A Semantic Service Environment: A Case Study in Bioinformatics

Stephen Potter and Stuart Aitken

School of Informatics, The University of Edinburgh, Appleton Tower,
Crichton Street, Edinburgh EH8 9LE, UK
{stephenp, stuart}@inf.ed.ac.uk

Abstract. In recent years, web services have become increasingly important components of the scientific methodology of certain domains. Currently, however, the description and use of most these is purely 'syntactic'; that is, the semantics of the services are left to the human user to infer or acquire by other means before deciding whether and how to use a service. Consequently, there are opportunities to bridge this semantic gap through the application of emerging semantic web and semantic web service technologies in these domains, thereby enriching and expanding a user's service interactions. This paper presents its authors' experiences of the application and use of these emerging technologies in a displicine in which web services already play a key role: bioinformatics.

1 Introduction

As a discipline, bioinformatics is notable for its diversity of aims and methods, and the heterogeneous nature of the computing resources applied to achieve them [13]. Bioinformaticians are accustomed to creating analysis pipelines [2] [16] [19] for, say, the assembly of a complete gene sequence from a set of subsequences derived from a lab experiment or the alignment of potentially homologous gene sequences. The dynamic nature of the databases accessed in these steps — the rate of data production is such that databases may be updated daily — creates a demand for automation in the analysis process, with the explicit representation and storage of the workflow, its invocation, and potentially, its exchange and reuse. Due to uncertainty of service availability, dynamism in the selection of an active service must be taken into account. Finally, the bioinformatician may wish to inspect the results as the workflow progresses to obtain feedback and determine whether the processing parameters are correct.

Web services is one of the paradigms that has been adopted within bioinformatics for exposing computational resources; these offer the advantages of providing an open architecture using relatively standardised transport and communications layers. However, these transactions occur at a syntactic level; there is, as yet, little semantic description of the available services. Consequently, bioinformatics presents an opportunity to apply emerging semantic web services technologies and standards to an existing set of services, and, in so doing, to learn more about the engineering aspects of such an enterprise. This paper presents our approach and experiences of an application of this sort; this should be of interest those who are planning similar applications, or who are involved in the design of these technologies and standards.

Section 2 of this paper provides a discussion of the nature of current bioinformatics web services, and presents one of a number of scenarios we have used to guide our approach. Section 3 itemises some of the requirements we identify for a practical semantic layer in this domain. This is followed by a description of how we have gone about the task of introducing these semantics. Finally, following a brief discussion of related work (section 5), some of the implications of this work are discussed and some conclusions drawn in section 6.

2 Bioinformatics Web Services

Before proceeding, it is first necessary to say a little about the nature of extant web services in bioinformatics. These resources are often provided by large bioinformatics 'data centres' such as the European Bioinformatics Institute (EBI)[1], the Virginia Bioinformatics Institute (VBI)[2], and the DNA Data Bank of Japan (DDBJ).[3] Many commonly used bioinformatics computational components are available over the web, through either (or both) 'manual' web browser interfaces or conventional 'programmatic' web services interfaces. 'Manual bioinformaticians' will create an analysis pipeline by cutting and pasting data from one browser interface to another, often accompanied by an arbitrary amount of editing and reformatting of the data. Here, though, we are primarily concerned with the programmatic interfaces; for our purposes we assume that, in general, the offered web services share the following characteristics:

- a web service can be considered an 'information transformer', converting one particular input into an output; values of other parameters may qualify this behaviour.
- a service will appear as a 'black box' to its user; that is, the transformation it effects appears as a single atomic process.
- a web service will be 'idempotent'; that is, no notion of state persists from one call to the service to the next (exceptions to these last two points include certain EBI services which have a notion of state that allows clients to query the current status of a running service).
- a service will be independent, in that its operation does not depend on the existence or availability of other services.
- the interface to a service will be through a SOAP [9] 'remote procedure call' over HTTP; this interface will be described using a WSDL [4] document.
- the WSDL document will not introduce any complex (XML Schema) typing of inputs or outputs; instead, it will rely on the use of 'simple' types such as "string".

