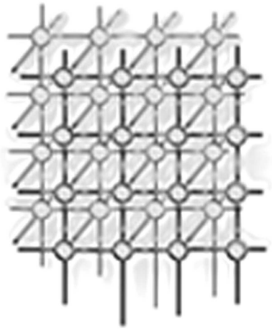
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*



Taverna: Lessons in creating a workflow environment for the life sciences

Tom OINN[1],Mark GREENWOOD[4], Matthew ADDIS[2], M. Nedim ALPDEMIR[4], Justin FERRIS[2], Kevin GLOVER[3], Carole GOBLE[4], Antoon GODERIS[4], Duncan HULL[4], Darren MARVIN[2], Peter LI[5], Phillip LORD[4], Matthew R. POCOCK[5], Martin SENGER[1], Robert STEVENS[4], Anil WIPAT[5] and Chris WROE[4]

[1]*EMBL European Bioinformatics Institute, Hinxton, Cambridge, CB10 1SD, UK*
[2]*IT Innovation Centre, University of Southampton, SO16 7NP, UK*
[3]*School of Computer Science and Information Technology, University of Nottingham, NG8 1BB, UK*
[4]*School of Computer Science, University of Manchester, M13 9PL, UK*
[5]*School of Computing Science, University of Newcastle, NE1 7RU, UK*

KEY WORDS: scientific workflow, Semantic Grid environment, life sciences, Web Services

SUMMARY

Life sciences research is based on individuals, often with diverse skills, assembled into research groups. These groups use their specialist expertise to address scientific problems. The *in silico* experiments undertaken by these research groups can be represented as workflows involving the co-ordinated use of analysis programs and information repositories that may be globally distributed. With regards to Grid computing, the requirements relate to the sharing of analysis and information resources rather than sharing computational power. The ^{my}Grid project has developed the Taverna workbench for the composition and execution of workflows for the life sciences community. This experience paper describes lessons learnt during the development of Taverna. A common theme is the importance of understanding how workflows fit into the scientists' experimental context. The lessons reflect an evolving understanding of life scientists' requirements on a workflow environment, which is relevant to other areas of data intensive and exploratory science.

*Correspondence to: Mark Greenwood, School of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, UK.
E-mail: markg@cs.man.ac.uk
Telephone : +44 (0) 161 275 6183
Fax: +44 (0) 161 275 6204.

Received
Revised

1. Introduction

Much of biology is based on comparative and speculative reasoning: predictions are based on similar observations made previously. Discovery involves combining and collating results obtained from a number of local and remote analyses and data resources available to the community. These *"in silico"* experiments complement experiments performed in the laboratory by generating new information from available data and forming hypotheses for confirmation in the laboratory.

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 [1] for molecular biology and environmental data analysis, SEEK for ecology [2] , GriPhyn for particle physics [3], and SCEC/IT for earthquake analysis and prediction [4]. The "Workflow in Grid Systems" workshop at the GGF10 conference [5], and the formation of the Global Grid Forum Workflow Management Research Group illustrate significant and growing interest in workflows within the Grid community [6]. The diverse nature of the tasks being performed has led each group to adopt or develop a workflow solution best suited to their requirements rather than use a common standard.

In this paper, we describe ^{my}Grid: a UK e-Science pilot project building middleware to support exploratory, data-intensive, *in silico* experiments in molecular biology. ^{my}Grid has developed the Taverna workflow workbench environment which enables the scientific user to create and run workflows written in the Simplified conceptual workflow language (Scufl). These are enacted using the Freefluo workflow enactment engine. Our emphasis is on building workflows that link together third party applications (both remote and local) that are familiar to the scientist, using a language and tools designed for the scientist.

The design of the Taverna has been driven by: the users we wish to support; the nature of the existing resources they wish to orchestrate; and the type of *in silico* experiment they wish to perform.

Two classes of users ^{my}Grid supports are biologists and bioinformaticians. They have a deep knowledge of the scientific functionality of the resources they want to link together, perhaps have some limited programming experience, but have little or no knowledge of specific middleware solutions such as Web/Grid Services to access them.

Biology resources are published by the third class of users that ^{my}Grid caters for–the service providers. Currently, Taverna provides the means to access over 1000 of these services. They take the form of applications such as analysis algorithms for comparing sequences, databases arising from species-specific genome projects or holding cross species data sets for proteins or nucleotides, visualisation tools for protein structures, simulations of heart excitation models and so on. Some are replicated but many are unique and only available through licensing arrangements at the host site. Most have poor or missing programmatic interfaces. Although many early bioinformatics tools used the command line extensively (e.g. GCG [7] or EMBOSS [8]), more recently analysis has moved toward web based interfaces. Thus

data transfer moved from file transfers between commands to "cut and paste" between web browsers. The resources encompass heterogeneous and semi-structured data exposed using a diverse range of mechanisms. There are no prescribed standards for formats, APIs or delivery platforms.

Finally, ^{my}Grid users are often members of small research groups that begin their analyses, as low volume *ad-hoc* experiments that are incrementally and rapidly prototyped in an exploratory way. Some are "one-offs" and effectively disposable, whilst others are later developed into production workflows which will be executed repeatedly. Much of the scientific benefit comes from combining results from many different workflow runs, so it is essential to manage the data produced by each experiment. Scientists frequently undertake similar analysis to that of other groups, and therefore the workflow designed by one user is often suitable for adoption or adaptation by many others.

These requirements have lead to several major design lessons:

An open world service assumption. Biologists have strong opinions about the particular services that they wish to use; they generally do not accept substitutes. We wanted to be able to use any service as it was presented, rather than require service providers to implement services in a prescribed manner and thus create a barrier to adoption. Accordingly, Taverna caters for a variety of different service interfaces, and does not require adherence to a common universal type system. Consequently, the data is largely opaque to the middleware. This is a drawback when we come to integrate results.

Easy and rapid *ad-hoc* workflow design. Quickly and easily finding services and adapting previous workflows is key to effective workflow prototyping. As the target end users for Taverna are not necessarily expert programmers, we have: developed a graphical workflow workbench; developed a portal for launching workflows; used semantic technologies to provide service descriptions that are closer to scientists' view of their experiments than implementation-specific syntactic types [9]; and, most importantly, defined a tiered architecture that hides the complexity of different services enabling the user to think about the experiment rather than its execution.

A multi-tiered abstraction architecture. We have a requirement to both present a straight forward 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 multi-tiered approach to resource discovery and execution that separates application and user concerns from operational and middleware concerns. The result is a three-tiered model for describing resources and their interoperation at different levels of abstraction. Scufl, a workflow language for linking applications, 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.

A data flow centric model. Bioinformaticians are familiar with the notion of data flow centric analysis. Much of the analysis we want to support is in the form of pipelines in

which data about a particular biological entity is integrated from a number of resources and further analysed. Examples include workflows to support work on the genetic basis of Graves' Disease [10] and Williams Beuren Syndrome (WBS) [11]. Thus, to support bioinformaticians in their current practices Taverna emphasises a data flow centric specification.

Fault tolerance. Any software operating in a networked, distributed environment is required to cope gracefully with failure. In bioinformatics, where many services are not professionally supported, service failure is common. Fault tolerance mechanisms such as dynamic service substitution and retry are supported by Taverna.

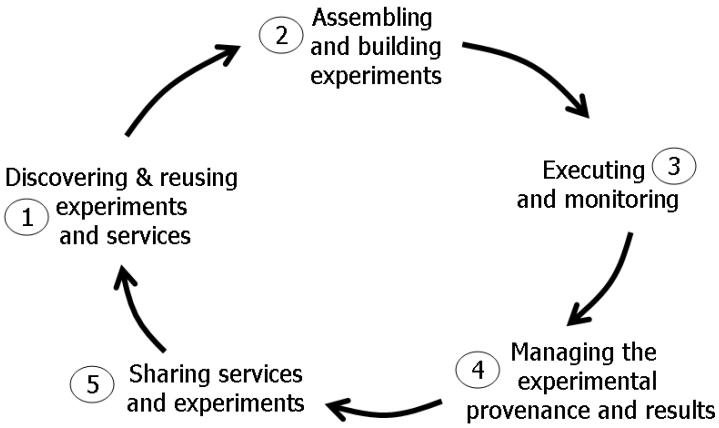
Support for the e-Science lifecycle. Using a workflow makes it easier for scientists to describe and run their experiments in a structured, repeatable and verifiable way compared to cutting and pasting between multiple web-based forms. However, workflows are not a complete solution for supporting *in silico* experiments. They exist in a wider context of scientific data management, as illustrated in Figure 1. The user is still at the centre, interacting with the workflows and the services, and interpreting the outcomes. It is desirable that workflows become a resource in their own right, to be described and shared (steps 1 and 5). It is essential that data produced by a workflow carries with it some record of how and why it was produced i.e. the provenance of the data (step 4). Thus ^{my}Grid has built an environment and components that marry the workflow environment with sophisticated methods of provenance collection [12] and semantic-based resource and workflow discovery [9]. All ^{my}Grid components are organised into a context-aware Service Oriented Architecture coordinated by a common data model and an events-bus [13].

To steer ^{my}Grid and Taverna developments, there has been close collaboration with two groups investigating the genetics of human disease. The Institute for Human Genetics at Newcastle University, UK have developed workflows investigating the basis for Graves' Disease [10]. While St Mary's Hospital at the University of Manchester, UK are investigating the foundations of WBS [11]. Other scientists using Taverna are working on gene annotation for the investigation of susceptibility to Trypanosomiasis in cattle, investigations for small molecules, executing the JUMBO library for molecules, gene identification in the chicken genome, and managing simulations of cellular models of cardiac electrophysiology. Taverna currently has over 500 installations. Consequently, we are now in a position to assess what it is about workflows that makes them such a promising technology:

Making tacit procedural knowledge explicit: For at least the last 250 years this has been recognised as essential in science. Each experiment must carry with it a detailed "methods" description, to allow others both to validate the results, but also re-use the experimental method for their own purposes. Our experience suggests that workflows allow the same 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 analysis we support are already undertaken by scientists who orchestrate their applications by hand. Workflows can drastically reduce analysis

Figure 1. The e-Science life cycle



time by automation. For example, Taverna workflows developed by the WBS team have reduced a manual task that took 2 weeks to be an automated task that typically takes just over two hours.

