framework lets users add customized functions such as data visualization, tracking, and querying.

Scientific Workflow in caGrid

Globus-based caGrid services are Web services invoked by SOAP, with WSDL-defined interfaces. Globus implements two sets of Web services features that are particularly important for Web-scale computing: access to stateful resources and secure access.

In scientific computing, users want to access and manipulate state in service interactions. For example, scientists might submit a job to a scheduler and want to query the state of this specific job instance or get state notification when the job completes (note that the scheduler contains multiple job instances). The Web Service Resource Framework (WSRF)[8] specification lets service clients access stateful resources. A resource-generation operation (job submission, for example) creates a new resource instance

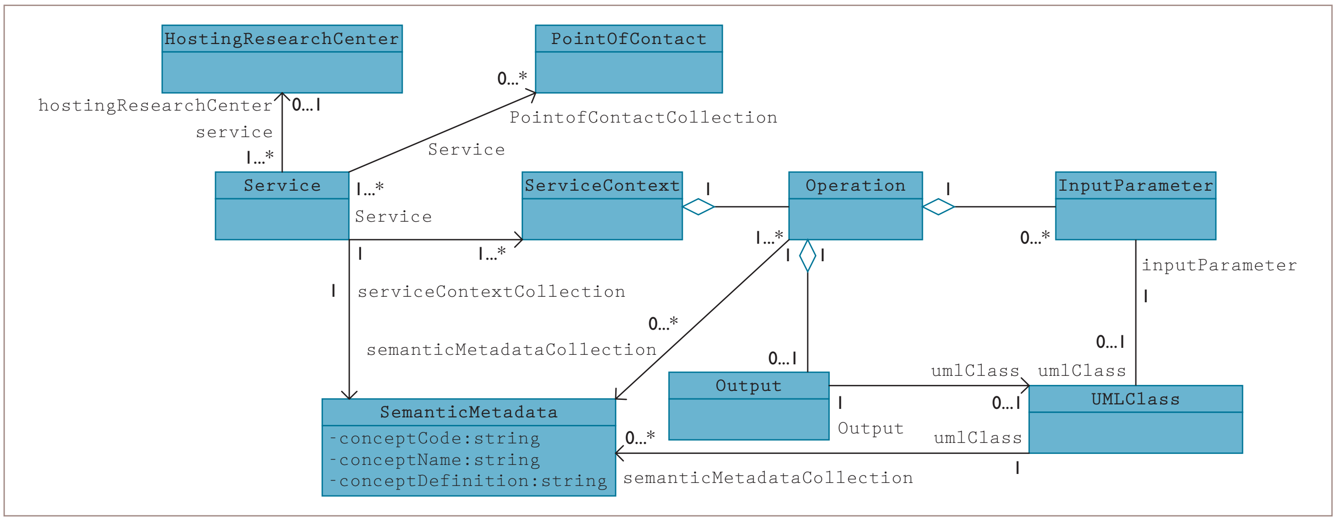


Figure 3. caGrid service metadata for service discovery. Users can query caGrid services using these metadata.

and returns an element called `ReferenceProperties` that identifies this instance. Resource-access operations (such as job state query) can simply launch a SOAP invocation as normal and insert the `ReferenceProperties` into the SOAP header to access a specific resource instance, or query resource states (called *resource properties* in WSRF).

In Web-scale scientific collaboration, secure communication is important for protecting resources across organizational boundaries. The Grid Security Infrastructure (GSI)[9] is a set of components in the Globus Toolkit that provides security features. It uses public-key cryptography to offer functions such as secure encryption, authentication (identifying the caller/sender), authorization (checking access rights), and delegation (performing a task on a delegator's behalf).

Our caGrid plug-in for Taverna had certain requirements. It must

- make Taverna aware of caGrid services, so that caGrid users can choose available services they'll later orchestrate using Taverna;
- configure services at build-time with caGrid properties, such as a resource identifier and security, if needed; and
- receive instruction from the Taverna engine at runtime to invoke caGrid services with a possible Globus extension.

We extended three interfaces in Taverna — the `Scavenger`, `Processor`, and `TaskWorker` — to integrate it into caGrid. As we mentioned earlier, a scavenger contains a customized services query and the query's result — that is, a collection of processors. A processor contains the build-time information for a single step inside a Taverna workflow. Each processor corresponds to a `TaskWorker` interface, which defines the action the processor will perform at runtime. Figure 2 shows interactions between Taverna and the components in the caGrid plug-in, which we'll now look at in some detail.

