19

Taverna/^{my}Grid: Aligning a Workflow System with the Life Sciences Community

Tom Oinn, Peter Li, Douglas B. Kell, Carole Goble, Antoon Goderis, Mark Greenwood, Duncan Hull, Robert Stevens, Daniele Turi, and Jun Zhao

19.1 Introduction

Bioinformatics is a discipline that uses computational and mathematical techniques to store, manage, and analyze biological data in order to answer biological questions. Bioinformatics has over 850 databases [154] and numerous tools that work over those databases and local data to produce even more data themselves. In order to perform an analysis, a bioinformatician uses one or more of these resources to gather, filter, and transform data to answer a question. Thus, bioinformatics is an *in silico* science.

The traditional bioinformatics technique of cutting and pasting between Web pages can be effective, but it is neither scalable nor does it support scientific best practice, such as record keeping. In addition, as such methods are scaled up, slips and omissions are more likely to occur. A final human factor is the tedium of such repetitive tasks [397].

Doing these tasks programmatically is an obvious solution, especially for the repetitive nature of the tasks. Some bioinformaticians have the programming skills to wrap these distributed resources. Such solutions are, however, not easy to disseminate, adapt, and verify. Moreover, one of the consequences of the autonomy of bioinformatics service providers is massive heterogeneity within those resources. The advent of Web services has brought about a major change in the availability of bioinformatics resources from Web pages and command-line programs to Web services [395], though much of the structural, value-based, and syntactic heterogeneity remains. The consequent lack of a common type system means that services are difficult to join together programmatically, and any technical solution to *in silico* experiments in biology has to address this issue.

Many scientific computing projects within the academic community have turned to workflows as a means of orchestrating complex tasks (*in silico* experiments) over a distributed set of resources. Examples include DiscoveryNet [373] for molecular biology and environmental data analysis, SEEK for ecology [19, 20], GriPhyn for particle physics [110], and SCEC/IT for earthquake analysis and prediction [236].

Workflows offer a high-level alternative for encoding bioinformatics *in silico* experiments. The high-level nature of the encoding means a broader community can create templates for *in silico* experiments. They are also easier to adapt or repurpose by substitution or extension. Finally, workflows are less of a black box than a script or traditional program; the experimental protocol captured in the workflow is displayed in such a way that a user can see the components, their order, and inputs and outputs. Such a workflow can be seen in Figure 19.1.

^{my}Grid is a project to build middleware to support workflow-based *in silico* experiments in biology. Funded by the United Kingdom's e-Science Programme since 2001, it has developed a set of open-source components that can be used independently and together. These include a service directory [267], ontology-driven search tools over semantic descriptions of external resources and data [267], data repositories and semantically driven metadata stores for recording the provenance of a workflow and the experimental life cycle [494], and other components, such as distributed query processing [16] and event notification.[1]

^{my}Grid's workflow execution and development environment, Taverna, links together and executes external remote or local, private or public, third-party or home-grown, heterogeneous open services (applications, databases, etc.). The Freefluo workflow enactment engine[2] enacts the workflows. The Taverna workbench is a GUI-based application for bioinformaticians to assemble, adapt, and run workflows and manage the generated data and metadata. ^{my}Grid components are Taverna plug-ins (for results collection and browsing, provenance capture, service publication, and discovery) and services (such as specialist text mining). Thus the workbench is the user-facing application for the ^{my}Grid middleware services. At the time of writing, Taverna 1.3 has been downloaded over 14,000 times[3] and has an estimated user base of around 1500 installations. Taverna has been used in many different areas of research throughout Europe and the United States for functional genomics, systems biology, protein structure analysis, image processing, chemoinformatics, and simulation coordination. Since 2006, ^{my}Grid has been incorporated into the United Kingdom's Open Middleware Infrastructure Institute to be "hardened" and developed to continue to support life scientists.

19.1.1 A Bioinformatics Case Study

An exemplar Taverna workflow currently being used for systems biology is shown in Figure 19.1. This workflow uses data stored in distributed databases

[1] `http://www.mygrid.org.uk.`
[2] `http://freefluo.sourceforge.net.`
[3] See `http://taverna.sourceforge.net/index.php?doc=stats.php.`

to automate the reconstruction of biological pathways that represent the relationships between biological entities such as genes, proteins, and metabolites.

The interaction pathways generated by the workflow are in the form of a data model, which is specified by the XML-based Systems Biology Markup Language (SBML) [201]. A core SBML workflow is responsible for generating an SBML model. This is then populated, through the SBML API, by the supplementary workflows that gather data for the model (see Figure 19.1). The SBML model can then be used to perform biological simulations.

These workflows typify the needs of bioinformatics analyses. It is a typically datacentric workflow, gathering many kinds of data from a variety of locations and from services of a variety of technology types. As will be seen throughout the chapter, many types of resources are used, and all of these can be incorporated into Taverna. The workflows have to be run repeatedly, and such an analysis would be long and tedious to perform manually.

