# Automating Experiments Using Semantic Data on a Bioinformatics Grid

**Chris Wroe, Carole Goble, Mark Greenwood, and Phillip Lord,** *University of Manchester*

**Simon Miles, Juri Papay, Terry Payne, and Luc Moreau,** *University of Southampton*

***Services form the key component of the workflows used in the myGrid project to represent bioinformatics experiments. Abstraction allows greater portability, but choosing a service to execute depends on a large amount of additional metadata.***

The transition from laboratory science to *in silico* e-science has facilitated a paradigmatic shift in the way we conduct modern science. We can use computationally based analytical models to simulate and investigate scientific questions such as those posed by high-energy physics and bioinformatics, yielding high-quality results and discoveries at an unprecedented rate. However, while experimental media have changed, the scientific methodologies and processes we choose for conducting experiments are still relevant. As in the lab environment, experimental methodology requires samples (or in this case, data) to undergo several processing stages. The staging of operations is what constitutes the *in silico* experimental process.

Initial bioinformatics experiments typically required passing data through several programs in sequence. We'd format the data to conform to application-dependent file formats and then pass it through selected scientific applications or *services*, which would yield a handful of results or generate new data. This new data would in turn require reformatting and passing through other services. Often, a bioinformatician would have to manually transfer results between services by noting these values and rekeying them into a new interface or by cutting and pasting. Although problematic and error prone, this approach facilitated scientific exploration through experimentation with different hypotheses using different services. This service-oriented approach underpins emerging technologies such as Web Services and the Grid.

The use of *workflows* formalizes earlier ad hoc approaches for representing experimental methodology. We can represent the stages of *in silico* experiments formally as a set of services to invoke. Although this formalization can simplify the representation of experimental methodology, referring to specific services limits the utility, portability, and scalability of such workflows. They're prone to the removal or modification of any of the services on which they depend. We can't readily share workflows with colleagues or execute them on other computer infrastructures unless the same services exist on the new infrastructure. Even in an open, shared-services environment, several scientists invoking the same workflow would result in service contention, because each workflow would require the same instances. Additionally, social and human factors add further constraints: to preserve their intellectual property, scientists prefer to publish their experiments' structure while keeping the invoked service instances' details private.

By abstracting the workflows, we can construct workflow templates representing the type or class of service to invoke at each experimental stage, without specifying which instance of the service should be used. To use a template, we instantiate the abstracted service representations according to the available services and then manage the data flow appropriately to ensure interoperation between the services. In this article, we address how to use workflow resolution to perform such instantiation through service discovery and semantic reasoning. We'll also consider whether to perform resolution before or during workflow execution and whether resolution should be manual or automated. These choices have implications on the metadata's nature and the workflow execution engine's capabilities.

Several projects have recognized the need for abstract scientific workflows and processes to produce executable workflows from them (see the "Related Work" sidebar for more information). myGrid, a UK-based e-science pilot project, is implementing an environment to support *in silico* experimentation guided by existing bioinformatics scenarios, including investigation of the genetic basis of Grave's disease.[1] By describing and registering services and by writing and executing workflows, we've reiterated the need for abstraction while uncovering significant complexity beneath an apparently simple story. This is particularly the case in dealing with the surprising diversity among apparently identical services. Using the Basic Local Alignment Search Tool, or BLAST, as an example (see the "BLAST" sidebar), we can demonstrate this diversity and the mechanisms of *workflow harmonization* necessary to address it.

## Related Work

Several other groups have found the need for more abstract workflows to decouple the scientific process from the Grid's complexities and rapid turnover.

The Planning for Execution in Grids project, or Pegasus (http://pegasus.isi.edu) is part of the Grid Physics Network (GriPhyN) project. Like myGrid, it too uses the notion of abstract workflows, but here automation takes a much more prominent role at all stages in the workflow life cycle. Rather than deferring to the scientist to break down the goal into smaller tasks, Pegasus uses AI planning to construct a strategy for processing input data into a desired data product by the most efficient means.[1]

The Scientific Data Management Center project has developed an environment in which abstract scientific workflows can be created and compiled into an executable workflow through an abstract-as-view mapping. That is, abstract tasks are created as a view over executable tasks.[2]

### References

1. E. Deelman et al., "Mapping Abstract Complex Workflows onto Grid Environments," *J. Grid Computing*, vol. 1, no. 1, Jan. 2003, pp. 25–39.
2. B. Ludascher, I. Altintas, and A. Gupta, "Compiling Abstract Scientific Workflows into Web Service Workflows," *Proc. 15th Int'l Conf. Scientific and Statistical Database Management* (SSDBM 03), July 2003, IEEE CS Press, pp. 251–254.