Appropriate level of abstraction Bioinformaticians have traditionally automated analyses 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". Workflow systems such as ^{my}Grid's support the separation of these concerns.

The rest of the paper is organised as follows. Section 2 further elaborates on the background to Taverna, outlining requirements in detail and presenting the life cycle of in silico experimentation. Section 3 introduces the major Taverna components. Section 4 concentrates on the workflow design phases of the life cycle, and Section 5 on executing and monitoring workflows. Section 6 completes the life cycle with metadata and provenance associated with managing and sharing results, and the workflows themselves. Section 7 discusses related work. Section 8 discusses the lessons learnt from these experiences, highlighting that in many cases the issues and solutions involve related technical and non-technical aspects. These lessons drive our current and future work.

2. Further Background of Taverna

By taking a concrete example we can illustrate the kinds of users, analyses and resources that inform our requirements, and in turn impact on the design of the workflow system. Members of St Mary's Hospital Academic Unit of Medical Genetics, at Manchester University are investigating the genetic basis of a rare congenital disorder Williams Beuren Syndrome (WBS) [11]. It is known to be caused by deletion of a small region of human chromosome seven. By examining the genes known to be in that region, researchers can begin to understand the origins of the complex physical and behavioural features of affected children. The completion of the initial phase of the Human Genome Project [14] now means researchers do not need to examine the affected region in the laboratory, but can instead search a genome database into which other large scale genome sequencing projects have deposited their results. Selected records can then be submitted for further analysis: i) to characterise any genes in those new sequences using analysis tools, and gather related information from other databases; and ii) to characterise proteins that will be produced from those genes. By adopting a workflow-based approach using Taverna, the St Mary's team have automated tasks that manually took two weeks to take two hours, and have shared and adapted their workflows with other scientists. The bioinformaticians rapidly picked up the Taverna workbench, and evolved their workflows over many generations of experimentation. The chief bottlenecks were wrapping the services and integration of the results. The former is a temporary bootstrapping issue; the latter is more significant and discussed in Section 6.

New DNA sequences are released to the various genome sequence databases on a regular (hourly) basis. As a result, the WBS researchers wish to rerun their workflows frequently (weekly) and repeatedly, to discover new sequences of relevance and to characterise them. The different classes of services needed to perform the WBS analysis are numerous (currently numbering 30), and distributed widely. They are provided by a mixture of large scale bioinformatics centres such as the National Centre for Bioinformatics (NCBI), which maintains among other things the large genome database GenBank [15], and many distributed bioinformatics groups that maintain smaller scale databases or analysis tools. For example, two analysis tools to predict the presence of genes in DNA: Genscan [16] and Twinscan [17] are published by groups at Massachusetts Institute of Technology and Washington University St Louis.

From the early 1990s, the biological community has enthusiastically adopted web technology to disseminate data and analysis methods. Bioinformaticians perform these low volume *in silico* experiments by cutting and pasting data between web pages, sometimes assisted by bespoke screen-scraping parsers and PERL scripts to overcome format discrepancies. However, the complexity of *in silico* experiments together with the volumes of data produced by high throughput technologies is now threatening to overwhelm the users of this standard web technology. Analysis methods are constantly evolving and, as more resources become available, more *in silico* experiments can be done. This in turn generates more resources and knowledge for designing further experiments.

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 [18]. The emergence of Web Services [19], along with the availability

of suitable tool support, has seen a significant number of bioinformatics web resources becoming publicly available and described with a WSDL interface. Early examples include the XEMBL [20], and openBQS [21] hosted by the European Bioinformatics Institute (EBI), and the services provided by XML Central of DDBJ [22]. Recently, the range of available Web Services is growing: the BioMOBY project is gathering an expanding range of services [23], pathway data are available from the KEGG API[†], and a range of analysis services are offered by the PathPort project [24]. Taverna does not have a monopoly on the use of these services, since they will be used by a range of different clients because of the diverse nature of the life sciences. Moreover, it is difficult to predict which small research-group experimental service of today will become the established *de-facto* standard service of the future.

Services used by WBS and other projects using Taverna include the following:

- EMBOSS, The European Molecular Biology Open Source Software Suite, a package of over 270 functions for sequence analysis originally developed by Human Genome Mapping Project. No simple Web or Grid Services are available by default. However, ^{my}Grid has developed Soaplab, a framework that allows legacy command line applications to be automatically deployed as Web Services given a description of their interface [25]. Soaplab is used to add Web Service capability to EMBOSS commands.

- SeqHound is a database of biological sequences and structures which also provides over 160 functions [26]. Again no Web/Grid service interface is available. They instead provide a Representational State Transfer (REST) style interface [27], where all information required for the service invocation is encoded in a single HTTP GET or POST request.

- Some of the smaller distributed groups have adopted the BioMOBY project's conventions for publishing Web Services. BioMOBY provides a registry and messaging format for bioinformatics services [23]. Currently over 140 services are listed in BioMOBY registries.

- Some groups have yet to publish their application as a service at all. If no service is provided we obtain the application, install it on a ^{my}Grid server and deploy it as a Web Service. This process can be simplified by the use of Soaplab.

There are currently over 1000 services accessible to a ^{my}Grid user. Although the majority involve complex interaction patterns (in the case of Soaplab/ EMBOSS), specific messaging formats (in the case of BioMOBY), or use different protocols and paradigms (in the case of SeqHound), they follow a small number of stereotyped patterns. The user's 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 the user 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 is mostly semi-structured and heterogeneous. There are a large number of data formats including those for gene sequences, for protein sequences, as well as bespoke formats produced by many analysis tools (including Genscan and Twinscan). These

[†]`http://www.genome.jp/kegg/soap/`

are rarely encoded in XML and there is usually no formal specification that describes these formats. Interpreting the data as it is passed between different databases and analysis tools is therefore difficult. This contrasts with data in other scientific workflow projects which have much more centralised control of data formats. For example, the SEEK project provides tools for ecologists within the project to describe their data using XML Schema and ontologies, and so support middleware driven data integration [28]. DiscoveryNet [1] 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 corresponding to a job to be scheduled on a Condor cluster [29].

Thus, 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-orientated architecture. Taverna's target audience of life scientists want easy access and composition of as wide a range of services as feasible. In general, the life science community is unlikely to take up an imposed data model. Besides, much of the data within bioinformatics is intrinsically hard to structure being textual, or in legacy flat file formats.

3. Architecture of Taverna

Figure 2 gives the architecture of Taverna and associated ^{my}Grid components.

Taverna has two major conceptual architectural abstractions:

A three-tiered data model for describing resources and their interoperation at different levels of abstractions, from the user perspective to the services perspective.

- The Application Data Flow layer, is aimed at the user and is characterised by a User-Level Workflow Object Model (indicated by a transparent rounded rectangle that covers most of the Taverna Workbench in Figure 2). 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 the data consumed and produced by logical services and connecting them together. They are not interested in the implementation styles of the services, and the scientists should not need to be familiar with the concepts or details of service-orientated architectures.
- The Execution Flow layer relieves the user of most 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 characterised by the Enactor Internal Object Model (indicated by a transparent rounded rectangle overlayed on Freefluo in Figure 2) and by the ^{my}Grid Contextual Information Model (indicated by a transparent rounded rectangle that spans the Taverna workbench, the workflow enactor (Freefluo), the service discovery component (Feta), and the data (MIR) and metadata (KAVE)

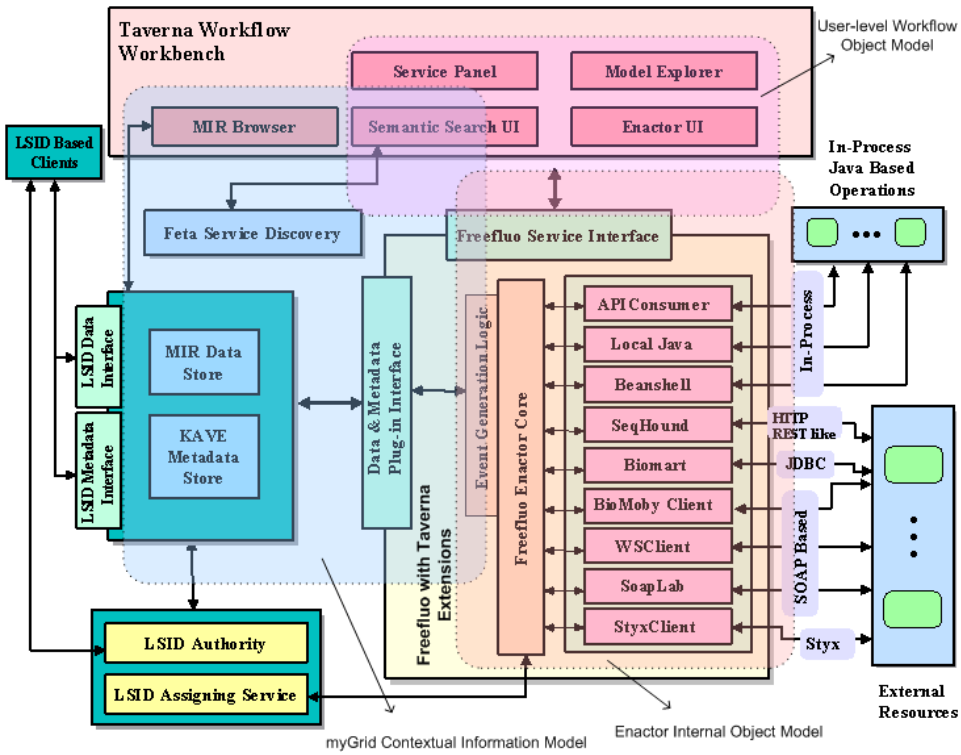


Figure 2. Architecture of Taverna and associated ^{my}Grid components

store components of ^{my}Grid). 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 to understand the behavioural semantics of this layer to avoid duplicating the implicit behaviour.