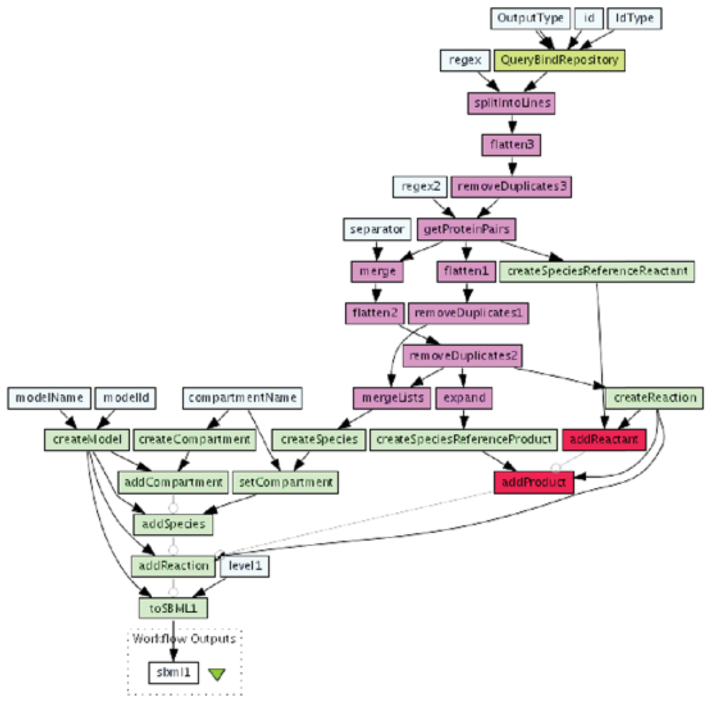


Figure 19.1: An SBML model construction workflow. This workflow retrieves protein interactions from the BIND database, which are then used to populate an SBML model using the core SBML workflow. Four types of processors are used in this example: WSDL, consumer API, local Java, and nested workflow processors. These processors are joined together by data links (arrows) and coordination links.

The rest of this chapter is organized as follows. Section 19.2 further elaborates on the background to Taverna and then Section 19.3 outlines requirements in detail. Section 19.4 introduces the major Taverna components, and architecture. Section 19.5 concentrates on the workflow design and Section 19.6 on executing and monitoring workflows. Section 19.7 completes the workflow life cycle with metadata and provenance associated with managing and sharing results and the workflows themselves. Section 19.8 discusses related work and Section 19.9 reflects on our experiences and showcases future developments in Taverna 2.0.

19.2 The Bioinformatics Background

Life scientists are accustomed to making use of a wide variety of Web-based resources. However, building applications that integrate resources with interfaces designed for humans is difficult and error-prone [395]. The emergence of Web services [58], along with the availability of suitable tool support, has seen a significant number of bioinformatics Web resources become publicly available and described with a Web Services Description Language (WSDL) interface.

There are currently over 3000 services accessible to a ^{my}Grid user. Although the majority involve complex interaction patterns or specific messaging formats, or use different protocols and paradigms, they actually follow a small number of stereotyped patterns. The users' lack of middleware knowledge means they should not be expected to deal with the differences between these patterns. In addition, given the number and distribution of services, users cannot be expected to have existing knowledge of what services are available, where they are, or what they do.

The data produced by these services are mostly semistructured and heterogeneous. There are a large number of data formats, including those for gene sequences and protein sequences, as well as bespoke formats produced by many analysis tools. These are rarely encoded in XML, and there is usually no formal specification that describes these formats. Interpreting or reconciling these data as they are passed between different databases and analysis tools is therefore difficult.

This situation is in contrast with data in other scientific workflow projects that have much more centralized control of data formats. For example, the SEEK project provides tools for ecologists to describe their data using XML schema and ontologies and so support middleware-driven data integration [59].

DiscoveryNet [373] requires each application service to be wrapped, allowing data to adhere to a common format. Other projects are more uniform than ^{my}Grid in the way applications on distributed resources are accessed. For example, abstract Pegasus workflows used in the SCEC/IT project are first compiled into concrete workflows. Each step of a concrete workflow corresponds to a job to be scheduled on a Condor cluster [111].

Taverna differs from these projects by placing an emphasis on coping with an environment of autonomous service providers and a corresponding "open world" model for the underlying Grid and service-oriented architecture. Taverna's target audience of life scientists wants easy access and composition of as wide a range of services as feasible, and this reinforces the need for an open access policy for services, despite the obvious difficulties.

19.3 Aligning with Life Science

From the background and introduction, we can define the key requirements for the Taverna workflow system that drive us to align with life science:

- *Ease of use*. The target end users for Taverna are not necessarily expert programmers.
- *Dataflow centric*. Bioinformaticians are familiar with the notion of dataflow centric analysis. We want to enhance how biologists perform their models of analysis, not change their model of analysis.
- *Open world assumption*. We want to be able to use any service as presented rather than require service providers to implement services in a prescribed manner and thus create a barrier to adoption.
- *Easy and rapid user-driven ad hoc workflow design*. Quickly and easily finding services and adapting previous workflows is key to effective workflow prototyping.
- *Fault tolerant*. Any software operating in a networked, distributed environment is required to cope gracefully with failure.

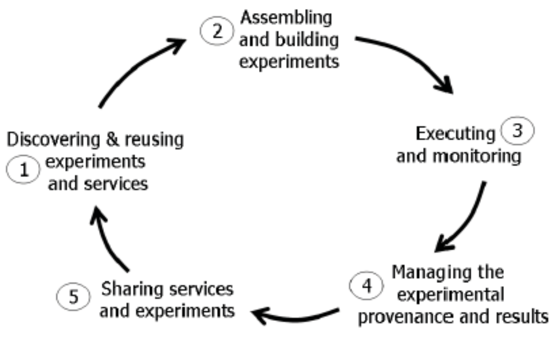

2 Assembling and building experiments
3 Executing and monitoring
4 Managing the experimental provenance and results
5 Sharing services and experiments
1 Discovering & reusing experiments and services

Figure 19.2: The e-Science life cycle.

- *Support for the e-Science life cycle.* Workflows are not a complete solution for supporting *in silico* experiments. They exist in a wider context of scientific data management, as illustrated in Figure 19.2. It is essential that data produced by a workflow carry some record of how and why they were produced, i.e., the provenance of the data.