## The bioinformatics experimental life cycle

Current practice in laboratory e-science can be described as an *experimental life cycle*. The scientist begins with a high-level goal to test a hypothesis or integrate new discoveries with existing knowledge. Both before and during an experiment, the scientist must make decisions about the granularity of each subtask in the experimental design, thus ensuring that each task is unambiguous and realizable (that is, some service actually exists that might achieve this task). The decisions involve decomposing high-level goals into simpler tasks and choosing the most appropriate class of service to accomplish each task. We're developing a methodology and the software environment to help scientists use the Grid's resources in performing these *in silico* experiments.

To illustrate this, consider a scenario involving a hypothesis about whether a novel protein found in diseased tissue could have a causative role in that disease. The first step consists of finding out whether biologists have designed these types of *in silico* experiments before. A centralized service registry categorizes previously published experimental designs and services using associated metadata (see the "Metadata" sidebar on page 50). If no existing design matches the desired experiment, the biologist will instead search or browse for designs that possess some features relevant to the current goal, including those that operate on protein data and those that infer functional information. Such informal registries already exist on the Web covering both laboratory and *in silico* design (see, for example, www.protocol-online.org/prot/Bioinformatics). myGrid is also developing registries that provide more structured metadata descriptions to support more precise searches.[2]

Each experimental goal can be accomplished by a sequence of tasks. For our example, we might use this sequence:

1. Find similar proteins that have already been studied.
2. Retrieve functional information about those proteins.
3. Collate these functions.
4. Search the biomedical literature to find associations between these functions and the disease.

Initially, the scientist must choose credible methods to accomplish each task. The registry provides a classification of services. Browsing this classification would reveal that several classes of sequence similarity service exist, one of which uses BLAST. This tool is available from numerous locations, and the most appropriate one to use will vary over time. By using service classes, the scientist is

## BLAST

Biologists often begin with a newly discovered sequence and must then deduce its role by comparing it with databases of already characterized sequences and finding similar ones. BLAST, the Basic Local Alignment Search Tool, is the most widely used tool for performing these similarity searches.[1] BLAST is highly parameterizable, able to search over many databases with many types of sequences. In fact, BLAST has several subprograms that are used for specific tasks. For example, BLASTP is used to search protein databases for similar protein sequences, and BLASTN is used to search nucleotide databases for similar nucleotide sequences. The user chooses the appropriate subprogram.

Sequence similarity searches are highly computationally intensive but amenable to parallelization. At least one grid solutions provider, AVAKI, is working with BioGrid projects to allow grid-based access to BLAST.

### Reference

1. S.F. Altschul et al., "Basic Local Alignment Search Tool," *J. Molecular Biology*, vol. 215, no. 3, 5 Oct. 1990, pp. 403–410.

## Metadata

Rich, structured metadata is a key feature of the Semantic Grid and therefore pervades myGrid's design. It provides machine-interpretable descriptions to drive discovery and interoperation of resources such as services, workflows, data, notifications, and people. We can gain far more by searching over the relations between resources than by finding resources in isolation—for example, "Who has published a paper with a topic related to Graves' disease?" By using common underlying technologies such as RDF,[1] the RDQL query language,[2] and OWL[3] to explicitly represent structured metadata, we aim to provide uniform access to this often underutilized and implicit body of links.

In the case of service metadata, service providers use various ways to describe their services, access policies, contract negotiation details, and so on.[4–7] Although these descriptions share many features, there's little agreement on how to incorporate them into a unified model. myGrid researchers at the University of Southampton have used RDF and RDQL to implement a framework for unifying various service metadata models and so to simplify metadata querying.[8]

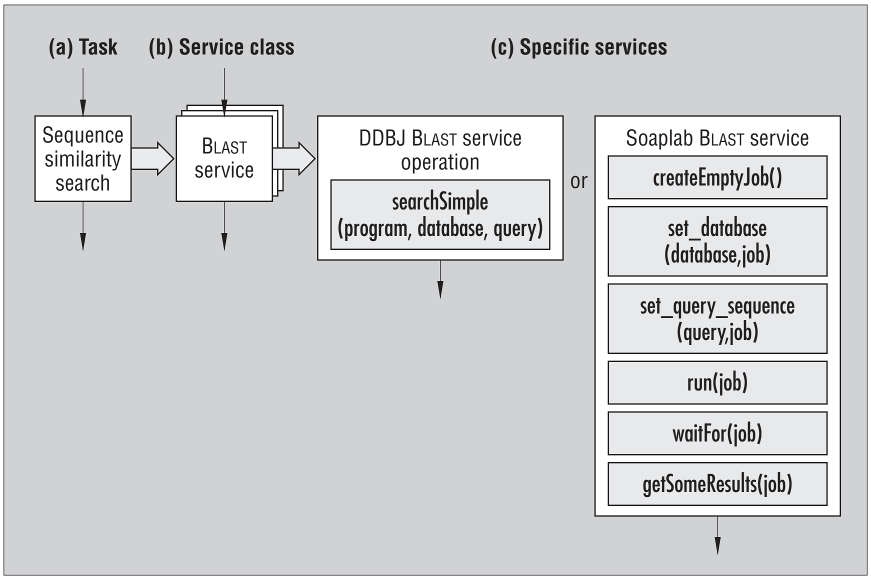
The provision of metadata should be open to both service providers and users who can give extra information on that service—for example, a statement of recommendation or otherwise. The latter type of metadata, called *third-party metadata*, has interesting related problems in that publishing such metadata in public registries might not be reasonable. In myGrid, we provide local personalized views over public registries so that third-party metadata can be locally stored and used.