– 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 characterised by the Enactor Internal Object Model and is catered for by an extensible processor plug-in architecture for the Freefluo enactment engine.

A framework that provides three levels of extensibility:

– The first level provides a plug-in framework to add new GUI panels to facilitate user interaction for deriving and managing the behavioural 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 recognise 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 snap shots 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.

We now introduce the Taverna workbench first, and then discuss each of the architectural abstractions in more detail.

3.1. Taverna workbench

The Taverna workbench provides the main user interface to the above components. The main function of the workbench itself is to enable the construction and editing of Scufl workflows, loading and saving these in an XML serialisation (known as XScufl). The basic workbench supports the creation and editing of workflows using the model explorer and the service panel, manipulating a workflow object model. The workbench enactor User Interface (UI) provides the interface for running workflows. The enactment capability is provided by Freefluo with Taverna extensions, and communicates with the UI using the Taverna data model. The semantic search UI is an interface to the Feta semantic service discovery component to complement the basic service panel. The MIR Browser provides navigation capability over data stored in the ^{my}Grid Information Repository (MIR), including experimental designs (i.e. workflows or any other operations), specific experimental results and intermediate data.

As Scufl already provides a user-centred data-flow abstraction, the core of the workbench is a data-model centic GUI, shown in Figure 3. The main mechanism for interaction with the workflow is through the *Explorer View* (labelled A in Figure 3). This presents a tree view, showing the various services or processors present in the workflow. The same data is also visualised as a graph (B). The user can tailor the amount of information that displayed in this figure. Many of the processors, for example, present a large number of ports, most of which are unused in any particular workflow, but which complicate the display. The user may select new services to add to the workflow from a palette of available services(D). Services are gathered using a variety of different techniques, often dependent on the source. For example, ^{my}Grid supports a UDDI-based registry of services and workflows called GRIMOIRES that Feta can operate over; BioMOBY provides a central registry of services; and both Soaplab and

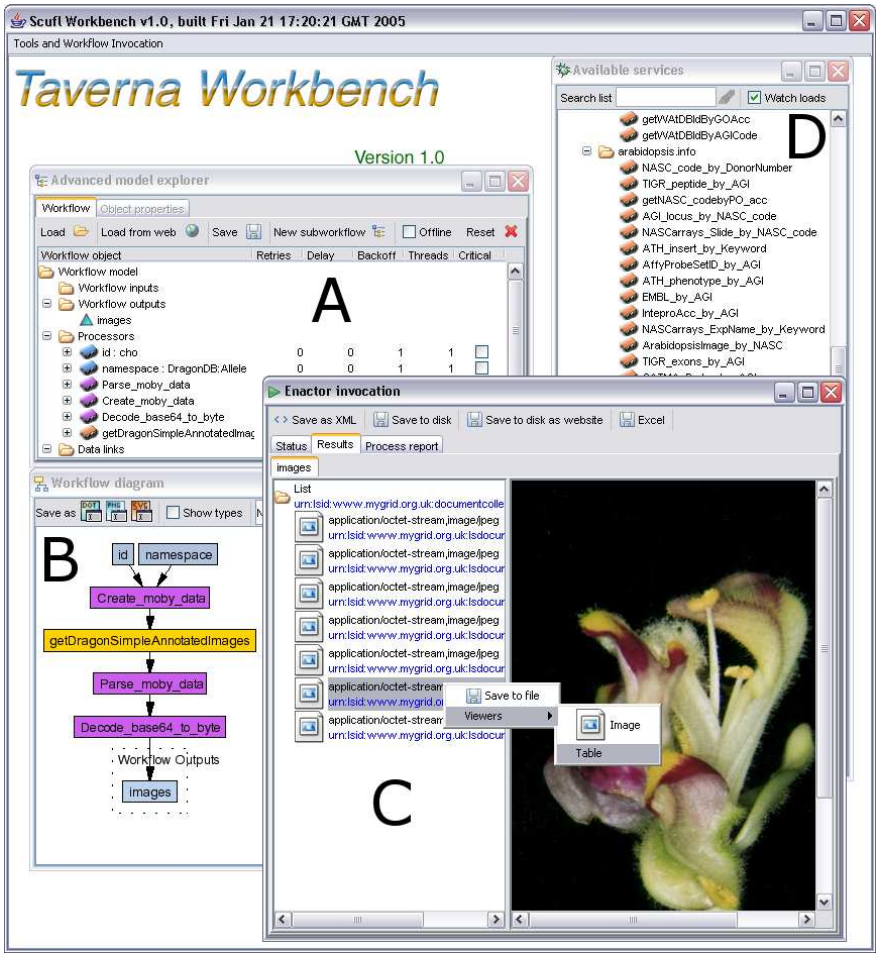


Figure 3. The Taverna Workbench showing a tree structure explorer (A) and a graphical diagram view (B) of a Scufl workflow. The results of this workflow are shown in the enactor invocation window in the foreground (C). A service palette showing the range of operations which can be used in the composition of a workflow is also shown (D).

Seqhound provide many services from one installation which can be discovered introspectively. Finally, the enactor panel enables the user to enact workflows and view the generated results.

Although construction of a workflow essentially consists of building a directed graph, describing the data flow, the user generally interacts with the workflow explorer. The graph layout view (B) is generated automatically from the information in the workflow model, using the GraphViz layout package [30]. It is currently not possible to directly interact with the

graphical view, although it is possible that later versions of the workbench will provide this functionality.

3.2. The Application Data Flow layer: the Taverna Workflow Object Model and the Scufl language

The Scufl language is essentially a data flow centric language, defining a graph of data interactions between different services (or, more strictly, processors–see Section 3.4). Scufl is designed to reflect the users 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 Graves disease use case is shown in Figure 6. A simplified example to illustrate Scufl is shown in Figure 4.

The components of a Scufl workflow are:

A set of inputs that are 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 behaviour of a processor is to receive data on its input ports, (process 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 behaviour of the linked processors. For example, in Figure 4 two coordination links are defined so that that one processor will not process its data until another processor completes, even though there is no direct data connection between them.

3.3. The Execution Flow layer

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. Unlike other projects like Pathport [24] and caBIG [31], Taverna does not attempt to formally structure the domain data itself in our attempt to cater for the use of any services we are presented by the bioinformatics community.

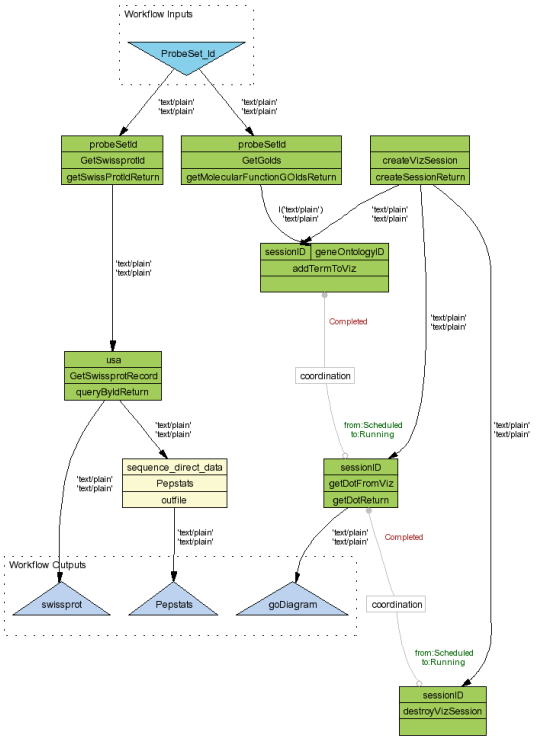


Figure 4. An Example Scufl model. This small workflow has been extracted from a Graves' Disease case study example. It shows one input, three outputs and eight processors. All the processors are labelled top to bottom with input ports, processor name and output ports. All the processors in this example are standard WSDL-described standard web services, except for "Pepstats" which is a Soaplab processor. All the links are data links except for two coordination links on the right hand side. The links are labelled with syntactic type information: "l(text/plain)" indicates a list of plain text strings.