19.4 Architecture of Taverna

The requirements described have led to several major design lessons. Figure 19.3 illustrates how Taverna takes a layered approach to its overall architecture. This is driven by the need to present a useful, high-level presentation in which biologists can coordinate a variety of resources. Our user base neither knows nor cares about such things as port types, etc. We have a requirement both to present a straightforward perspective to our users and yet cope with the heterogeneous interfaces of our services. A major consequence of this for the workflow system architecture has been to provide a multitiered approach to resource discovery and execution that separates application and user concerns from operational and middleware concerns.

Scufl, a workflow language for linking applications [326], is at the abstraction level of the user; an extensible processor plug-in architecture for the Freefluo enactor manages the low-level "plumbing" invocation complexity of different families of services. In between lies an execution layer interpreting the Taverna Data Object Model that handles user-implied control flows such as implicit iteration over lists and a user's fault-tolerance policies.

Figure 19.3 shows how the ^{my}Grid components are divided between the three layers of ^{my}Grid's design.

- The Application Data Flow layer is aimed at the user and is characterized by a User-Level workflow object model. The purpose is to present the workflows from a problem-oriented view, hiding the complexity of the interoperation of the services. When combining services into workflows, users think in terms of (see Figure 19.4) the data consumed and produced by logical services and connecting them together. They are not interested in the implementation styles of the services.
- The Execution Flow layer relieves the user of most of the details of the execution flow of the workflow and expands on control-flow assumptions that tend to be made by users. This layer is characterized by the Enactor Internal Object Model and by the ^{my}Grid Contextual Information Model. The layer manages list and tree data structures, implicitly iterates over collections of inputs, and implements fault recovery strategies on behalf of the user. This saves the user explicitly handling these at the application layer and avoids mixing the mechanics of the workflow with its conceptual purpose. A drawback is that an expert bioinformatician needs

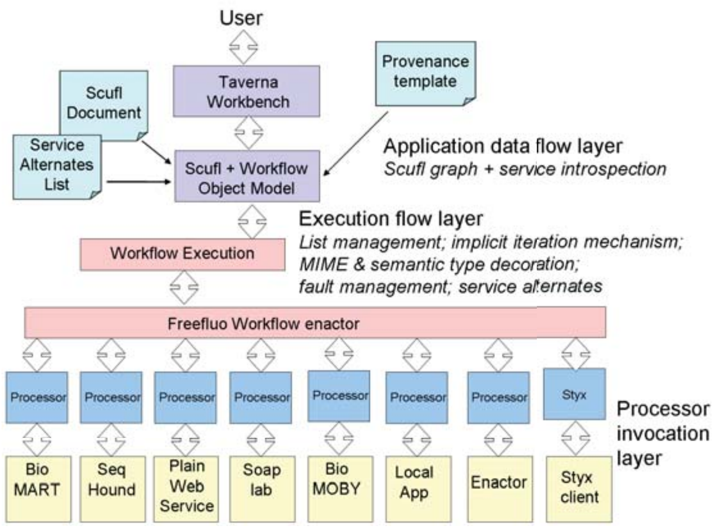

User
Taverna Workbench
Provenance template
Scufl Document
Service Alternates List
Scufl + Workflow Object Model
Application data flow layer
Scufl graph + service introspection
Execution flow layer
List management; implicit iteration mechanism; MIME & semantic type decoration; fault management; service alternates
Workflow Execution
Freefluo Workflow enactor
Processor
Styx
Processor invocation layer
Bio MART
Seq Hound
Plain Web Service
Soap lab
Bio MOBY
Local App
Enactor
Styx client

Figure 19.3: An overview of Taverna in layers.

to understand the behavioral semantics of this layer to avoid duplicating the implicit behavior.

- The Processor Invocation layer is aimed at interacting with and invoking concrete services. Bioinformatics services developed by autonomous groups can be implemented in a variety of different styles even when they are similar logical services from a scientist's perspective. This layer is characterized by the Enactor Internal Object Model and is catered to by an extensible processor plug-in architecture for the Freefluo enactment engine.

$^{\text{my}}$Grid is designed to have a framework that can be extended at three levels:

- The first level provides a plug-in framework to add new GUI panels to facilitate user interaction for deriving and managing the behavioral extensions incorporated into Taverna. This extensibility is made available at the workbench layer.
- The second level allows for new processor types to be plugged in to enable the enactment engine to recognize and invoke new types of services (which can be both local and external services). This permits a wider variety of workflows to be constructed and executed. This level of extensibility is provided at the workflow execution layer.
- The third level is provided for loosely integrating external components via an event–observer interface. The workflow enactor generates events during critical state changes as it executes the workflow, exposing snapshots of important parts of its internal state via event objects (i.e., messages). Those event objects are then intercepted and processed by observer plug-

ins that can interact with external services. This level of extensibility is made available at the workflow execution layer.