### References

1. G. Klyne and J.J. Carroll, *Resource Description Framework (RDF): Concepts and Abstract Syntax*, W3C Working Draft, 2003, www.w3.org/TR/2003/PR-rdf-concepts-20031215.
2. L. Miller, A. Seaborne, and A. Reggiori, "Three Implementations of SquishQL, a Simple RDF Query Language," *The Semantic Web: ISWC 2002, Proc. 1st Int'l Semantic Web Conf.*, LNCS 2342, Springer-Verlag, 2002, pp. 423–435.
3. D.L. McGuinness and F. van Harmelen, *OWL Web Ontology Language Overview*, W3C Working Draft, 2003, www.w3.org/TR/2003/PR-owl-features-20031215.
4. A. Ankolekar et al., "DAML-S: Web Service Description for the Semantic Web," *The Semantic Web: ISWC 2002, Proc. 1st Int'l Semantic Web Conf.*, LNCS 2342, Springer-Verlag, 2002, pp. 348–363.
5. *UDDI Version 2.04 API Specification*, OASIS UDDI Specification Tech. Committee, July 2002, http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm.
6. R. Chinnici et al., *Web Services Description Language (WSDL) Version 2.0, Part 1: Core Language*, W3C Working Draft, 10 Nov. 2003, www.w3.org/TR/2003/WD-wsdl20-20031110.
7. M.D. Wilkinson and M. Links, "Biomoby: An Open-Source Biological Web Services Proposal," *Briefings in Bioinformatics*, vol. 3, no. 4, Dec. 2002, pp. 331–341.
8. S. Miles et al., "Personalized Grid Service Discovery," *IEE Proc.*, vol. 150, no. 4, Aug. 2003, pp. 252–256.

committing to the *way* a task is performed, not the particular *instance* of a service used to perform it. If the scientist wants to rerun the experiment later, the workflow template can be resolved again to account for changes in BLAST service availability. If the scientist wants to share the design, others can personalize the choice of BLAST service to suit their local preferences. Because resolution doesn't affect the workflow's scientific design, it's a credible target for automation.

However, on the Web (and the Grid), heterogeneous services provide similar capabilities but with varying configurations. We've found that even if two services provide access to exactly the same application, there's no guarantee we can use them in the same way. This is certainly the case with our example service providing access to BLAST. Several have been implemented, including one by DNA Databank of Japan, or DDBJ (www.xml.nig.ac.jp/wsdl/index.jsp), and another within Soaplab (http://industry.ebi.ac.uk/soaplab), developed by the European Bioinformatics Institute as part of the myGrid project. To accommodate this diversity, we need an additional stage in which we modify the experiment's low-level design to accommodate the specifics of the service instances to be executed. We call this process *workflow harmonization*. Figure 1 shows how the initial task of finding similar sequences progresses through several tiers of detail during the experiment's life cycle until



(a) Task
(b) Service class
(c) Specific services
Sequence similarity search
BLAST service
DDBJ BLAST service operation
searchSimple (program, database, query)
or
Soaplab BLAST service
createEmptyJob()
set_database (database,job)
set_query_sequence (query,job)
run(job)
waitFor(job)
getSomeResults(job)

**Figure 1. Tiered specification of a single step in an *in silico* experimental design: (a) Specify the high-level task. (b) Specify a class of service that accomplishes that task, in this case the BLAST service. (c) Choose a specific service, and modify the design to work with that service.**

it eventually becomes a segment of an executable workflow.

## myGrid discovery and execution architecture

The myGrid project has developed four key components (see Figure 2) to support the proposed experimental design life cycle. The scientist interacts with, personalizes, and chooses services, workflows, and data through a *workbench*. This workbench acts as a client to two components, used for discovering workflows and the services that can instantiate parts of a workflow. The *personalized view component* lets the scientist personalize the description of services by attaching *third-party metadata*.[2] It draws service descriptions from global or local service registries into a unified RDF-based framework, and it gives the user functions for attaching metadata to those service descriptions and querying the metadata. This lets the user attach his or her own recommendations in the form of local service metadata, which can be acted upon during workflow resolution. The *semantic find component* caters to conceptual metadata expressed in OWL. OWL descriptions of service classes are classified using description logic reasoning. This lets the registry index services against an evolving service classification and thus help resolve service classes to service instances.

Once discovered or built, a workflow is run by our *FreeFluo workflow execution engine*, which can handle WSDL-based Web service invocation. FreeFluo supports two XML workflow languages:

- One was inspired by IBM's Web Service Flow Language, which we used early on.
- Our own, XScufl, was developed as part of the Taverna project in collaboration with the Human Genome Mapping Project.