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 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 they should be substituted. Alternative services are typically either an identical

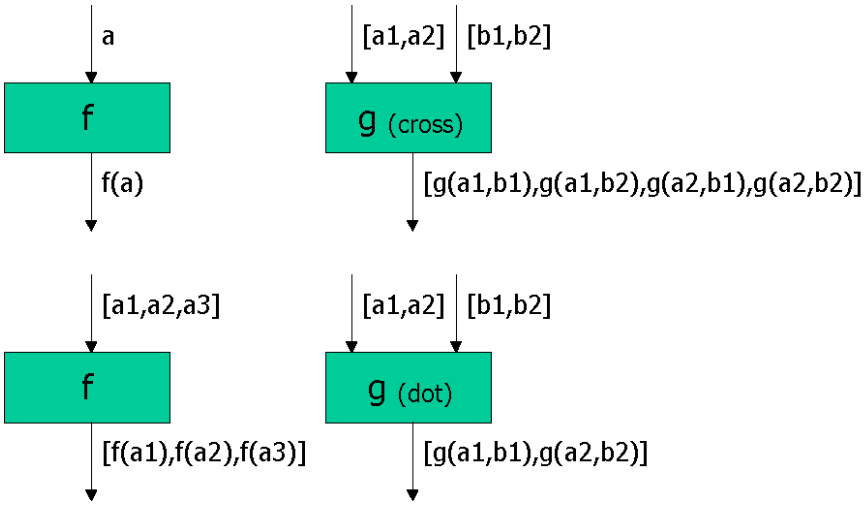


Figure 5. Configurable iteration. For example, processor implements a function f - it takes one input a and produces result f(a). If this processor is given a list of inputs [a1,a2,a3], the implicit iteration will produce a list of results, one for each input. This is equivalent to 'map f [a1,a2,a3]'. Where a processor has more that one input the default is to apply the function to the cross product of all the input lists, however sometimes the dot product is what is required. The configurable iterators allow users to specify how the lists of input value should be combined using these cross and dot operators.

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. As alternates are few they can be predefined statically before the execution of the workflow. In the case of long running workflows alternates need to be sought dynamically during execution to ensure they are available.

3.4. The Processor Invocation layer

While the Scufl language defines the data flow, it does not fully describe the service interactions to enable this data flow. As we introduced earlier, many bioinformatics services are presented with a small number of stereotypical invocation patterns. Many of the services are presented using a simple query/answer interface; in this they are probably reflective of their history being essentially a programmatic version of a HTML form directed at a CGI script. Other services are presented using standard toolkits, including Soaplab or the Seqhound services. In Table I, we show two alternate presentations of the same logical service–in this case a BLAST service [32]. In this case, the CGI-like service would be a "Document Style" interface, presented using Web Services; a Soaplab service appears as an Object Style service using Web Services; and a Seqhound service would appear as a Document Style service presented using a REST style HTTP access.

Table I. Two different service interfaces to BLAST, a widely used bioinformatics tool. BLAST operates over a biological sequence, has a number of parameters and returns a single complex BLAST report. The "Document Style" interface has a single method taking a complex set of parameters, while the "Object Style" interface uses object identifiers to provide an *ad hoc* object orientation.

Document Style	BlastReport performBlast(Sequence, gap, etc. . .);

Object Style	ObjectIdentifier getInstance();
	void setSequence(ObjectIdentifier, Sequence);
	void setGap(ObjectIdentifier, Gap);
	. . .
	BlastReport invoke(ObjectIdentifier);

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. The fields of the Web Service operation request message correspond to the input ports and the fields of the return message to the output ports.
- A local Java function processor, where services are provided by directly through a Java implementation with parameters as input ports and results as output ports.
- 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.
- A BioMOBY processor.
- 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.
- A BioMart processor that directly accesses predefined queries over a relational database using a JDBC connection.
- A Styx processor that executes a workflow subgraph containing streamed services using peer to peer data transfer based on the Styx Grid service protocol [33].

Figure 6 shows a workflow that uses a number of different processor types. We expose this to the user through a colour scheme.

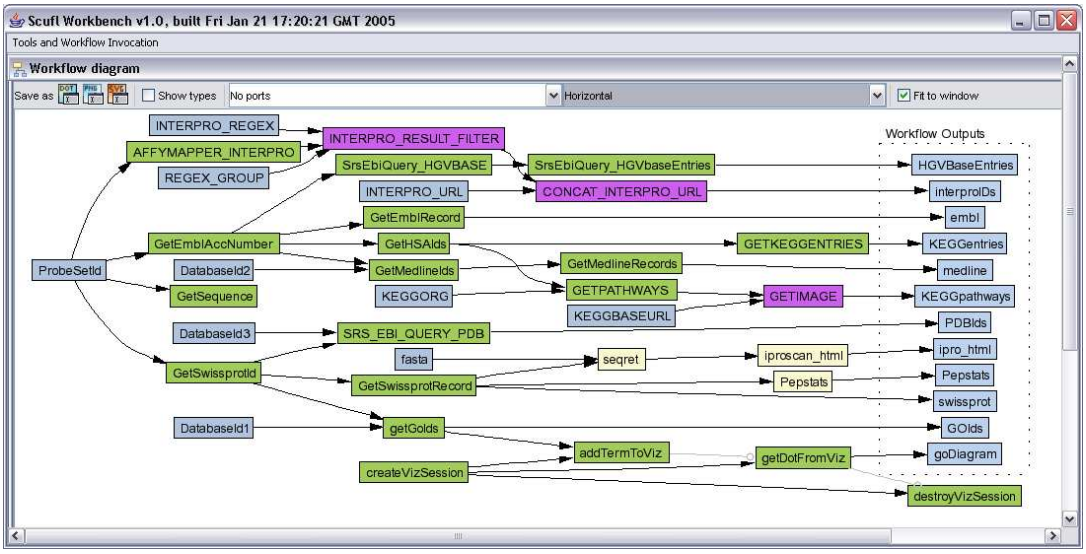


Figure 6. A Scufl workflow from the Graves' disease case study. This workflow uses a number of different Scufl processor types, e.g. WSDL Web service operations and Soaplab services. The GraphViz layout library has been used to lay the workflow left to right rather than top to bottom as in the previous diagram. The workflow input is displayed on the left and the outputs along the right edge the diagram.

3.4.1. The Freefluo enactment engine

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

3.5. An extensible framework

As stated in Section 3 Taverna provides three levels of extensibility which are explained in Section 3.5.1 through to Section 3.5.3 below.

3.5.1. Extending the workbench

The Taverna workbench provides a plug-in layer which enables GUI extensions. New extensions are required to implement a single Java interface and are then discovered using Apache's commons-discovery libraries [‡].

[‡]`http://jakarta.apache.org/commons/discovery/`

3.5.2. Extending the behaviour of the workflow execution engine

As stated above in section 3.4, Scufl is designed to be extensible through the use of processor types. A set of *Processors* are already defined which manage service interaction by presenting a common abstraction over different styles. The extensible processor plug-in architecture has an additional advantage; as well as abstracting over different style of presentation, it also abstracts away from the technology used to implement these different services. If a new kind of resource becomes available then this can be incorporated by the addition of a new processor type.

3.5.3. Extensions via the observer interface for an integrated view

Taverna facilitates integration with external services via a contextualised event observer framework [13] so that other components may be attached to provide richer capabilities. This is achieved by another plug-in layer that is integrated into the workflow execution environment, through which the contextual information and the provenance of the workflow execution is made available to the integrated components. Two aspects of this framework are important:

- A common information model [34] is combined with the pervasive use of the Life Science Identifier (LSID) [35] throughout to provide a consistent representation of experimental objects and to uniquely identify entities such as workflows and data, to record the context to the experiments and loosely couple the middleware components. As such, components such as semantic service discovery, provenance gathering and results management enable ^{my}Grid to support the whole experimental lifecycle.

- The workflow execution engine communicates with the integrated components via plug-ins that observe the events generated as the enactor's internal state changes. For example, when an intermediate step in the workflow completes its execution, the enactor generates an event and makes the intermediate results available to the event listeners. The data store plug-in responds to this event by obtaining the intermediate results and storing them in the data store (i.e. MIR) in their appropriate experimental context. As such, the plug-in architecture is instrumental in facilitating the automatic propagation of the experimental context across the participating components.

We have used these aspects extensively for those components which support the wider scientific context of the workflow. For example we have integrated a data (MIR) and a metadata (KAVE) storage service, which enables components to gather the data generated by services. These are discussed more fully in Section 4.

The existing framework will be extended further by a mediator service, namely the *e-Science Mediator* that acts as an integration broker among independently running ^{my}Grid Services including the workflow enactor service. We propose that each event generated by important state changes in individual ^{my}Grid services is defined by a typed tuple, and that the type of a data structure is specified using an XMLSchema in conformance to the ^{my}Grid information model. As such the collection of event types for a service is said to delineate an information space that is externally visible for that component. The *e-Science Mediator*

then exposes a higher-level information space that is created by processing (i.e. correlating, aggregating, filtering etc.) the raw events received from the core services. As such, complex e-Science applications can be constructed by subscribing to such e-Science events.

The following sections expand on the e-Science life cycle: the design phases, discovering resources and forming workflows; executing and monitoring workflows and finally the metadata and provenance associated with managing and sharing results.

4. Discovering resources and designing workflows

Through the plug-in mechanism of both Taverna and Freefluo the scientist can perform tasks of the extended workflow process directly from the workflow environment.

Workflow construction is placed in the hands of the domain expert, the scientist. This corresponds to designing a suitable laboratory protocol for their investigation. Creating a scientific experimental protocol can be broken down into stages. 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. 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.

In the laboratory, scientist rely upon experience and previous work to choose experimental parameters and protocols, either consulting with colleagues or using laboratory cookbooks. In moving over to *in silico* experimentation the challenge is to replicate this process, to enable service selection when composing or modifying workflows. This is particularly important in the case of Taverna as it has been designed to support the rapid development of *ad hoc*, experimental, workflows.

Service discovery can be conceptually split into two processes: 1. **Discovery:** finding available services (or workflows) so that they can be included in a Scufl model; and 2. **Selection:** selecting the appropriate service (or workflow) from this list for inclusion in a workflow. . We consider these two processes separately.

4.1. Service Discovery

Taverna uses a variety of different mechanisms for discovery of services, ranging from lightweight schemes to heavyweight registries. The population of the service list follows an incremental approach. Flexible approaches to discovering available resources are an essential part of supporting the experimental life cycle.

Public registries such as UDDI [36]. We are in favour of registries, but their limited usefulness is due to the lack of widespread deployment. They are generally perceived by the community to be a heavy-weight solution.