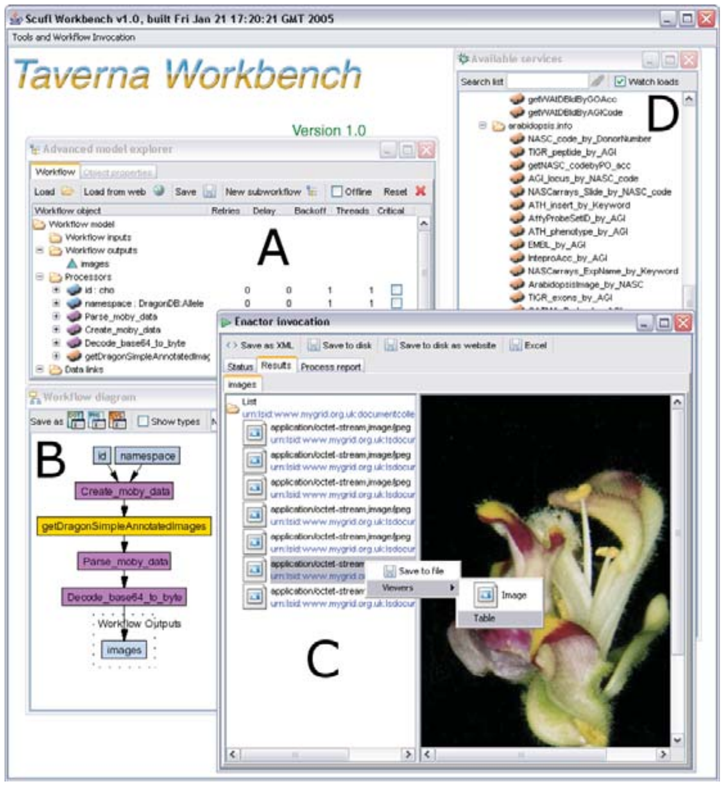


Figure 19.4: The Taverna workbench showing a tree structure explorer (a) and a graphical diagram view (b) of a Scufl workflow. The results of this workflow are shown in the enactor invocation window in the foreground (c). A service palette showing the range of operations that can be used in the composition of a workflow is also shown (d).

The Scufl language [326] is essentially a dataflow centric language, defining a graph of data interactions between different services (or, more strictly, processors). Scufl is designed to reflect the user's abstraction of the *in silico* experiment rather than the low-level details of the enactment of that experiment.

Internally to Taverna, Scufl is represented using a Workflow Object Model along with additional information gained from introspecting over the services. A typical workflow developed in the systems biology use case is shown in Figure 19.1.

The components of a Scufl workflow are:

- A set of inputs that are entry points for the data for the workflow.
- A set of outputs that are exit points for the data for the workflow.
- A set of processors, each of which represents a logical service — an individual step within a workflow. A processor includes a set of input ports and a set of output ports. From the user's perspective, the behavior of a processor is to receive data on its input ports (processing the data internally) and to produce data on its output ports.
- A set of data links that link data sources to data destinations. The data sources can be inputs or processor output ports, and data destinations can be outputs or processor input ports.
- A set of coordination links that enable running order dependencies to be expressed where direct data flow is not required by providing additional constraints on the behavior of the linked processors. For example, in Figure 19.1, the coordination links are defined so that one processor will not process its data until another processor completes, even though there is no direct data connection between them.

Part of the complexity of workflow design is when the user needs to deal with collections, control structures such as iterations, and error handling. Scufl is simplified to the extent that these are implicit. This layer fills in these implicit assumptions by interpreting an Internal Object Model that encodes the data that passes through a workflow. This data model is lightweight; it contains some basic data structures, such as lists and trees, and enables the decoration of data with MIME types and semantic descriptions to enable later discovery or viewing of the data.

The addition of data structures such as lists to the data object model brings about an added complexity. There are a number of ways in which the list could be handled by the service. Taverna uses an implicit, but configurable, iteration mechanism, as shown in Figure 19.5. Where a processor takes a single list as inputs, the enactment engine will invoke the processor multiple times and collate the results into a new list. Where a processor takes two (or more) list inputs, the service will be invoked with either the cross or dot product of the two lists.

Taverna supports fault tolerance through a configurable mechanism; processors will retry a failed service invocation a number of times, often with increasing delays between retry attempts before finally reporting failure. Users can specify alternative services for any Scufl processor in the order in which they should be substituted. Alternative services are typically either an identical service supplied by an alternative service provider or, rarely, a completely different service that the user deems to be substitutable without damaging the workflow's intention.

While the Scufl language defines the data flow, it does not fully describe the service interactions to enable this data flow.

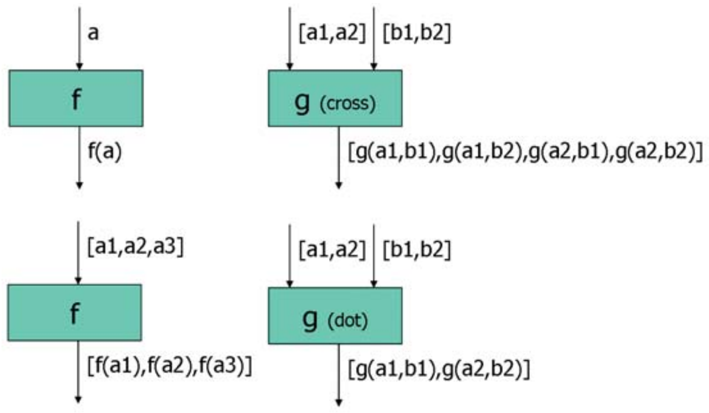


Figure 19.5: Configurable iteration. For example, a processor implements a function f — it takes one input a and produces result $f(a)$. If this processor is given a list of inputs [$a1$,$a2$,$a3$], the implicit iteration will produce a list of results, one for each input. This is equivalent to "map f [$a1$,$a2$,$a3$]." Where a processor has more than one input, the default is to apply the function to the cross product of all the input lists, however, sometimes the dot product is required. The configurable iterators allow users to specify how the lists of input values should be combined using these cross and dot operators.

It would be impossible to describe the interaction with all of the different service interfaces within a language like Scufl. Instead, Scufl is designed to be extensible through the use of processor types. We define a set of *processor plug-ins* that manage service interaction by presenting a common abstraction over these different styles. Current processors include:

- A WSDL Scufl processor implemented by a single Web service operation described in a WSDL file.
- A local Java function processor, where services are provided directly through a Java implementation with parameters as input ports and results as output ports (Figure 19.1).
- A Soaplab processor, implemented through a CORBA-like stateful protocol of the Web service operations in a Soaplab service.
- A nested workflow processor, implemented by a Scufl workflow (Figure 19.1).
- A BioMOBY processor (Figure 19.6). Several smaller groups have adopted the BioMOBY project's conventions for publishing Web services. BioMOBY provides a registry and messaging format for bioinformatics services [469].
- A SeqHound processor that manages a representational state transfer (REST) style interface, where all information required for the service

invocation is encoded in a single HTTP GET or POST request (Figure 19.6).
- A BioMart processor that directly accesses predefined queries over a relational database using a JDBC connection (Figure 19.6).
- A Styx processor that executes a workflow subgraph containing streamed services using peer-to-peer data transfer based on the Styx Grid service protocol [357].