The FreeFluo engine (http://freefluo.sourceforge.net) and the Taverna workflow development environment (http://sourceforge.net/projects/taverna) are open source and downloadable. FreeFluo specifically supports workflow resolution, either asking the user to select between alternative services or automatically using the personalized view component. We envisage that the execution engine will eventually support automatic workflow harmonization. We also intend to support the use of the grid services specified by the Open Grid Services Architecture (see the "Grid Service Discovery" sidebar) alongside Web services.

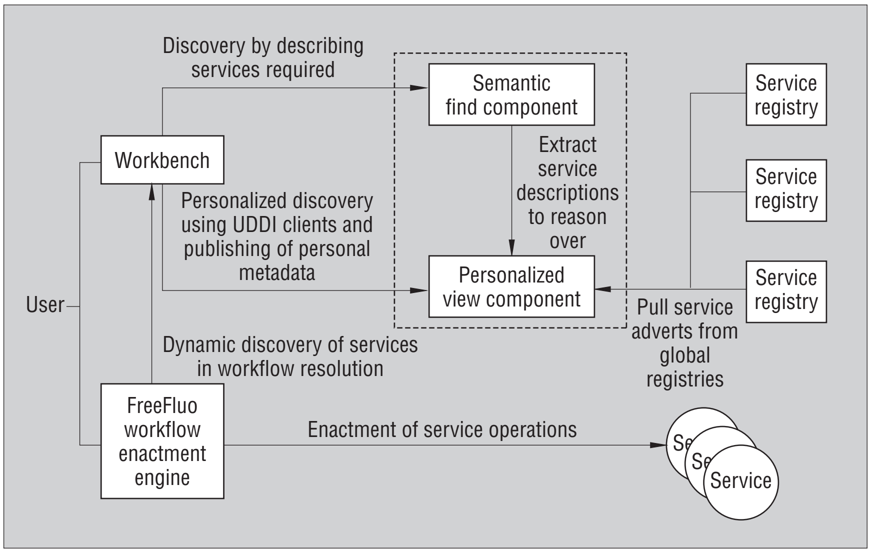

Discovery by describing services required
Semantic find component
Service registry
Workbench
Extract service descriptions to reason over
Service registry
Personalized discovery using UDDI clients and publishing of personal metadata
Personalized view component
Service registry
User
Pull service adverts from global registries
Dynamic discovery of services in workflow resolution
FreeFluo workflow enactment engine
Enactment of service operations
Service

**Figure 2. The interaction of components in the myGrid architecture during workflow creation, resolution, and execution.**

## Metadata to support resolution and harmonization

Although we first thought we could support automated resolution and harmonization by attaching a single, homogenous nugget of metadata to each service, we discovered that the scientist must create and use at least seven types of service metadata at specific points in the life cycle. Before discussing the metadata types, let's look at the lifecycle stages.

### Stage 1. Workflow creation

The availability or otherwise of specific services at workflow creation time shouldn't

## Grid Service Discovery

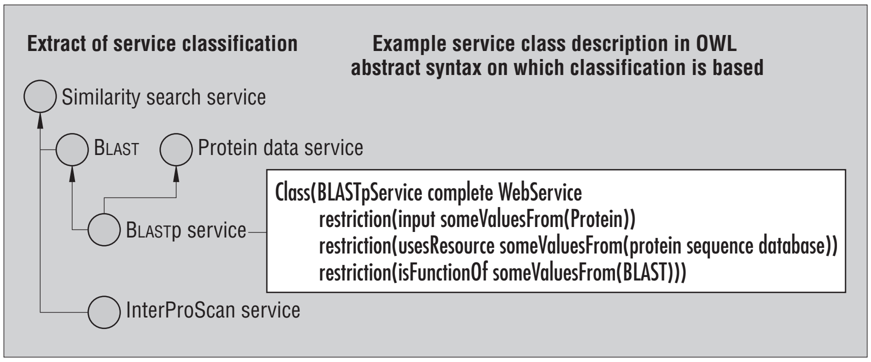
Service discovery is a critical element in large-scale, open distributed systems such as the Grid because it facilitates the dynamic identification of resources (abstracted as services).[1] Although standards have been developed for grid service registries, there has been little experience in using them for high-level tasks such as workflow resolution and harmonization. Condor is a workload management system that incorporates many Grid protocols and methodologies. ClassAds is the metadata language central to Condor's ability to match resource requests to resource offers. However, in general, ClassAds has been used for matching lower-level resource requirements such as memory and disk space with a job's requirements.[2]

Grid services have additional factors that must be considered during semantic discovery. First, services have lifetimes, so that a service found to be suitable at the time of instantiating a workflow template might not exist when the workflow instance is executed. Related to this is the concept of factories, which are used to create service instances with particular functions, possibly giving different service configurations on creation. Also, grid services include the idea of structural service data elements—that is, metadata that is attached to and can be queried from services at the location at which they are deployed. This provides a mechanism for providing metadata in addition to publishing their descriptions in registries.