Metadata-Based Service Query

In step 1 (see Figure 2), a user first initiates a service query through the Taverna workbench (`newScavenger()`). The caGrid Scavenger performs this query against service metadata stored as resource properties of the caGrid index service, which leverages the caGrid service metadata definition that Figure 3 illustrates. The `HostingResearchCenter` hosts a service, which might have several `PointsOfContact`. Its `ServiceContext` contains a list of `Operations`; each operation has a list of `InputParameters` and an `Output`, both of whose data types can be defined via `UMLClasses`. caGrid service metadata annotates `Service`, `Operation`, and `UMLClass` with `SemanticMetadata`. caGrid has defined a common vocabulary for `UMLClass` and `SemanticMetadata`, which users can leverage for semantic-based querying.

The caGrid plug-in provides three querying methods:

- *string based*, which lets users locate services related to "pathology," for example;
- *property based*, which, for instance, locates services hosted by "Ohio State University"

with the name `DICOMDataService` and the operation `PullOp`; and
- *semantic based*, which locates services, for example, that are semantically annotated as `Core Grid Service`.

We can combine these three methods to form complex queries. Looking at Figure 2, we can see that the caGrid `Scavenger` turns user queries into XPath clauses and sends them to a caGrid index to get a list of services (`query()` and `service list`). The workbench displays this service list (`getProcessorList()`) for users to choose.

caGrid Configuration: State Management and Security

After the service query, Taverna populates the workbench with a list of processors that users can add to build a new workflow (step 5 in Figure 2, `newProcessor()`). For Globus-specific extensions, (step 6, or `wsrfConfig()`), users configure `ReferenceProperties` for processors by linking resource-generation processors to resource-access processors. In step 7, `securityConfig()`, users configure security for processors, specifying, for example, which certificate and which encryption method to use.

caGrid Service Invocation

At runtime, when the workflow execution reaches a certain processor, Taverna creates a `TaskWorker` class that fetches the input data from the workbench and wraps them into a SOAP message body. After that, `TaskWorker` inspects the WSRF and security configuration the processor created and enforces these configurations by changing the SOAP body and the header accordingly (`enforceWSRF()` and `enforceSecurity()`). Then, `TaskWorker` sends the SOAP message out for service invocation (`call()`) and obtains a response (`soap response`). The workbench displays the returned results (`execution result`).

You can download the caGrid plug-in from www-unix.mcs.anl.gov/~madduri/taverna/ and install it in Taverna (1.7.1.0 or a more recent release). We've tested it with several caGrid workflows, one of which we describe next.

Microarray Clustering Workflow: A Case Study

To illustrate our caGrid plug-in's application, we tested it with a microarray hierarchical clustering workflow that involves services hosted at multiple institutions.

Microarrays are a high-throughput technology used to measure the expression of tens of thousands of genes in different tissues or cells. Scientists represent the data from each microarray via a vector (profile) in which each element represents a gene's expression level. They use clustering analysis to identify similar expression profiles across genes or samples.[10] In particular, hierarchical clustering is popular for grouping microarrays into a multilevel hierarchy in which, at each level, arrays in the same cluster are more similar to each other than those in different clusters. To cluster data, the user must identify and retrieve relevant microarrays, preprocess them, and then invoke the hierarchical clustering program. In the past, we might have programmed this sequence of steps using a scripting language such as Perl. Instead, we use Taverna and the caGrid plug-in to identify relevant services, compose those services with additional building blocks (for data transformation), and orchestrate their execution. Our workflow involves three major steps:

1. Identify and retrieve the microarray data of interest. We used CQL, the query language that caGrid Data Services uses, to specify this data and retrieve it from a caArray data service hosted at Columbia University (http://cagridnode.c2b2.columbia.edu:8080/wsrf/services/cagrid/ CaArrayScrub).
2. Preprocess, or normalize, the microarray data before clustering them. We used a GenePattern analytical service (http://node255.broad.mit.edu:6060/wsrf/services/cagrid/Preprocess Dataset MAGEService), which provides normalization, floor and ceiling thresholding, variation filtering, and other preprocessing functions. We used an instance of this service hosted at MIT's Broad Institute.
3. Run hierarchical clustering on the preprocessed data. We invoked the geWorkbench analytical service Columbia University hosts (http://cagridnode.c2b2.columbia.edu:8080/wsrf/services/cagrid/Hierarchical ClusteringMage).