The Freefluo engine is responsible for the enactment of the workflow. The core of the engine is workflow language independent, with specific extensions that specialize Freefluo to enable it to enact Scufl.

19.5 Discovering Resources and Designing Workflows

Workflow construction is driven by the domain expert, that is, the scientist. This corresponds to designing a suitable laboratory protocol for their investigation. The life cycle of an *in silico* experiment (see Figure 19.2) has the following stages:

- *Hypothesis formation.* First, the scientist determines the overall intention of the experiment. This informs a top-level design, and would be the overall "shape" of the workflow, including its inputs and desired outputs.
- *Workflow design.* Second, this design is translated into a concrete plan. In the laboratory, this translation would consist of choosing appropriate experimental protocols and conditions. In an e-Science workflow, this maps to the choice and configuration of data and analysis services.
- *Collecting.* The workflow needs to be run, the services invoked, data coordinated, etc (See Section 19.6). In the laboratory, this is handled by protocols for entering results in laboratory books. As the workflow is executed, the results have to be collected and coordinated to record their derivation path. To comply with scientific practice, records need to be kept on where these data came from, when they were acquired, who designed and who ran the workflow, and so forth. This is the provenance of the workflow and is described more fully in Section 19.7.
- *Analyzing and sharing.* As in a laboratory experiment, results are analyzed and then shared.

19.5.1 Service Discovery

In this section, we describe the *service discovery* and *service choice* aspects of running *in silico* experiments in Taverna.

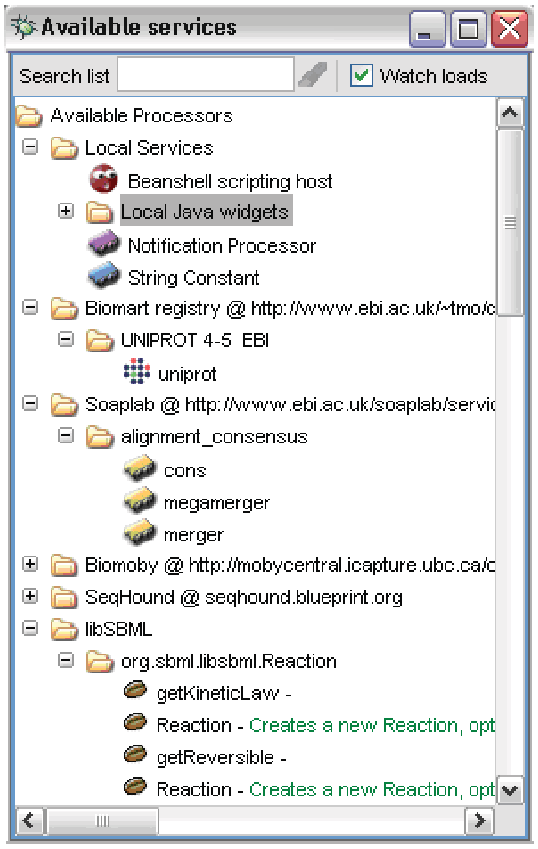
Taverna uses a variety of different mechanisms for discovery of services and populates the service list using an incremental approach. Flexible approaches to discovering available resources are an essential part of supporting the experimental life cycle:

- *Public registries such as UDDI* [430]. We are in favor of registries, but their limited usefulness is due to the lack of widespread deployment. They are generally perceived by the community to be a heavyweight solution [430].
- *GRIMOIRES.* An enriched prototype UDDI registry service developed by $^{\text{my}}$Grid, with the ability to store semantic metadata about services.
- *URL submission.* Users can add new services by directly pointing to a URL containing WSDL files. The workbench will introspect over the description and add the described services to a palette of services.
- *Workflow introspection.* Users can exploit existing experience by loading existing workflows, observing how services have been used in context, and adding those services to the available services palette.
- *Processor-specific mechanisms.* Many of the service types Taverna supports through its processor plug-ins provide their own methods for service discovery.
- *Scavenging.* Local disks are scavenged for WSDL files that are introspected over, or users create a Web page containing links to service descriptions and, when pointed at this page, Taverna explores all available service descriptions, extracts services, and makes them available. While crude, this works well and gives users considerable flexibility in loading the palette of available services that fits their current requirements.

Taverna's access to 3000 services means that service selection is increasingly important. Figure 19.6 is grouped according to the service locations, which means that services of the same type are grouped together and color coded. In addition, there is a simple search by name facility.

A common task is to locate a new service based on some conceptual description of the service semantics. To enable service selection by bioinformaticians, we must represent their view of the services and domain [480]. We have investigated a number of different mechanisms to drive the search process, including an RDF-based metadata-enriched UDDI registry [269], and a domain ontology [481] described in the W3C Web Ontology Language OWL.

Feta is our third and most recent version of a component for semantically searching for candidate services that takes a user-oriented approach to service discovery [268], a path also being trodden by the BioMOBY project. In practice, this means we describe an abstraction over the services—provided by the Taverna processors—rather than the services themselves. We have relatively shallow descriptions of the services. Although richer descriptions might enable more refined searching and sophisticated reasoning, they are expensive and time consuming to provide. In practice, search results do not have to be precise, as the final choice is made by the workflow designer (a biologist), not automatically by a machine. Finally, the use of shallow descriptions enables us to use simpler technologies to answer queries.