GRIMOIRES , an enriched prototype UDDI registry service developed by ^{my}Grid, with the ability to store semantic metadata about services (see section 4.2).

URL submission. Users can add new services, by directly pointing to a URL containing WSDL files and 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, adding those services to the available services palette, and selecting services for inclusion in a new model.

Processor-specific mechanisms. Many of the service types Taverna support through its processor plug-ins provide their own methods for service discovery. Both Soaplab and Seqhound services allow introspection over an installation, each installation providing a potentially large number of services. BioMOBY provides its own central registry. This mechanism is entirely extensible, so adding a method is relatively straight-forward.

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 surprisingly well, and gives users considerable flexibility in loading the palette of available services that fit their current requirements.

While these techniques solve the problems of service discovery, Taverna now provides routine access to over 1000 services. This, in turn, increases the importance of the second step in discovery–selection of the appropriate service.

4.2. Service Selection

The service palette (as shown in Figure 7) is grouped according to the service locations, which means that services of the same type are grouped together and colour-coded as in the workflow diagram, and supported by a simple search by name facility. This approach is adequate so long as the number of available services was sufficiently small. However, the number of services now available to Taverna demands more sophisticated search and classification mechanisms based on the properties and semantics of the services.

A common task is to locate a new service based on some conceptual description of the service semantics. For example, the scientist may require a component capable of performing a multiple sequence alignment, an operation whereby similarities between a group of genomic or proteomic sequences may be identified. The task for Taverna is to allow this kind of semantic querying across all service components. Service descriptions need to be enriched by additional metadata as simply scavenging service descriptions does not provide enough information to help users by identifying similar services.

To enable service selection by bioinformaticians, we must represent their view of the services and of the domain [37]. We have investigated a number of different mechanisms to drive the search process, including an RDF-based metadata enriched UDDI registry [38], and a domain ontology [39] described in the W3C Web Ontology Language OWL. The primary difficulty has been the high entry cost. There are research challenges in the construction and maintenance of ontologies to adequately capture task information for a domain as diverse as bioinformatics

Figure 7. The Taverna service palette

let alone all of science. Although the technology is functional and theoretically capable of performing the task of locating services based on a precise description of their function, the expertise required to deploy a registry and register services with appropriate metadata is a significant cost for user communities. This may change in the future if appropriate tools become available to make semantic service registries easy for people to exploit.

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 [40], 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, specifically queries over an RDF data model rather than classifying over an OWL model using a Description Logic reasoner. In fact, the descriptions are modelled using OWL, reasoned over to form a service classification and then deployed into an RDF data model for runtime searching. By limiting the expressivity of service descriptions, we can ease the difficulty of provision of tooling, and increase the chances of uptake of such semantic service discovery technology. The architecture and design of Feta is described in more detail in [9].

4.3. Service composition

Once the appropriate service components have been located, the user requires an interface allowing them to manually compose these services into a workflow. This is a simpler task than that of locating the components. There are a variety of tools available that allow users to work with a graphical representation of the workflow, a form that has a good fit with the underlying technology and should therefore be relatively intuitive even to a non-expert user. Most workflow design packages have adopted a view analogous to electric circuit layout, with services represented as 'chips' with pins for input and output [2, 41]. However from a user interface point of view, this arrangement can become less understandable as complexity increases. If the layout of service components on screen is left under the user's control then the user can tailor the workflow appearance, but this can result in an 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 (see Section 3.1).

4.4. Shim services

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 matches at a conceptual level. Consequently, many of the bioinformatics workflows created in Taverna contain numerous shim services [42] that reconcile the inevitable type mismatches between autonomous third-party services. A shim service is analogous to a shim from the physical worlda thin strip of metal used to align pipes or rails. Likewise, shim services align the inputs and outputs of closely related but otherwise incompatible services. We are currently building libraries of shims for de-referencing identifiers, syntax and semantic translation, mapping, parsing, differencing and so on. WSMO mediators [43] are a similar idea. A serious consideration is how a workflow system can manage shims, which of these shims can be included on demand based on service metadata instead of user intervention, and how we might use planning techniques to automatically discover and invoke shims.

5. Executing and monitoring workflows

In contrast to the design process, which must by definition be exposed to the user, it is desirable to hide as much as possible of the enactment machinery. This is not to suggest that the workflow enactment should appear as a 'black box' process since informing the user of the progress of any given workflow enactment is a critical component. However, details such as federation and fail-over between workflow engines, where possible, should not be exposed. Workflow enactment is a distinct service, whether actually implemented as such or by some software application programming interface.

A critical requirement of this service approach is that workflow invocation behaviour should be independent of the workflow enactment service used. To facilitate peer review of novel results, it is important that other scientists are 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 favoured workflow enactment service.

e-Science is a highly diversified field with respect to the requirements it places on workflow enactment. At one extreme, particle physics experiments produce vast data sets and corresponding computational loads, with a corresponding requirement to deal with the machinery of classical high performance computing (HPC) and networking (HPN). The life sciences domain is characterized by massive variety in terms of data types and the resources to operate on them. Workflow enactment systems in this domain must address different concerns to their HPC counterparts, with a greater emphasis on composing a wide range of services provided by autonomous groups and supporting the exploratory design and use of workflows.

While workflow solutions can in theory handle any given workflow-oriented problem there is a merit in specializing the solutions to the anticipated problem domain. For example, there is a difference in the primary sources of variability addressed by Taverna and Pegasus [44] which deals with more classical HPC problems. In Taverna, the operational services are fixed at particular locations. The WSDL description of a web service includes the specific location of the service endpoint. However, in an open environment, the set of available services will vary. In Pegasus, the "operational services" as applications owned by the experimental scientists, do not vary. However, they are not fixed to a specific location and deciding which services (applications) should run where is a major issue, especially in the context of a changing pool of computational resources. In short, for Taverna the primary source of variability is the set of services available, but those services are tied to fixed computational resources. In contrast for Pegasus, the primary source of variability is the set of computational resources available and how a fixed set of services is best scheduled over those resources. In Taverna, users may modify their workflows because "better" services have become available, or previous ones are no longer on-line. In Pegasus, there is significant workflow modification using the same services in response to the dynamic availability of computational resources.

Workflows in e-Science may also vary hugely in terms of expected invocation duration. Current Taverna workflows vary between two seconds and two weeks of runtime. The potential to invoke workflows over this kind of time scale imposes a strong requirement on any such

system to keep the user informed as to the progress of their enactment, to allow the user to interact with the running workflow in terms of inspecting intermediate results, manually cancelling substructures within the workflow or similar operations, and ideally to do all this from any physical location with preferably minimal network connectivity.

5.1. Fault tolerance and resilience

Any component-based architecture, where the components are not under a single controlling authority, contains components that will fail at some point. It is therefore the responsibility of the aggregating system, here the Freefluo workflow enactment engine, to handle such failures, retaining data integrity and making a reasonable 'best effort' to proceed. Should this be impossible, it is critical that the reporting functionality is sufficient to inform the user of exactly why the workflow was unable to complete.

Failure modes of a workflow system broadly into the following categories: 1. failure of the enactment engine itself; 2. failure of component services and; 3. failure of network fabric. .

5.1.1. Failure of the enactment engine

A single point to which workflows are submitted and from which status reports and results may be obtained, introduces a single point of failure. The possibility of failure increases for long running workflows. Thus we must deal with problems varying from software failures to wider system failures (e.g. loss of electricity supply). Failure of the enactment engine has not been a primary source of the failures reported by users so it has not been a priority in initial Taverna development. Our future implementation plans are to exploit the serialization of the workflow state in XML and a peer-to-peer architecture for the enactment engine service. Using these to replicate state intelligently between enactors within a peer group renders the enactment process almost immune to single point failures at the engine level.

5.1.2. Failure of services

Service failure is more complex and more likely than enactor failure. If a service failed because the machine it runs on is down, it is a candidate to be retried. If the service failed because the input data was invalid, it is inappropriate to try again. In addition to simply retrying the service invocation, it may be possible to locate an alternate service to invoke should the original service fail. In an ideal world, this could be discovered and inserted automatically. However, a great deal of service metadata is needed to make such a choice automatically [37] and users have expressed scepticism at the prospect of automatic selection of services that they would consider equivalent [40]. Except in the simplest cases, the context sensitive nature of many services makes automatic substitution that is acceptable to users extremely difficult. In the life sciences, many web resources may have very similar functionalities, but people distinguish between them based on knowledge of the experimental context and their personal preferences. In reality, only identical services running on an alternate service provider is deemed by our users to be acceptably interchangeable.

For improved fault tolerance when dealing with unreliable services, users specify an *Alternates list* explicitly for any given Scufl processor. Standard fault tolerance techniques such as retry and exponential back out of retry times are implemented. Thus a Scufl processor definition therefore includes: the implementing service, the number or retries, the time between retries, and optionally alternative services to be used if the first-choice service fails.

5.1.3. Failure of network fabric

This is similar to failure of service, but with the additional aspect that it may be possible to probe the network connectivity to a service host using standard internet protocols. An example is an enactor running on a laptop moves away from its wireless network. Current Web Services typically provide no facility to check whether a service is live or not. Currently Taverna does not currently distinguish between failure of the network fabric and failure of the services themselves. The introduction of retries and alternatives has provided improved reliability but we need more sophisticated monitoring to tell whether a long-running service has failed or is just taking longer than expected to complete.

5.2. Reporting

Given the variety of failure modes and potential remedies, 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 non-trivial without the explicit modelling and monitoring of state. The migration of application services to the Grid's Web Service Resource Framework [45] 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 8. 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 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 fifty or more processing components (e.g. Scufl processors), and each of these components can be retrying, using alternate 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.