Figure 4 illustrates the Taverna workflow. It contains an input processor to store the CQL expression, an output processor to store the clustered microarray data (both input and output

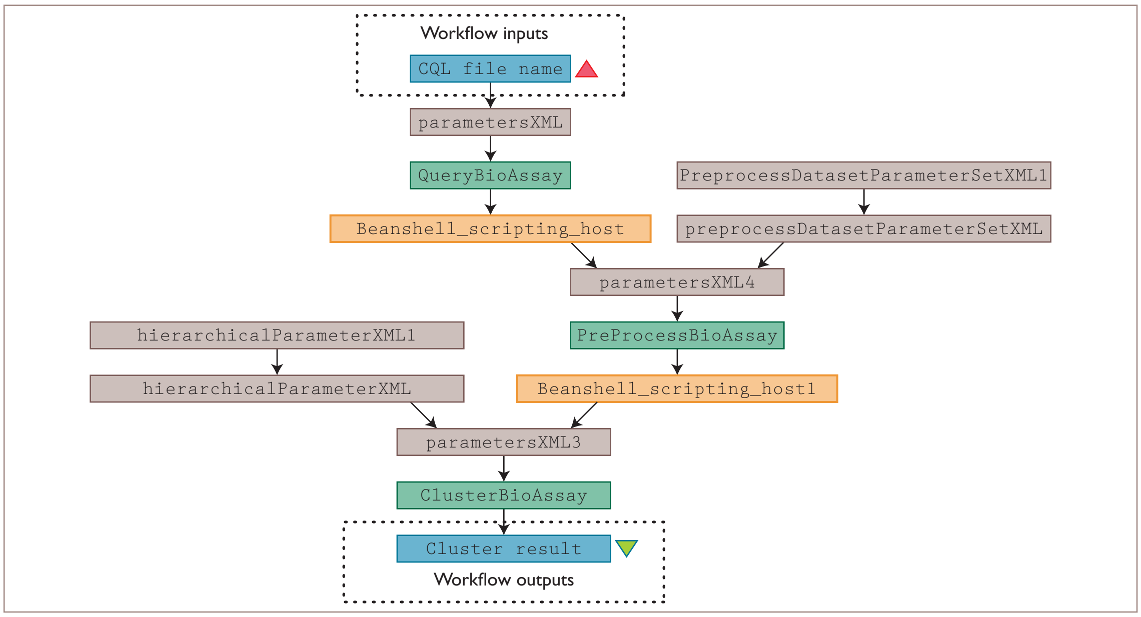


Figure 4. The microarray clustering workflow. It contains I/O processors, caGrid processors, and a few "shim" processors to deal with data transformation between services.

processors are blue), three caGrid processors (green) representing the three caGrid services just listed, and a few "shim" processors, such as XML splitters and beanshell scripts, to deal with data transformation between services.

Once we've created our workflow, we can use Taverna to execute it, monitor its execution, and examine both its output and the data it produces at intermediate steps. Figure 5 shows the execution trace (bottom left) and the execution result (right), which, in this case, is an XML document representing the hierarchically clustered arrays.

Compared to our approach, traditional approaches based on, for example, Perl scripting are more time-consuming and error-prone. Moreover, they can't handle Web-scale specific issues, such as services that aren't visible or understandable to users or services that are volatile. In contrast, with our plug-in, we can discover caGrid services using domain metadata, compose, execute, and monitor workflows in a visualized manner. Overall, we believe that Taverna offers many desirable features with little programming effort.

The work we've described here is a first step toward providing a comprehensive and easy-to-use workflow suite that extends scientific investigations to Web scale. Future work, which we're pursuing with the Taverna team, includes providing more comprehensive security support and optimizing dataflow execution. For example, we plan to improve performance for large data sets using tools such as GridFTP to directly transfer data between processors instead of through the workflow engine.