Available Processors
- Local Services
  - Beanshell scripting host
  - Local Java widgets
  - Notification Processor
  - String Constant
- Biomart registry @ http://www.ebi.ac.uk/~tmo/c
  - UNIPROT 4-5 EBI
    - uniprot
- Soaplab @ http://www.ebi.ac.uk/soaplab/servic
  - alignment_consensus
    - cons
    - megamerger
    - merger
- Biomoby @ http://mobycentral.icapture.ubc.ca/c
- SeqHound @ seqhound.blueprint.org
- libSBML
  - org.sbml.libsbml.Reaction
    - getKineticLaw -
    - Reaction - Creates a new Reaction, opt
    - getReversible -
    - Reaction - Creates a new Reaction, opt

Figure 19.6: An example palette of local (BeanShell scripts, Java widgets) and remote (Biomart, Soaplab, BioMOBY, Seqhound) services that can be used for the construction of workflows in Taverna. libSBML methods made available as local services via the API consumer and that were used for the construction of the exemplar systems biology workflow are also shown.

19.5.2 Service Composition

Most workflow design packages have adopted a view analogous to electric circuit layout, with services represented as "chips" with pins for input and output [20,409]. However, from a user interface point of view, this arrangement can become less understandable as complexity increases. If the layout of service components onscreen is left under the user's control, then the user can tailor the workflow appearance, but this can result in a large amount of time being spent effectively doing graph layout rather than e-Science. In Taverna, the graphical view of a workflow is read-only; it is generated from the underlying workflow model. One advantage of this is that it is easy to generate different graphical views of the workflow, showing more or less detail as required.

When composing workflows in an open world, we have no control over the data types used by the component services. A service identified by a scientist as being suitable may not use the same type as the preceding service in the workflow, even if the data match at a conceptual level. Consequently, many of the bioinformatics workflows created in Taverna contain numerous "shim" services [202] that reconcile the inevitable type mismatches between autonomous third-party services. We are currently building libraries of shims for dereferencing identifiers, syntax and semantic translation, mapping, parsing, differencing, and so on.

19.6 Executing and Monitoring Workflows

Execution of a workflow is largely an unseen activity, except for monitoring the process and reviewing records of an experimental run (see Section 19.7). A critical requirement of ^{my}Grid's service approach is that workflow invocation behavior should be independent of the workflow enactment service used. To facilitate peer review of novel results, it is important that other scientists be able to reproduce *in silico* experiments in their context and verify that their results confirm the reported novel results.

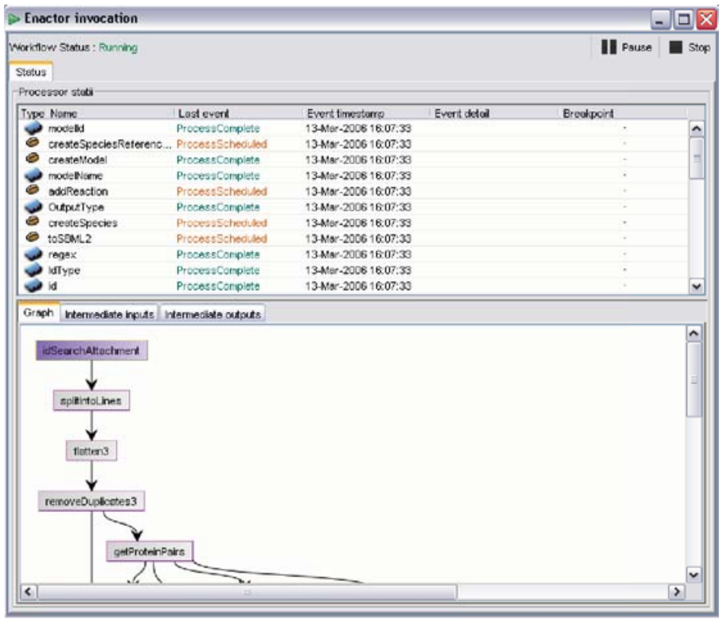
Executing workflows using different enactment services is given less emphasis in business workflows, which will typically be carefully negotiated and agreed by the businesses involved and executed in a fixed, known context. In contrast, a scientific workflow will be shared and evolved by a community and executed by many individual scientists using their favored workflow enactment service.

19.6.1 Reporting

Reporting the progress of a workflow is a complex task. Information about service invocation is unavailable in the general case. Defining how far a service is through a given invocation, so progress can be displayed, is nontrivial without the explicit modeling and monitoring of state. The migration of application services to the Grid's Web Service Resource Framework [100] is a solution that we are investigating.

The reporting mechanism in Taverna is a stream of events for each processing entity, with these events corresponding to state transitions of the service component. For example, a message is emitted when the service is first scheduled, when it has failed for the third time and is waiting to retry, etc. These message streams are collated into an XML document format and the results presented to the user in tabular form as shown in Figure 19.7.

The introduction of reporting in Taverna does not alter the workflow results. What it does alter is users' understanding of what is going on and



Type	Name	Last event	Event timestamp	Event detail	Breakpoint
	modelId	ProcessComplete	13-Mar-2006 16:07:33		-
	createSpeciesReferenc...	ProcessScheduled	13-Mar-2006 16:07:33		-
	createModel	ProcessComplete	13-Mar-2006 16:07:33		-
	modelName	ProcessComplete	13-Mar-2006 16:07:33		-
	addReaction	ProcessScheduled	13-Mar-2006 16:07:33		-
	OutputType	ProcessComplete	13-Mar-2006 16:07:33		-
	createSpecies	ProcessScheduled	13-Mar-2006 16:07:33		-
	toSBML2	ProcessScheduled	13-Mar-2006 16:07:33		-
	regex	ProcessComplete	13-Mar-2006 16:07:33		-
	idType	ProcessComplete	13-Mar-2006 16:07:33		-
	id	ProcessComplete	13-Mar-2006 16:07:33		-

Figure 19.7: Status information. When running a workflow, the Taverna workbench displays status information from the workflow enactor. For each Scufl processor, the last event is displayed along with the appropriate time and additional detail if available. This additional detail can include progress through an iteration (e.g. "item 2 of 6") and retry information. The status information also allows the selection of a processor and viewing of the relevant intermediate inputs and outputs. Each data item has been assigned a Life Science Identifier (LSID). More detailed trace information is also available using the "Process report" tab.

therefore their confidence that the system is doing what they want. Overall, the feedback from Taverna's initial users was that workflow execution without suitable monitoring was not acceptable. They were willing to accept workflows that occasionally failed; their experience with form-based Web services was that these were unreliable. However, workflow execution could not be a "black-box" service, users need feedback on what is happening, whether the workflow completed successfully or failed, and they need this recorded in logging records.