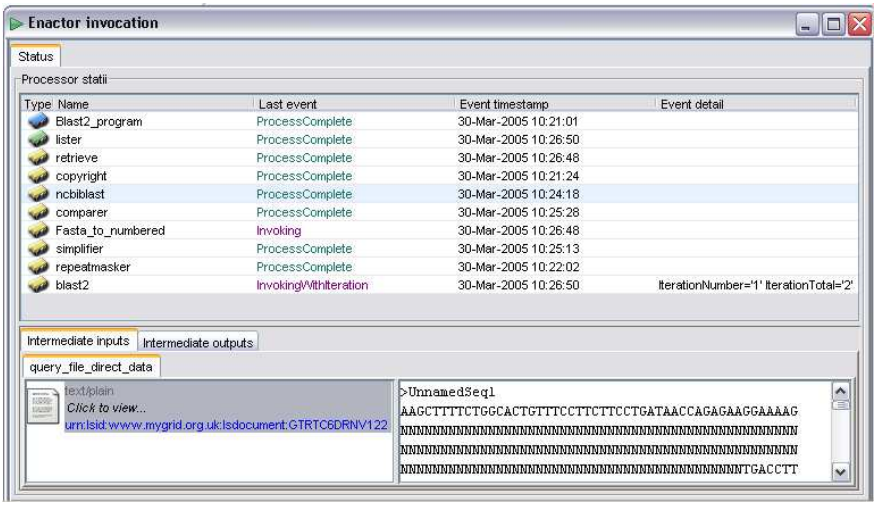


Figure 8. Status information. When running a workflow the Taverna workbench displays status information from the workflow enactor. For each Scufl processor the last event is displayed along with the appropriate time and additional detail if available. This additional detail can include progress through an iteration (e.g. 'item 2 of 6') and retry information. The status information also allows the selection of a processor and viewing the relevant intermediate inputs and outputs. Each data item has been assigned an LSID. More detailed trace information is also available using the 'Process report' tab.

6. Managing and sharing workflows and their results

One of the difficulties created by the automation of the data gathering that Taverna allows, is that it becomes possible to generate very large quantities of data more easily than was previously possible. It becomes critically important to enable the storage of this data in an organised manner. More over, as we are working in the context of scientific *in silico* experimentation, we need to be able to store information about the derivation of data, to potentially enable the rerunning or replication of the experiment.

The extensibility framework enables the storage of data as it is generated by the enactor. Services might store data in third party database. However, data at each point of the workflow is stored within the ^{my}Grid Information Repository (MIR). While this is backed with a relational database, the open world assumption within ^{my}Grid, means that we cannot formally structure the data that is placed in the MIR. It is instead stored with respect to an Information Model based on the experimental process [34].

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 process was used and what parameters

were applied to that process. Thus, in addition to the raw data, we have devised a model of metadata describing the provenance of all aspects of the experiment: the data's derivation path, an audit trail of the service's invoked, the context of the workflow and the knowledge outcomes as a result of its execution. We have adopted to two key technologies:

Life Science Identifiers. The description of the derivation of data necessitates reference to the data sets both inside and outside the control of ^{my}Grid. Bioinformatics has adopted view standards for the identification of data, instead using an *ad hoc* system of *accession numbers*. The recent Life Science Identifer (LSID) standard [35] provides a migration path from the legacy accession numbers, to an identification scheme based on URI's. While ^{my}Grid originally adopted this specification to enable referencing external entities, it has since used it for *all* data being entered into the MIR.

Resource Description Framework (RDF). The MIR has a fixed schema that reflects the common entities used in e-Science experimental life cycle untied 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. Additionally, it provides a well-defined link to the ontology, described in the previous section, enabling specialisation and generalisation of queries. The KAVE (Knowledge Annotation and Verification of Experiments) metadata store has a flexible schema due to its use of RDF. This allows statements to be added outside the fixed schema of the MIR, 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 using RDQL.

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 some novel result 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 metadata 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 KAVE-Taverna-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. It is complicated somewhat by the requirements for fault tolerance and the correspondingly larger range of possible events that may occur. In the event of a failure users should expect access to sufficient information so that they can comprehend the failure and take appropriate action.

Making all results, indeed all relevant experimental data, identifiable by LSIDs, and using RDF statements to record metadata about those data, is not specific to a workflow system. Any applications that scientists use to create *in silico* data, for example, a laboratory information management system, could also record provenance information in the same way and allow users

to query it without having to know the system that produced it. The effective integration of provenance information that can be automatically generated, and additional metadata, for example a scientists conclusions based on the experimental results, that is supplied through manual annotation, is part of our current work. Further details of provenance in ^{my}Grid can be found in [12]. The use of OWL and RDF for service selection and provenance logging makes ^{my}Grid a pioneer of the so called Semantic Grid movement [46].

6.1. Data integration and visualisation

^{my}Grid has mechanisms for recording the intermediate and final outcomes of services executed by a workflow. Using LSIDs and the provenance model in RDF we link together these results. Our open world model, however, means that we do not automatically integrate these results. Thus Taverna supports application interoperation but not application integration. Integration is special to each collection of chained services and the purpose of the workflow. Integration steps require a Semantic Mediation mechanism mapping data to a common schema of the kind adopted by the Kepler [28] toolset in the SEEK and GEON projects. Other techniques include the use of Shims to incrementally populate data objects that integrate data across applications, and closing off the open world in situations where it is worth the effort of building a more strongly typed model across a small number of applications and a static yet permanent workflow. An interesting research task is the use of the RDF provenance metadata to provide a more generic solution that can be re-used between related workflows;

7. Related work

The Life Sciences is a field where there are many scientists, unfamiliar with middleware technologies, who want an easy way of rapidly pulling together third-party services into prototypical *in silico* experiments. In contrast, in other fields, such as physics and astronomy, the prime scenario involves carefully designed workflows linking applications. The goal is to exploit computational grid resources to enable *in silico* experiments that were previously impractical due to resource constraints. It is unsurprising that the range of possible scenarios and the emerging nature of the technology has led to a variety of different workflow systems.

7.1. Life sciences perspective

The diversity of life sciences research leads to significant interest in techniques for combining resources. One abstract workflow familiar to most in biology is the annotation pipeline. This consists of an initial input, usually a DNA or protein sequence, and then a series of analyses that gives as much information as possible about the input sequence, based on other experiments and literature which refer to that sequence or biologically similar ones. It is in essence exploratory: attempting to answer the question, "What is known about this sequence?" in an environment of rapidly progressing scientific knowledge. A variety of workflow systems for bioinformatics have been developed; research systems include the PLAN programmable integrator [47], structural genomic workflows [48], BioOpera [49], and commercial systems

include Incogen VIBE (Visually Integrated Bioinformatics Environment) [50], TurboWorx [51] and PipelinePilot [52]. The mine-it bioinformatics modelling tool [53] has taken the same approach, not for sequence data, but for the advanced analysis of gene-expression and micro-array data. In contrast to Taverna these have typically been closed systems designed to build workflows from a fixed suite of analysis tools with specialist programming needed to incorporate new ones. In a rapidly developing environment where heterogeneous data is the norm, and many of the most respected tools use their own peculiar input and output formats [18, 54], many scientists are reluctant to risk becoming locked into an ageing analysis suite.

7.2. Scientific workflow systems

Table II provides a brief comparison of several representative scientific workflow systems. These 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 proven ability to link together a significant range of autonomous bioinformatics services and its flexibility, particularly in terms of the metadata, including LSIDs and provenance information, generated to aid scientists manage and share workflow results.

The Kepler workflow system [2] has been developed for scientists with a range of interests in and is built on Ptolemy II, a mature application from the electrical engineering domain [55]. Like Taverna, Kepler is dataflow oriented, with the core description being the processing of data through a set of connected actors (processing steps). Kepler's strengths include its mature 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 [41] system was originally developed as a data analysis problem-solving environment for a gravitational wave detection project. It is also dataflow oriented, and is particularly strong at exploiting the structure available in signal processing data. It is aimed at CPU intensive engineering and scientific applications, primarily allowing scientists to compose their local applications and distribute the computation on a set of Triana servers. In contrast, DiscoveryNet, another UK e-Science pilot project, uses a proprietary workflow engine. DiscoveryNet scientific workflows are used to allow scientists to plan, manage, share and execute complex knowledge discovery and data analysis procedures [1]. In DiscoveryNet all services are wrapped to conform to a standard tabular data model, which gives benefits in exploiting established knowledge discovery techniques.

The Geodise system [56] is similar to Taverna in its emphasis on fitting with the established working practices of its target users, engineers, and having a language matched to their problem domain, in its case MatLab. It differs in its focus on engineering design search and optimisation, where workflow runs use different input parameters and are compared through an explicit "objective function". Geodise's motivation for exploiting a Grid is the computational demands of established engineering applications, while Taverna's is the easy access and incorporation of third-party services into *in silico* experiments.

The Pegasus system [44] abstracts from the detail of workflow design. The user provides a workflow template and artificial intelligence planning techniques are used to deal with the moving of data and execution of applications on a heterogeneous and changing set of computational resources. The emphasis is on the planning and scheduling the execution of

Table II. A brief comparison of Grid workflow systems

	Taverna	Kepler	DiscoveryNet	Geodise	Triana	Pegasus
Scientific domains	biology	biology, ecology Geology	biology, chemistry	engineering	astronomy	astronomy, biology
Prime strengths	usability, low startup cost	flexibility	usability, database integration	engineers know MatLab	usability, P2P support	automated planning
Domain resources remote	web services and domain specific	web services	web services	No	web services	application programs
Domain resources local	Java resources	Java and PERL resources	wrapped applications	MatLab	wrapped Java resources	n/a
Adding resources	find and use	find, wrap, deploy and use	find, wrap and use	find, wrap and use	find, wrap, deploy and use	find, deploy and use
Linking disparate resources	shims	shims, common data model	common data model	n/a	common data model	shim applications
Prime data types	string	XML, string	relational table	numerical	numerical?	files
Workflow language	Scufl	MOML	DPML	MatLab scripts	Java	Chimera's VDL
Software required	Java	Java, (Ptolemy)	commercial software	MatLab (commercial)	Java	Condor DAGMan
Open source	yes	yes	no	no	yes	yes

large numbers of related jobs on a computational Grid, where users describe the results set that they require and there are alternative strategies for calculating that result set.