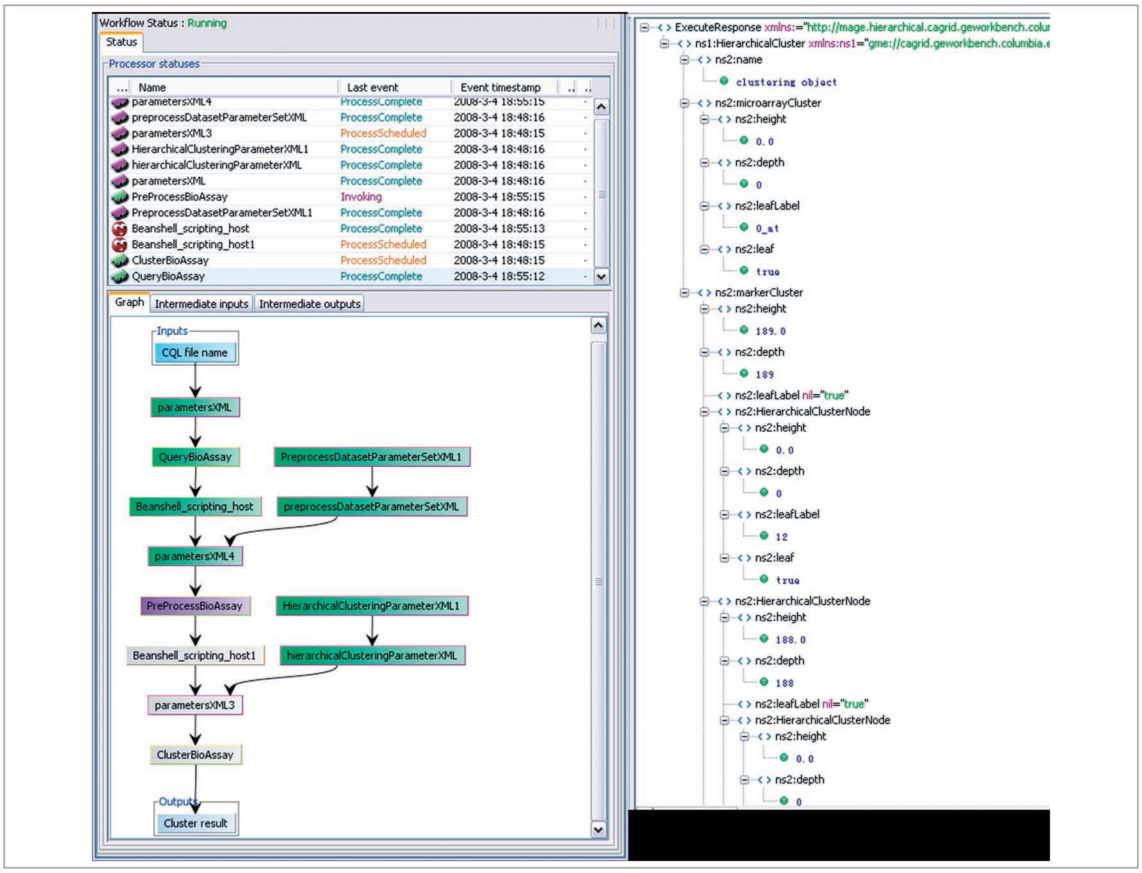
We believe that the solution and experience we present here are applicable to other application scenarios in related disciplines. We're working to apply the same methods to other domains of experimental and simulation science.

Acknowledgments

We thank Stian Soiland-Reyes for his great help in using Taverna and developing the plug-in, and Paolo Missier for the constructive discussion on Taverna's modeling style. We also thank the caGrid team for their help in various use cases. This project has been funded in whole or in part with federal funds from the US National Cancer Institute, National Institutes of Health, under contract no. N01-CO-12400.

References

1. M.N. Huhns and M.P. Singh, "Service-Oriented Computing: Key Concepts and Principles," *IEEE Internet Computing*, vol. 9, no. 1, 2005, pp. 75–81.
2. I. Foster, "Service-Oriented Science," *Science*, vol. 308, no. 5723, 2005, pp. 814–817.
3. I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *J. Computer Science and Technology*, vol. 21, no. 4, 2006, pp. 513–520.
4. M.B. Blake and M.N. Huhns, "Web-Scale Workflow: Integrating Distributed Services," *IEEE Internet Comput-*