When a workflow may contain 50 or more processing components (e.g. Scufl processors), and each of these components can be retrying, using alternative implementations, etc., the complete state of a workflow is highly complex. Users require a visualization that allows them to see at a glance what is happening, acquire intermediate results where appropriate, and control the workflow progress manually should that be required.

19.7 Managing and Sharing Workflows and Their Results

As the use of workflows increases the ability to gather and generate data in large quantities, the storage of these data in an organized manner becomes essential for analysis within and between experiments. For scientists, workflows are the means to an end; their primary interest is in the results of experiments. This interest, however, goes beyond examining the results themselves and extends to the context within which those results exist. Specifically, the scientist will wish to know from where a particular result was derived, which key process was used, and what parameters were applied to that process. Thus, in addition to the raw data, we have devised a model of meta data describing the provenance of all aspects of the experiment: the data's derivation path, an audit trail of the services invoked, the context of the workflow, and the evidence of the knowledge outcomes as a result of its execution [494]. Another view is that it is the traditional who, where, when, what, and how questions applied to *in silico* science. These different aspects of provenance can be used for life scientists in different scenarios:

- to repeat a workflow execution by retrieving the "recipe" recorded in the provenance;
- to reproduce a data product by retrieving the intermediate results or inputs from which these data were derived;
- to assess the performance of a service that is invoked in different experiment runs at different times;
- to debug the failure of a workflow run, e.g. which service failed, when and why it failed etc.;
- to analyze the impacts of a service/database update on the experiment results, by comparing the provenance of repeated runs;
- to "smartly" rerun a workflow if a service is updated by using provenance to compute which part of a workflow is required to be rerun as a consequence of the update; and
- to aggregate provenance of a common data product that is produced in multiple runs.

We have adopted two key technologies for provenance collection:

- *Life Science Identifiers.* The description of the derivation of data necessitates reference to the data sets both inside and outside the control of $^{\mathrm{my}}$Grid. Bioinformatics has adopted view standards for the identification of data instead of using an ad hoc system of *accession numbers*. The recent Life Science Identifier (LSID) standard [93] provides a migration path from the legacy accession numbers to an identification scheme based on URIs.
- *Resource Description Framework (RDF).* The Dako data store has a fixed schema that reflects the common entities used in an e-Science experimental

life cycle not tied to any scientific discipline. The use of a fixed schema provides performance benefits. However, RDF's basic graph data model is well suited to the task of representing data derivation. The Knowledge Annotation and Verification of Experiments (KAVE) meta data store has a flexible schema due to its use of RDF. This allows statements to be added outside the fixed schema of the Dako data store, as is needed when providing subject-specific information. KAVE enables other components in ^{my}Grid to store statements about resources and later query those statements.

One can distinguish between provenance of the data and provenance of the process, although the two are linked. The primary task for data provenance is to allow the exploration of results and the determination of the derivation path for the result itself in terms of input data and intermediate results en route to the final value. "Side effect" information about how intermediate and final results have been obtained is generated and stored during workflow invocation. Thus the workflow engine produces not just results but also provenance meta data about those results. Side effect information is anything that could be recorded by some agent observing the workflow invocation, and it implicitly or explicitly links the inputs and outputs of each service operation within the workflow in some meaningful fashion. The associated component RDF Provenance Plug-in listens to the events of workflow execution and stores relevant statements using KAVE; for example, a name for a newly created data item or a meaningful link between the output of a service and the inputs that were used in its creation.

Process provenance is somewhat simpler than data provenance and is similar to traditional event logging. Knowledge provenance is the most advanced and contextual of the meta data results. Often a user does not need to see a full "blow by blow" account of the processes that executed during the workflow or a full account of the complete data-derivation path. Instead they wish to relate data outcomes across a group of processes annotating the relationships between outcomes with more semantically meaningful terms than "derived by." As each such provenance fingerprint is unique to the workflow and the user, a *provenance template* accompanies the Scufl document to be populated by the provenance capture component and stored in the KAVE.

19.8 Related Work

In life sciences there are many scientists who want an easy way of rapidly pulling together third-party services into prototypical *in silico* experiments. This contrasts with fields such as physics and astronomy, where the prime scenario involves carefully designed workflows linking applications to exploit computational Grid resources for *in silico* experiments that were previously impractical due to resource constraints.

Scientific workflow systems vary in terms of their intended scientific scope (the kinds of analyses supported), their technical scope (the kinds of resources that can be composed), their openness to incorporating new services, and whether or not they are open source. The strengths of Taverna are its ability to link together a significant range of autonomous bioinformatics services and its flexibility, particularly in terms of the metadata generated to help manage and share workflow results.

The Kepler workflow system [19, 20] has been developed for ecologists, geologists and biologists and is built on Ptolemy II, a mature application from electrical engineering [366]. Kepler's strengths include its library of Actors, which are mainly local applications, and its suite of Directors that provide flexible control strategies for the composition of Actors. The Triana [409] system was originally developed as a data analysis environment for a gravitational wave detection project. Like Taverna and Kepler, Triana is also data-flow oriented. It is aimed at CPU intensive applications, allowing scientists to compose their local applications and distribute the computation.