### References

1. I. Foster et al., "Grid Services for Distributed System Integration," *Computer*, vol. 35, no. 6, 2002, pp. 37–46.

2. R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," *Proc. 7th IEEE Int'l Symp. High Performance Distributed Computing* (HPDC 98), IEEE CS Press, 1998, pp. 140–146.


Extract of service classification
Example service class description in OWL abstract syntax on which classification is based
Similarity search service
BLAST
Protein data service
BLASTp service
InterProScan service

```
Class(BLASTpService complete WebService
    restriction(input someValuesFrom(Protein))
    restriction(usesResource someValuesFrom(protein sequence database))
    restriction(isFunctionOf someValuesFrom(BLAST)))
```

**Figure 3. Using a classification to discover classes of service to fulfill a task.**

dictate what scientists use at runtime: they must be able at this point to choose between service classes. In the scenario described earlier, the biologist starts with a protein sequence and searches for similar proteins. In addition to BLAST, several other services including InterProScan and FASTA could be used to perform these searches. By organizing these service classes along several axes, such as the input data they accept and the function they provide, the biologist can also discover functionally similar service classes by simply browsing multiple hierarchical views. However, providing such a multiaxial classification by hand is difficult and error prone.[3] So, our system automatically calculates the myGrid service classification by using description logic reasoning over conceptual descriptions of service functionality written in OWL.[4] (The initial implementation is in DAML+OIL, OWL's predecessor.) Figure 3 shows an extract of such a service classification and an example of an OWL class description used to calculate the hierarchy.

We found that these classes must be very specific to ensure the workflow resolution process doesn't alter the experiment's bioinformatics design against the scientist's wishes—for example, inserting an InterProScan service for a BLAST service. OWL allows the use of anonymous class descriptions in place of preenumerated service class names. We can therefore provide a compositional mechanism for forming service class descriptions. This lets service publishers describe exactly the class to which their service belongs and lets users specify requirements without needing a prohibitively large preenumerated classification. An exact, unambiguous description of functionality is essential if these descriptions are to successfully drive automated resolution.

A high-level standard for this conceptual description of a service is being developed by the DAML services coalition[5] and is now available as an OWL ontology (OWL-S version 0.9; www.daml.org/services/daml-s/0.9).

## Stage 2. Workflow resolution

For common tasks, we assume each class (however specific) has several available member services at any one time. So, when selecting a service, the scientist should consider additional criteria that are particular to a workflow's user (as opposed to its creator). Let's look first at those criteria and then at the process of using them to select services.

***Metadata.*** From the bioinformatics point of view, the services can be considered to be functionally equivalent. So, the scientist might need to consider additional *operational* metadata such as performance, security, provenance recording, or cost of a service. For example, one BLAST service might provide much more provenance metadata than another on the versions of databases searched, while performance monitoring might show that the other returns results much faster. The service's publisher can't provide all this information at registration. Both the Web Services and Grid communities are developing architectures and metadata standards for constantly monitoring performance.[6] Performance metadata is seen as dynamic and provided by third parties, whereas the service provider might publish static forms of operational metadata in public registries or personalized views.[2]

How a service is presented can affect the degree to which it can be configured. For example, the Soaplab BLAST service lets users configure the number of search hits returned, while the simple version of the DDBJ service doesn't provide this. If the user has specific requirements on the way a service must be configured (for example, the number of hits returned), she must add this to the abstract service description so that it can be considered when the system chooses which BLAST service to execute. Each service also requires a description of possible configurations. Choosing between functionally similar services should also be informed by others' experience, so we must provide access to information detailing how others have used these services in their workflows. Therefore, we need provenance information about services. In myGrid, execution of a workflow gives rise to a provenance record detailing what data was used and with which services.[7] Users can query this information to find out the context in which service classes have been used.

***Resolution before or during workflow execution.*** A service class constrained by the criteria just described should provide enough information to select an individual service at any given time. The scientist can perform resolution either before executing a workflow or once the service is actually required in the running workflow. Providing resolution during execution lets the user select services most suited to current conditions. This is essential on the Grid, where services constantly appear and disappear during the lifetime of long-running workflows.

***Resolution by hand or by machine.*** To accommodate this in myGrid, the workflow execution engine, FreeFluo, can perform on-the-fly resolution either automatically or by prompting for user intervention. Manual resolution can occur by browsing over unstructured metadata in the personalized view component. In the short term, most services don't have all the necessary types of structured metadata to support automated service selection. So, we must at least support a manual version of resolution that relies on human users interpreting unstructured metadata. However, supporting just-in-time resolution becomes difficult if that resolution depends on repeated intervention by the user. In myGrid, we're driving the provision of more structured metadata at each level and developing a two-step automated resolution process that will depend on that metadata.