The variety of different scientific workflow systems comes from the variety of different types of e-Science. At the current time, there are no established standards and the key factor in the adoption of particular systems is their effectiveness for specific user communities.

The use of workflows for "programming in the large" to compose web services has led to significant interest in a standard workflow language within the web services stack. There is a clear candidate in BPEL [57] which has industry support from both IBM and Microsoft, as it was created through the agreed merge of their earlier web service composition languages WSFL [58] and XLANG [59] respectively. (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.)

One reason why Taverna workflows use their own proprietory language Scufl rather than a potential standard is historical. In the initial stages of the ^{my}Grid project in 2001 BPEL did not exist; it was still the subject of confidential discussions between IBM and Microsoft. There were certainly no open-source tools easily available. The more significant reason is conceptual. We did initially try to use IBM's WSFL language, but this did not match how our target users wanted to describe their *in silico* experiments [60]. WSFL forced users to think in terms of web service ports and messages rather than passing data between bio-services. The mismatch was very significant with stateful services that had their own interaction protocol such as Soaplab services. Scufl processors are a direct result of this experience.

The issues of experience of using the BPEL and open-source tools are still present. At the GGF10 workshop [5] there was clearly very limited experience of using BPEL for scientific workflows. While open-source tools now exist, the OASIS WSBPEL site also lists the large number of language issues that are unresolved, so differences between tools are inevitable.

Whether BPEL, or some higher-level that translates to BPEL, will become a standard workflow language in the life science domain is an open question. The power of BPEL is that you can program your orchestration of web services in detail. The cost of this power is that features such as iteration, provided implicitly in Scufl, have to be programmed, and all services must be web services. The focus of BPEL, and most business-oriented workflow languages, is control flow. Extensive research on workflow control patterns has shown that all languages have limitations in terms of what can be easily expressed [61]. The ease of mapping a dataflow approach, which has proved an appropriate abstraction for many scientists, to BPEL's constructs needs further investigation. For Taverna users a migration path, so that workflows prototyped in Scufl could be transformed to BPEL for more "heavyweight" use, would be attractive.

8. Lessons Learnt

^{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. The resulting Taverna software package currently has some 500 installations. Our experience with local users, such as those using it for the Williams Beuren analyses, indicates initial success. Users with expert scientific knowledge, but little or no knowledge of middleware, have been able to write workflows that automate their commonly performed bioinformatics analyses. They do so in a fraction of the time of the previous manual methods, have been verified to produce valid scientific results, and in fact are now progressing on to find novel results. They do this using a representation that is straightforward to communicate to other scientists.

8.1. Abstraction is key

One of the keys to its initial success has been the level of abstraction that the workflow environment allows users to operate at. The structure of a Scufl workflow corresponds well

to the scientist's conceptualisation of a bioinformatics task. Each resource being orchestrated corresponds to a bioinformatics application or access to a database recognised by the scientist. The links between resources are specified in terms of dataflow, again corresponding to the way many scientists view a complex task as an analysis pipeline, with data flowing through that pipeline. Complex control flow constructs such as iteration over data sets are hidden from the user through the use of implicit but configurable iteration strategies in the Execution Layer. Complex orchestration of individual application services is achieved through the use of processor plug-ins. A viable option because whole sets of services follow the same invocation pattern.

8.2. Finding the limits of abstraction

The abstraction at the Application layer breaks down if the user wants to take control of finer grained interaction with individual services. For example, a biological simulation service may run for several days to weeks, and require monitoring and steering. Our current solution is to handle this service specific interaction independently of the workflow specification. More complex control flow involving complex conditional branches, or complex parallel iteration through multiple data sets also breaks the abstraction. As users become more familiar with the workflow environment they have been tempted to push the current representation to perform more complex tasks. To some extent the environment has been augmented to support some of these requirements. For example, simple iteration strategies can now be extended and there is support within the workflow for a form of conditional branching. There is however a tension between keeping the language simple for the majority of users, and supporting more advanced functionality. In some cases we advise that complex business logic should not form part of the workflow specification, but be encapsulated as a service which is then *orchestrated* by the workflow. Multiple levels of abstraction can also make it more difficult for the user to trace the origin of a failure because each step in the workflow may actually encompass multiple service interactions, any one of which may have been the point of failure. In many respects this does not become an issue because the scientific user does not have the skills or authority to resolve persistent service failures anyway. This role must be left to service or ^{my}Grid administrators who are able to interpret more fine grained debugging information from both the workflow enactor and service installation.

An alternative approach to implement the abstraction, as taken by other projects such as Pegasus, would be to compile the abstraction specified in Scufl to a concrete specification. This would allow user interaction at this more concrete level during workflow enactment.

8.3. Lowering the cost of entry

By coupling a simple workflow language with a straight-forward workflow authoring tool, scientific workflows can be authored quickly given the ready availability of services. The rate limiting step is the discovery of existing services or making applications available as a service. ^{my}Grid has mitigated this by (i) treating data as opaque to the workflow system, and so no effort is needed making a service conform to a common data model; and (ii) providing Soaplab as a framework to deploy command line applications as services. This approach has meant

1000+ services are now available that provide broad coverage of functionality, especially in the commonly performed sequence analysis arena. However, two issues remain (i) the many small bioinformatics groups that provide analysis tools via the web, have yet to provide Web Service interfaces to their applications (TwinScan and Genscan gene prediction tools are an example). Neither does their licensing accommodate others such as ^{my}Grid deploying such a service for general use on their behalf; (ii) these services have not been designed to work together and so Shim services have to be inserted to ensure flow of compatible data. Until a significant library of such Shims exists, the authoring of each workflow must be accompanied by their co-development.

8.4. Supporting the wider context

Much of the initial benefit of the workflow system has been based on this ability of scientists to rapidly write workflows to automate mundane tasks. However, we are recognising the longer term success of the workflow system will depend on how effectively users cope with the large amount of services now available: how easily they can find and reuse other scientists workflows, and whether the use of a workflow system makes managing analyses results easier not harder. The core workflow system cannot hope to support all these functions of the wider e-Science process. By providing an extensible framework to the core system, and developing plug-ins we are able to support the workflow life cycle solely within the Taverna workbench. Feta supports user centred discovery of both workflows and services in a UDDI registry, GRIMOIRES, that can then be rapidly incorporated into a nascent workflow. However, given the rapid expansion of available services, the number of structured descriptions of these services lags behind the number of services. Involvement of service providers will be essential in closing this gap. The novelty of scientific workflows means users are unsure how much they should guard their value. Although, a key benefit is perceived to be the ease by which they can be shared and re-used, it is not currently clear how willing scientists will be to actually share these resources widely, exactly because they could be re-used by a competitor so rapidly. Our users quickly realised that the know-how embodied in the workflow itself was valuable intellectual property and some demanded a copyright service be incorporated in all workflows.

8.5. Integrating and visualising results

Taverna and the ^{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 visualisation 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 a significant time to analyse 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 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 re-used 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 know to orchestrate a limited number of services and will be permanent in nature, so it is worth the effort of building a more strongly typed model.

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), new applications become available, as do novel service frameworks for deploying them. By working closely with both our users, service providers, and other workflow projects, we continue to extend the basic core functionality to fulfil a wide range of uses.

9. Acknowledgements

This work is supported by the UK e-Science programme EPSRC GR/R67743. The authors would like to acknowledge the ^{my}Grid team. Hannah Tipney developed workflows for investigation of 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 Network Inference.

REFERENCES

1. A. Rowe, D. Kalaitzopoulos, M. Osmond, M. Ghanem, and Y Guo. The Discovery Net system for high throughput bioinformatics. *Bioinformatics*, 19(90001):225i–231, 2003.
2. I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock. Kepler: Towards a Grid-Enabled system for scientific workflows. In *Workflow in Grid Systems Workshop in GGF10*, Berlin, March 2004.
3. Ewa Deelman, James Blythe, Yolanda Gil, and Carl Kesselman. Workflow management in GriPhyN. In J. Nabrzyski, J. Schopf, and J. Weglarz, editors, *Grid Resource Management*. Kluwer, 2003.
4. Jihie Kim, Yolanda Gil, and Marc Spraragen. A knowledge-based approach to interactive workflow composition. In *14th International Conference on Automatic Planning and Scheduling (ICAPS 04)*, Whistler, Canada, 2004.
5. GGF. GGF10 workflow workshop. `http://www.extreme.indiana.edu/groc/ggf10-ww/`, March 2004.
6. UK National e Science Centre. NeSC workshop on e-Science workflow services. `http://www.nesc.ac.uk/action/esi/contribution.cfm?Title=303`, December 2003.
7. J Devereux, P Haeberli, and O Smithies. A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Research*, 12:387395, 1984.
8. P Rice, I Longden, and A Bleasby. EMBOSS: the european molecular biology open software suite. *Trends Genetics*, 16:276277, 2000.
9. Phillip Lord, Pinar Alper, Chris Wroe, and Carole Goble. Feta: A light-weight architecture for user oriented semantic service discovery. In *European Semantic Web Conference*, 2005. Accepted for Publication.
10. Peter Li, Keith Hayward, Claire Jennings, Kate Owen, Tom Oinn, Robert Stevens, Simon Pearce, and Anil Wipat. Association of variations in I kappa B-epsilon with Graves disease using classical and ^{my}Grid methodologies. In Simon J Cox, editor, *Proceedings of UK e-Science programme All Hands Meeting*, pages 287–293. EPSRC, September 2004.
11. R.D. Stevens, H.J. Tipney, C.J. Wroe, T.M. Oinn, M. Senger, P.W. Lord, C.A. Goble, A. Brass, and M. Tassabehji. Exploring Williams Beuren Syndrome Using ^{my}Grid. In *Bioinformatics*, volume 20, pages i303–310, 2004. Intelligent Systems for Molecular Biology (ISMB) 2004.