DiscoveryNet uses a proprietary workflow engine, and all services are wrapped to conform to a standard tabular data model. DiscoveryNet scientific workflows are used to allow scientists to plan, manage, share, and execute knowledge discovery and data analysis procedures [373]. In the Pegasus system [160], users provide a workflow template and artificial intelligence planning techniques are used to coordinate the execution of applications on a heterogeneous and changing set of computational resources. The emphasis is on the scheduling large numbers of jobs on a computational Grid, where there may be alternative strategies for calculating a user's result set.

The use of workflows for "programming in the large" to compose web services has led to significant interest in a standard workflow language, with BPEL[1] [24] a strong candidate, created through the agreed merge of IBM's WSFL [254] and Microsoft's XLANG [416]. One reason why Taverna workflows use Scufl rather than a potential standard is historical. In the initial stages of the ^{my}Grid project in 2001, BPEL did not exist. The more significant reason is conceptual. Initial experiments showed IBM's WSFL language did not match how our target users wanted to describe their *in silico* experiments [7]. WSFL forced users to think in terms of Web service ports and messages rather than passing data between bioservices.

[1] BPEL was originally termed BPEL4WS and is being promoted as a standard called WSBPEL through OASIS (Organization for the Advancement of Structured Information Standards), an international consortium for e-business standards.

19.9 Discussion and Future Directions

$^{\text{my}}$Grid set out to build a workflow environment to allow scientists to perform their current bioinformatics tasks in a more explicit, repeatable, and shareable manner:

- *Making tacit procedural knowledge explicit.* For at least the last 250 years, this has been recognized as essential in science. Each experiment must carry with it a detailed "methods" description to allow others both to validate the results and also reuse the experimental method. Our experience suggests that workflows allow this to be achieved for *in silico* experiments. They are formal, precise, and explicit, yet straightforward to explain to others.
- *Ease of automation.* Many of the analyzes we support have already been undertaken by scientists who orchestrate their applications by hand. Workflows can drastically reduce analysis time by automation. For example, Taverna workflows developed by the Williams–Beuren Syndrome team have reduced a manual task that took two weeks to be an automated task that typically takes just over two hours [397].
- *Appropriate level of abstraction.* Bioinformaticians have traditionally automated analyzes through the use of scripting languages such as PERL. These are notoriously difficult to understand, often because they can conflate the high-level orchestration at the application level with low-level "plumbing."

Taverna and the $^{\text{my}}$Grid suite enables users to rapidly *interoperate* services. It does not support the semantic *integration* of the data outcomes of those services. We underestimated the amount of data integration and visualization provided by the existing Web-delivered applications. They often integrate information from many different analysis tools and provide cross-references to other resources. Accessing the analysis tool directly as a service circumvents this useful functionality. Although the scientist is presented with results in hours, not weeks, it now takes significant time to analyze the large amount of often fragmented results. A solution is complicated by the fact that the workflow environment does not "understand" the data and so cannot perform the data integration necessary. We have provided integration steps within workflows, written as scripts that integrate and render results, but these are specific to each workflow design. We are currently investigating a multi-pronged approach: (i) the use of Semantic Web technology to provide more generic solutions that can be reused between related workflows; (ii) appropriate workflow designs using shims and services under the control of the user to build data objects; and (iii) closing off the open world in situations where the workflows are known to orchestrate a limited number of services and will be permanent in nature, so it is worth the effort to build a more strongly typed model.

Since January 2006, the ^{my}Grid suite, including Taverna 2.0, has moved to a new phase. As part of the United Kindom's Open Middleware Infrastructure Institute (OMII-UK)(`http://www.omii.ac.uk`), ^{my}Grid is to be integrated with a range of Grid services and deployed in a common container with job submission services, monitoring services, and large-scale data management services. Focus is placed on the following:

- *Grid deployment.* Deploying the Taverna architecture within a Grid container, making the enactor a stateful service, and a server-side distributed service, and supporting stateful data repositories.
- *Improved security.* Authentication and authorization management for data, metadata and implementation of credentials for access control of services.
- *Revised execution and processor models.* Support of interactive applications, long running processes, control-based workflows, data flows with large data throughput, enhanced provenance collection, and credential handling. We already have a user interaction service that allows users to participate interactively with workflows.
- *Improved data and metadata management.* Incorporating better user-oriented result viewers and incorporating SRB and OGSA-DAI data implementations.
- *Integration with third-party platforms.* Examples are Toolbus and EGEE. We also plan to continue to interoperate with other workflow systems, specifically Kepler and the ActiveBPEL system emerging from UCL.
- *Extending services.* To execute over more domain services, such as the R suite, and over generic services such as GridSAM job submission.

The field of scientific workflows is rapidly evolving, and as a project in this area ^{my}Grid must also evolve. We engage different user communities (such as biological simulation), and new applications become available, as do novel service frameworks for deploying them. By working closely with our users, service providers, and other workflow projects, we continue to extend the basic core functionality to fulfill a wide range of uses.

Acknowledgments

This work is supported by the UK e-Science programme ^{my}Grid grants EPSRC GR/R67743, EP/D044324/1, EP/C536444/1, and the Link-Up e-Science sisters project. The authors would like to acknowledge the ^{my}Grid team. Peter Li is funded by the BBSRC. Hannah Tipney developed workflows for investigating Williams–Beuren Syndrome and is supported by The Wellcome Foundation (G/R:1061183). We also thank our industrial partners: IBM, Sun Microsystems, GlaxoSmithKline, AstraZeneca, Merck KgaA, geneticXchange, Epistemics Ltd, and Cerebra.