The first step finds all services that are members of the service class specified in the workflow. This uses the semantic find component to look up all services that are indexed by the relevant class. If the class is specified as an OWL class definition, the system's find component uses description logic reasoning to relate this description to the current service classification and so link into the index of available services. Again, to preserve the experiment's overall scientific design, automated substitution occurs only between highly similar services.

The system's workflow execution component invokes the second step if more than one service is available to perform the task. Additional operational constraints specified by the workflow author help the system select a service that, for example, performs better, costs less, or provides more provenance information. The system queries for operational metadata within the registry, which is then compared and the appropriate selection made.

## Stage 3. Workflow harmonization

Although the chosen service might perform the task adequately, we have no information about how to actually run it. We need a strategy to take the selected service and ensure that we can run it within the workflow context. For this, we need two mechanisms to accommodate the low-level differences between functionally equivalent services:

- Mapping the format of parameters passed in and out of the service
- Mapping the invocation method in terms of calls to low-level service operations

***Format harmonization.*** In our example, the specific Blast program Blastp requires a protein sequence as input. Bioinformatics has many formats for protein sequences, or records that include protein sequences. In workflow resolution, we might have selected between different services that required the same information but in different data formats. We aim to accommodate this by interposing harmonization services to transform data upstream of the main user-specified service (such as Blast). We find these harmonization services automatically through a registry query for the service class that supports format transformation with the relevant inputs and outputs. We then insert them into the workflow. To preserve the experiment's overall design, these inserted services must be experimentally neutral. Automatic insertion of an experimentally significant service such as InterProScan would alter the high-level design and, from the scientist's point of view, might invalidate the results.

***Invocation and interface harmonization.*** Invocation of a service often has several stages, and multiple low-level operation calls are often needed to perform a given high-level workflow step. Each service might provide calls for each one of these stages or might combine several stages into a single call. This is certainly the DDBJ's implementation of a Blast Web Service. Table 1 shows the steps involved and how each service combines invocation steps differently and conforms to a different interface. Invocation and interface metadata are deeply intertwined, with specific operation calls in the interface reflecting each invocation stage. Much of the difference is due to Soaplab's implementation of a stateful service. (*Stateful* services maintain state information between operation calls. They are used when a multistep conversation is required but a constant connection can't be maintained.) As such, it has much more in common with emerging grid services as opposed to the simple interfaces advocated in the Web Services community. With Blast searches taking a significant period of time (usually several minutes to hours), most Blast providers operate a job-based service with queuing better suited to stateful interaction.

**Table 1. How Soaplab and DDBJ Blast services support different stages of invocation.**

| Stage of invocation | DDBJ Blast service | Soaplab Blast service |
|---|---|---|
| Creating a job | n/a | **createEmptyJob()** |
| Configuring the service | simpleSearch (program, database, query) | **set_database(database,job)** |
| Setting input data | | **set_query_sequence(query,job)** |
| Running the job | | **run(job)** |
| Getting output data | | **getSomeResults(job)** |

WSDL specifies the interface information in the table; of all the categories of metadata described, it's the one service developers provide most readily. However, to explicitly translate from an abstract description of a task such as "align protein sequences" to the specific sequence of low-level service calls, we need additional information to map between the two levels. To address this issue, myGrid has tried several approaches:

- Abolish diversity between services. Soaplab provides access to 150 services, all accessible via a common interface.
- Require each service to register an executable workflow fragment, which can be automatically swapped in for the abstract task if that service is chosen for the task.
- Provide constructs in the workflow language that simplify execution of services that require a series of calls. For example, Taverna developers have included plug-ins that allow the execution of a Soaplab service to appear to the user as one step in the workflow but which is in fact automatically expanded into the five calls shown in Table 1.

To complicate the issue, we've found there's often no simple one-to-one link between the higher-level domain description of service functionality and lower-level descriptions of invocation. For example, the DDBJ Blast service is more generic than the Soaplab Blastp service and can support two additional tasks using the underlying Blast programs: Blastn and Blastx. To specify the task (in this case which program to use), the workflow system must pass additional parameters to the service. The simplest way to denote this many-to-one mapping is by registering the DDBJ Blast service as belonging to three separate functional service classes and by providing separate invocation model metadata to correspond to each of these functionalities. Describing these polymorphic services is a difficult issue for which an ideal solution has yet to be found.

***Harmonization before or during workflow execution.*** Workflow harmonization must occur after resolution. So, if resolution is to occur at execution time, so must harmonization. This places challenges on the execution engine, which must respond to changes not only in services called but also in the structure of the workflow itself. FreeFluo provides the ability to nest component workflows within a top-level workflow. This allows the decoupling of fine-grained changes to individual service interaction from the workflow's high-level structure.