12. Jun Zhao, Chris Wroe, Carole Goble, Robert Stevens, Dennis Quan, and Mark Greenwood. Using Semantic Web Technologies for Representing e-Science Provenance . In *3rd International Semantic Web Conference (ISWC2004)*, 2004. To appear.
13. M.Nedim Alpdemir, Arijit Mukherjee, Norman W. Paton, Alvaro A.A. Fernandes, Paul Watson, Kevin Glover, Chris Greenhalgh, Tom Oinn, and Hannah Tipney. Contextualised workflow execution in ^{my}Grid. In P.M.A. Sloot, A.G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *Advances in Grid Computing: EGC 2005*, 2005.
14. The Genome International Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, February 2001.
15. Dennis A. Benson, Ilene Karsch-Mizrachi, David J. Lipman, James Ostell, and David L. Wheeler. GenBank. *Nucleic Acids Research*, 33:D34–D38, 2005. Database issue.
16. Chris Burge and Samuel Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997. See also `http://genes.mit.edu/GENSCAN.html`.
17. I. Korf, P. Flicek, D. Duan, and M.R. Brent. Integrating genomic homology into gene structure prediction. *Bioinformatics*, 17:S140–148, 2001. See also `http://genes.cs.wustl.edu/`.
18. Lincoln Stein. Creating a bioinformatics nation. *Nature*, 417:119 – 120, 2002.
19. David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture, W3C Working Group Note. `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`, November 2004.
20. Lichun Wang, Jean-Jack Riethoven, and Alan Robinson. XEMBL: distributing EMBL data in XML format. *Bioinformatics*, 18(8):1147–1148, 2002.
21. Martin Senger. Bibliographic query service. `http://industry.ebi.ac.uk/openBQS/`, 2002.
22. S Miyazaki, H Sugawara, K Ikeo, T Gojobori, and Y Tateno. DDBJ in the stream of various biological data. *Nucl. Acids Res.*, 32(90001):D31–34, 2004.
23. M. D. Wilkinson, D. Gessler, A. Farmer, and L. Stein. The BioMOBY Project Explores Open-Source, Simple, Extensible Protocols for Enabling Biological Database Interoperability. *Proc Virt Conf Genom and Bioinf*, 3:16–26, 2003.
24. J Dana Eckart and Bruno Sobral. A life scientists gateway to distributed data management and computing: The pathport/toolbus framework. *OMICS: A Journal of Integrative Biology*, 7(1):79–88, 2003.
25. M Senger, P Rice, and T Oinn. Soaplab - a unified sesame door to analysis tools. In *Proceedings UK OST e-Science 2nd All Hands Meeting*, September 2003.
26. Katerina Michalickova, Gary Bader, Michel Dumontier, Hao Lieu, Doron Betel, Ruth Isserlin, and Christopher Hogue. SeqHound: biological sequence and structure database as a platform for bioinformatics research. *BMC Bioinformatics*, 3(1):32, 2002.
27. Roy T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.
28. Shawn Bowers and Bertram Ludscher. An ontology-driven framework for data transformation in scientific workflows. In *International Workshop on Data Integration in the Life Sciences (DILS'04)*, Leipzig, Germany, 2004. Springer.
29. Ewa Deelmana, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, , Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus : Mapping scientific workflows onto the Grid. In *Across Grids Conference*, Nicosia, Cyprus, 2004.
30. Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, 2000.
31. William Sanchez, Brian Gilman, Manav Kher, Steven Lagou, and Peter Covitz. caGRID White Paper. `http://cabig.nci.nih.gov/guidelines_documentation/caGRIDWhitepaper.pdf`, July 2004.
32. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.
33. Rob Pike and Dennis M. Ritchie. The Styx- Architecture for distributed systems. *Bell Labs Technical Journal*, 4(2):146–152, 1999.
34. Nick Sharman, Nedim Alpdemir, Justin Ferris, Mark Greenwood, Peter Li, and Chris Wroe. The ^{my}Grid Information Model. In Simon J Cox, editor, *Proceedings of UK e-Science programme All Hands Meeting*, pages 287–293. EPSRC, September 2004.
35. Tim Clark, Sean Martin, and Ted Liefeld. Globally distributed object identification for biological knowledgebases. *Briefings in Bioinformatics*, 5(1):59–70, 2004.
36. Universal Description Discovery and Integration (UDDI) Technical Whitepaper. `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`.

37. C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, and L. Moreau. Automating experiments using semantic data on a bioinformatics grid. *IEEE Intelligent Systems*, 19(1):48–55, 2004.
38. Phillip Lord, Chris Wroe, Robert Stevens, Carole Goble, Simon Miles, Luc Moreau, Keith Decker, Terry Payne, and Juri Papay. Semantic and Personalised Service Discovery. In W. K. Cheung and Y. Ye, editors, *WI/IAT 2003 Workshop on Knowledge Grid and Grid Intelligence*, pages 100–107, Halifax, Canada, October 2003.
39. Chris Wroe, Robert Stevens, Carole Goble, Angus Roberts, and Mark Greenwood. A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *The International Journal of Cooperative Information Systems*, 12(2):597–624, 2003.
40. Phillip Lord, Sean Bechhofer, Mark D. Wilkinson, Gary Schiltz, Damian Gessler, Duncan Hull, Carole Goble, and Lincoln Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. In *International Semantic Web Conference*, 2004. Accepted For Publication.
41. M. Shields and I. Taylor. Programming Scientific and Distributed Workflow with Triana Services. In *Workflow in Grid Systems Workshop in GGF10*, Berlin, March 2004.
42. Duncan Hull, Robert Stevens, Phillip Lord, Chris Wroe, and Carole Goble. Treating shimantic web syndrome with ontologies. In *First AKT workshop on Semantic Web Services (AKT-SWS04) KMi, The Open University, Milton Keynes, UK. December 8, 2004*, 2004. Workshop proceedings CEUR-WS.org ISSN:1613-0073.
43. Emilia Cimpian, Adrian Mocan, Dumitru Roman, Francois Scharffe, and James Scicluna. WSMO Mediators. `http://www.wsmo.org/TR/d29/v0.1/`, March 2005.
44. Y. Gil, E. Deelman, J. Blythe, C. Kessleman, and H. Tangmunarunkit. Artificial Intelligence and Grids: Workflow Planning and Beyond. *IEEE Intelligent Systems special issue on e-science*, 19(1):26–33, 2004.
45. Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. The WS-Resource framework, March 2004.
46. Carole Goble and David De Roure. The Semantic Grid: Myth busting and bridge building. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)*, Valencia, Spain, 2004.
47. M. Chagoyen, M.E. Kurul, P.A. De-Alarcn, J.M. Carazo, and A. Gupta. Designing and executing scientific workflows with a programmable integrator. *Bioinformatics*, 20(13):2092–2100, 2004.
48. M. Cavalcanti, F. Baio, S. Rssle, P. Bisch, R. Targino, P. Pires, M. Campos, and M. Mattoso. Structural genomic workflows supported by Web Services. In *Proceedings of the 14th International Conference on Database and Expert Systems Applications (DEXA 2003) International Workshop on Biological Data Management (BIDM'03)*, pages 45–49, Prague, Czech Republic, September 2003.
49. BioOpera. Process Support for Bioinformatics. `http://ikplab12.inf.ethz.ch:8888/bioopera/website/main.html`, 04 June 2004.
50. INCOGEN VIBE. (Visual Integrated Bioinformatics Environment). `http://www.incogen.com/VIBE`, 04 June 2004.
51. TurboWorx. Enterprise. `http://www.turboworx.com`, 04 June 2004.
52. Pipeline Pilot. `http://www.scitegic.com/products_services/pipeline_pilot.htm`, 03 June 2004.
53. S. Frank, J. Moore, and R. Eils. A question of scale: Bringing an existing bio-science workflow engine to the grid. In *Workflow in Grid Systems Workshop in GGF10*, Berlin, March 2004.
54. R.D. Stevens, C.A. Goble, P. Baker, and A. Brass. A Classification of Tasks in Bioinformatics. *Bioinformatics*, 17(2):180–188, 2001.
55. Ptolemy Project. PtolemyII. `http://ptolemy.eecs.berkley.edu`, 11 May 2004.
56. Z. Jiao, J.L. Wason, W. Song, F. Xu, H. Eres, A.J. Keane, and S.J. Cox. Databases, Workflows and the Grid in a Service Oriented Environment. In *Euro-Par 2004, Parallel Processing*, pages 972–979, Pisa, August 2004.
57. Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services (Version 1.1). `ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`, May 2003.
58. F. Leymann. Web Services Flow Language (WSFL 1.0). `http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, May 2001.
59. S. Thatte. XLANG Web Services for Business Process Design. `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`, 2001.
60. Matthew Addis, Justin Ferris, Mark Greenwood, Darren Marvin, Peter Li, Tom Oinn, and Anil Wipat. Experiences with escience workflow specification and enactment in bioinformatics. In *Proceedings of UK e-Science All Hands Meeting*, pages 459–467, 2003.

61. W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.