Workflow Status : Running

Status

Processor statuses

...	Name	Last event	Event timestamp	..	..
	parametersXML4	ProcessComplete	2008-3-4 18:55:15		·
	preprocessDatasetParameterSetXML	ProcessComplete	2008-3-4 18:48:16		·
	parametersXML3	ProcessScheduled	2008-3-4 18:48:15		·
	HierarchicalClusteringParameterXML1	ProcessComplete	2008-3-4 18:48:16		·
	hierarchicalClusteringParameterXML	ProcessComplete	2008-3-4 18:48:16		·
	parametersXML	ProcessComplete	2008-3-4 18:48:16		·
	PreProcessBioAssay	Invoking	2008-3-4 18:55:15		·
	PreprocessDatasetParameterSetXML1	ProcessComplete	2008-3-4 18:48:16		·
	Beanshell_scripting_host	ProcessComplete	2008-3-4 18:55:13		·
	Beanshell_scripting_host1	ProcessScheduled	2008-3-4 18:48:15		·
	ClusterBioAssay	ProcessScheduled	2008-3-4 18:48:15		·
	QueryBioAssay	ProcessComplete	2008-3-4 18:55:12		·

```
< > ExecuteResponse xmlns:="http://mage.hierarchical.cagrid.geworkbench.colur
  < > ns1:HierarchicalCluster xmlns:ns1="gme://cagrid.geworkbench.columbia.e
    < > ns2:name
        clustering object
    < > ns2:microarrayCluster
      < > ns2:height
          0.0
      < > ns2:depth
          0
      < > ns2:leafLabel
          0_at
      < > ns2:leaf
          true
    < > ns2:markerCluster
      < > ns2:height
          189.0
      < > ns2:depth
          189
      < > ns2:leafLabel nil="true"
      < > ns2:HierarchicalClusterNode
        < > ns2:height
            0.0
        < > ns2:depth
            0
        < > ns2:leafLabel
            12
        < > ns2:leaf
            true
      < > ns2:HierarchicalClusterNode
        < > ns2:height
            188.0
        < > ns2:depth
            188
        < > ns2:leafLabel nil="true"
        < > ns2:HierarchicalClusterNode
          < > ns2:height
              0.0
          < > ns2:depth
              0
```

Figure 5. Execution trace and results for the microarray clustering workflow. In this case, Taverna created an XML document representing the hierarchically clustered arrays.

ing, vol. 12, no. 1, 2008, pp. 55–59.

5. W. Tan et al., "Workflow in a Service Oriented Cyberinfrastructure Environment," to appear in *Cyberinfrastructure Technologies and Applications*, J. Cao, ed., Nova Science Publishers, 2008.
6. M. Burstein et al., "A Semantic Web Services Architecture," *IEEE Internet Computing*, vol. 9, no. 5, 2005, pp. 72–81.
7. Oasis, "Web Services Business Process Execution Language Version 2.0," 2007; http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html.
8. I. Foster et al., "Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF," *Proc. IEEE,* vol. 93, no. 3, 2005, pp. 604–612.
9. V. Welch et al., "Security for Grid Services," *Proc. 12th IEEE Int'l Symp. High Performance Distributed Computing*, IEEE CS Press, 2003, pp. 48–57.
10. M.B. Eisen et al., "Cluster Analysis and Display of Genome-Wide Expression Patterns," *Proc. Nat'l Academy of Sciences USA,* vol. 95, no. 25, 1998, pp. 14863–14868.

Wei Tan is a research staff member in the Computation Institute at the University of Chicago and Argonne National Laboratory. His research interests include scientific workflow, service-oriented computing, and Petri nets. Tan has a PhD in automation science and engineering from Tsinghua University, China. Contact him at wtan@mcs.anl.gov.

Ian Foster is director of the Computation Institute at the University of Chicago and Argonne National Laboratory and the Arthur Holly Compton Distinguished Service Professor of computer science at the University of Chicago. His research interests include distributed computing, parallel computing, and computational science. Foster has a PhD in computer science from Imperial College, London. Contact him at foster@anl.gov.

Ravi Madduri is a senior software developer with the mathematics and computer science division at Argonne National Laboratory. He's been worked on data management and resource management components of the Globus Toolkit for the past seven years. He's also a lead architect of the workflow technologies with the caBIG project. Madduri has a master's degree in computer science from the Illinois Institute of Technology, Chicago. Contact him at madduri@mcs.anl.gov.