***Harmonization by hand or machine.*** Automating this process would greatly benefit scientists. Users shouldn't need the expertise required to rewrite the workflow at this fine-grained service interface level, because changes at this level should have no bearing on the scientific outcome. If these changes

Table 2. Metadata's uses in the experimental life cycle.

| | | Stages in service publication and use within a workflow | | | | |
|---|---|---|---|---|---|---|
| | | **Advertise a service (human)** | **Index a service (machine)** | **Specify a service class during workflow construction (human)** | **Harmonize service resolution and workflow (human)** | **Harmonize service resolution and workflow (machine)** |
| Seven types of service metadata | Conceptual | Service characteristics described by provider using conceptual description and advertised in a service registry; third parties add personal recommendation | Description logic reasoning used to index service within service classification | Desired characteristics described by scientist using DAML-S-style description of functionality together with operational constraints such as acceptable performance | Registry browsing search to select appropriate service | Description logic reasoning to determine current members of service class |
| | Configuration | | | | | Registry query to select between members based on configuration and provenance |
| | Provenance | | | | | |
| | Operational (performance/cost) | | | | | Registry query or negotiation based on cost performance |
| | Invocation model | Service characteristics described by provider using WSDL description and mapped to high-level conceptual description | | Ignoring low-level characteristics of individual services at this point | Manual workflow harmonization to accommodate specific invocation sequences, formats, and parameter names | On-the-fly workflow harmonization by the workflow executor |
| | Interface | | | | | |
| | Data format | | | | | |

are done manually, they can take significant time and effort. However, the difficulties of providing sufficient structured metadata, and the complex mapping between abstract task and concrete invocation, described earlier, make automated harmonization a challenge.

## Seven kinds of metadata

We've identified seven kinds of metadata that aid users and the myGrid system to create, resolve, and harmonize workflows:

- A *concept*-based description of the service would be written by the provider at publication time or possibly later by a third party and stored in a registry. For example, "BLAST would be described as a sequence similarity search service."
- *Configuration metadata* to support a particular task would be written by the provider at publication time and stored in the registry. For example, "blast@somedomainname.org can be configured to return version information about databases used in the search and the version of the software."
- A *provenance description* would state how others have used this service in the past and any recommendation or opinion they might have (opinions would be handwritten by third parties but usage information could be aggregated automatically from workflow provenance records). For example, "A colleague has stated that blast@somedomainname.org is reliable and that he has used this type of service in several published workflows."
- An *operational description* of the service in terms of cost, access rights, and quality of service would be written by both the provider and third parties and stored in a registry. For example, "blast@somedomainname.org provides public access at no cost, although a benchmark query takes on average 10 minutes."
- The *invocation model* of a service would be written by the provider at publication time and stored in the registry. For example, "blast@somedomainname.org provides a stateful invocation model which requires multiple service calls."
- The *interface* to a service would be written by the provider at publication time and stored in the registry. For example, "blast@somedomainname.org provides a WSDL interface document."
- The *format* in which the service expects input data and produces output data would be written by the provider at publication time and stored in the registry. For example, "blast@somedomainname.org returns a search report as an XML document conforming to a specified schema."

Table 2 summarizes how each kind of metadata is used during the experimental life cycle and by whom—a person or an automated process. The user specifies the concept, operational, configuration, and provenance metadata when the workflow is created and used in resolution. Either the user or an automated process uses the data formats, invocation model, and interface metadata to harmonize the workflow.

In the dynamic Grid environment, we must shield the scientist from the complexities of substituting one functionally equivalent service with another by either enforcing common interfaces or, more realistically, developing automated processes to cope with low-level diversity. Explicitly identifying the seven types of metadata (seven ways in which one service can differ from another) in myGrid is helping us understand the diversity of services and systematically develop mechanisms to cater to it. The mechanisms for creating, resolving, harmonizing, and executing workflows are integrated into the myGrid architecture. The architecture supports the whole experimental life cycle with components for storing the different types of metadata, such as personal preferences and semantic descriptions, and for reasoning over this data to find appropriate services in workflow resolution.

However, the key challenge still remains of how to provide and maintain sufficiently structured service metadata. Within myGrid, we're beginning to assess whether the development of service metadata creation and collection applications can make the maintenance of this metadata a realistic task. We're developing applications that let service publishers or third parties create functional descriptions of services that can then be registered. We're also investigating mechanisms to automatically collect performance and reliability metadata. Only when we can provide such accurate and up-to-date metadata will we be able to let the system make appropriate service selections and integrate them in the workflow. ■

## Acknowledgments

This work is supported by the UK e-Science program EPSRC GR/R67743 and Stanford University's DARPA DAML subcontract PY-1149. We also acknowledge the myGrid team and our industrial partners: IBM, Sun Microsystems, GlaxoSmithKline, AstraZeneca, Merck KgaA, geneticXchange, Epistemics, and Network Inference.

## References

1. R. Stevens et al., "Performing In Silico Experiments on the Grid: A Users Perspective," *Proc. UK OST e-Science 2nd All Hands Meeting*, UK Office of Science and Technology, 2003, pp. 43–50.
2. S. Miles et al., "Personalized Grid Service Discovery," *IEE Proc.*, vol. 150, no. 4, Aug. 2003, pp. 252–256.
3. A.L. Rector, "Clinical Terminology: Why Is It So Hard?" *Methods of Information in Medicine*, vol. 38, nos. 4–5, Dec. 1999, pp. 239–252.
4. C. Wroe et al., "A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data," *Int'l J. Cooperative Information Systems,* vol. 12, no. 2, Mar. 2003, pp. 197–224.
5. A. Ankolenkar et al., "DAML-S: Web Service Description for the Semantic Web," *The Semantic Web: ISWC 2002, Proc. 1st Int'l Semantic Web Conf.*, LNCS 2342, Springer-Verlag, 2002, pp. 348–363.
6. *A Grid Monitoring Architecture*, Global Grid Forum Performance Working Group, 2002, www-didc.lbl.gov/GGF-PERF/GMA-WG.
7. M. Szomszor and L. Moreau, "Recording and Reasoning over Data Provenance in Web and Grid Services," *Int'l Conf. Ontologies, Databases and Applications of Semantics* (ODBASE 03), LNCS 2888, Springer-Verlag, 2003, pp. 603–620.

## The Authors

**Chris Wroe** is a clinical research fellow in the Information Management Group of the University of Manchester's Department of Computer Science. His research interests include the use of metadata and ontologies to help represent biological information and the services that operate on them. He originally trained and practiced as a medical doctor before moving into medical informatics and more recently into bioinformatics. Contact him at the Dept. of Computer Science, Univ. of Manchester, Oxford Rd., Manchester, M13 9PL, UK; cwroe@cs.man.ac.uk; www.cs.man.ac.uk/~wroec.

**Carole Goble** is a professor in the Department of Computer Science at the University of Manchester and the director of the myGrid project. Her research interests center on the accessibility of information, primarily through the use of ontologies for representing and classifying metadata, and include many application areas, particularly life sciences. She cochairs the Semantic Grid Research Group in the Global Grid Forum standards organization. Contact her at the Dept. of Computer Science, Univ. of Manchester, Oxford Rd., Manchester, M13 9PL, UK; carole@cs.man.ac.uk.

**Mark Greenwood** is a research fellow in the Informatics Process Group of the University of Manchester's Department of Computer Science. His research interests include process modeling, software engineering, and the integration of business processes and the software systems to support them. He received his PhD in computer science from the University of Southampton. Contact him at the Dept. of Computer Science, Univ. of Manchester, Oxford Rd., Manchester, M13 9PL, UK; markg@cs.man.ac.uk; www.cs.man.ac.uk/ipg/People/mark.html.

**Phillip Lord** is a research associate in the Information Management Group of the University of Manchester's Department of Computer Science. His research focuses on the use of computer technology in biology and medicine. He works on the myGrid project, investigating using ontologies to enable service discovery. He received his PhD in genetics and molecular biology from the University of Edinburgh. Contact him at the Dept. of Computer Science, Univ. of Manchester, Oxford Rd., Manchester, M13 9PL, UK; p.lord@russet.org.uk; www.russet.org.uk/home.html.

**Simon Miles** is a researcher at the University of Southampton's School of Electronics and Computer Science. His research interests include service and workflow discovery, deployment of distributed grid applications, and the development of middleware supporting provenance information recording. He received his PhD in computer science in the area of agent-oriented software engineering from the University of Warwick. Contact him at the School of ECS, Univ. of Southampton, Highfield, Southampton, SO17 1BJ, UK; sm@ecs.soton.ac.uk; www.ecs.soton.ac.uk/~sm.

**Juri Papay** is a senior research fellow at the University of Southampton's School of Electronics and Computer Science. His research interests are distributed computing, performance prediction, stochastic simulations, and software design. He received his PhD in computer science from the University of Warwick. Contact him at the School of ECS, Univ. of Southampton, Highfield, Southampton, SO17 1BJ, UK; jp@ecs.soton.ac.uk; www.ecs.soton.ac.uk/~jp.

**Terry Payne** is a lecturer in the University of Southampton's School of Electronics and Computer Science. He received his PhD in AI from the University of Aberdeen. He is a coauthor of the DAML-S/OWL-S Service Description Language and a member of the Semantic Web Services Language Committee. Contact him at the School of ECS, Univ. of Southampton, Highfield, Southampton, SO17 1BJ; trp@ecs.soton.ac.uk; www.ecs.soton.ac.uk/~trp/index.html.

**Luc Moreau** is a reader in the University of Southampton's School of Electronics and Computer Science. His research interests include grid computing, focusing on semantic service discovery, notifications, and provenance; distributed systems; distributed garbage collection; and mobile agents. He received his PhD in computer science from the University of Liège. Contact him at the School of ECS, Univ. of Southampton, Highfield, Southampton, SO17 1BJ, UK; l.moreau@ecs.soton.ac.uk; www.ecs.soton.ac.uk/~lavm.