This last point raises the question of data formats for bioinformaticians. As has been noted (by, for example, Stein [18]) there are no agreed formats within bioinformatics for the formats used by services; typically they will return results represented in some *ad hoc* format, containing, say, a mixture of search results, hand-crafted natural language annotations and references to third-party publications and other data sources. As

[1] `http://www.ebi.ac.uk/xembl/XEMBL.wsdl`

[2] `http://staff.vbi.vt.edu/pathport/services`

[3] `http://xml.nig.ac.jp`

a result, much time is expended by the users of on-line services in writing code for 'screen-scraping' (for extracting data from standard web browser pages) or for converting data from the format produced by one service into the format expected as the input for the next. (Indeed, the realisation of this latter fact has led to the establishment of the Open Bioinformatics Foundation[4], whose goal of supporting bioinformatics programming has involved the creation of code libraries for a number of different programming languages for this sort of data manipulation.) This characteristic of services makes workflows in the domain particularly brittle: if a provider alters a result format, these translators must be re-engineered.

There are few universal standards for data represention. However, there are efforts to create standard representations in specific areas, for example, the MIAME (Minimum Information About a Microarray Experiment) initiative [3] aims to standardise both the recording and the reporting of microarray-based gene expression data. In the ontology arena, the Gene Ontology consortium has established a flat-file format for simple ontologies, and is moving to a more flexible format (the OBO format[5]). Currently, however, each web service (provider) will generally use its own manner of representing data, and, in the absence of standards, an alternative approach to automating service workflows is to make conversion programs — termed *shims*, following [11] — available as web services like any other. (In general, a shim service might be thought to perform a purely syntactical manipulation of its input. However, its effects might be more subtle; for instance, a shim service which selects a subset of some data might better be thought of as performing some semantic winnowing of this data.)

2.1 Scenario

To motivate this work, we have considered a number of real scenarios in which bioinformaticians achieve their aims by interacting with existing web services. These scenarios include: the assembly of a complete gene sequence from a set of subsequences; the alignment of homologous gene sequences; and the search for tissue homologies by identifying homologous genes. (These are comparable to the 'use cases' that are being developed under the BioMOBY initiative.[6])

By way of a concrete example of the use of bioinformatics web services, we here present a *sequence alignment scenario*. This scenario is illustrated in Figure 1 where part a) represents the three major steps in the workflow from the user's perspective, and part b) shows the mapping into web services. The steps are:

1. given the ID of the gene find the protein sequence that corresponds to the gene;
2. find those sequences that are potentially homologous with this sequence, i.e. find those with a high degree of similarity, and;
3. place those sequences into a relative alignment. The degree of alignment reflects the closeness of molecular function.

This corresponds to the following sequence of web service invocations:

[4] `http://www.open-bio.org`
[5] `http://www.geneontology.org/GO.format.html`
[6] `http://www.biomoby.org`

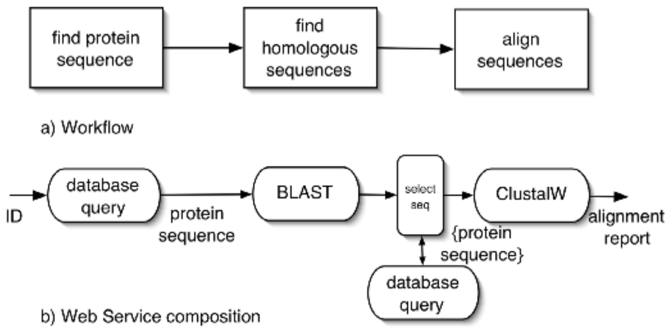


Fig. 1. Workflow and web services for the sequence alignment scenario

1. a protein sequence is obtained through a database lookup service using the supplied gene identifier;
2. a BLAST (Basic Local Alignment Search Tool) search for similar sequences in the sequence database is performed. (Since this is a search of a database of protein sequences, specifically it is the "blastp" variant program that is required here.) This is a pairwise comparison.
3. The most similar sequences are then input to the multiple sequence alignment program *ClustalW*. Multiple sequence alignment considers the entire set of matching sequences, and identifies regions of common structure.

A significant transformation of the output of the BLAST search was required to create the input to ClustalW. There are alternative ways of performing this transformation; here it was done by writing code to extract the identifiers of matching protein sequences from the BLAST output report, and then querying the database using these identifiers to retrieve the actual sequences.

In general, bioinformatics web services can be invoked by any client able to parse their WSDL descriptions, and handle SOAP messages over HTTP. Hence, using convenient libraries, sequences of service interactions and data transformations such as that outlined above can be embodied in a conventional computer program. Currently, however, this can be done only if the locations and data formats of the services are known to the programmer. Alternatively, a higher level interface and more flexibility is offered by a number of tools (such as the Taverna Workbench [15], which is aimed specifically (but not exclusively) at bioinformaticians) that allow their users to construct workflows and then, by handling the interactions with services, invoke them.

3 Desiderata for a Semantic Service Environment

By considering scenarios such as that above, we can identify the principal steps that the bioinformatician undertakes when constructing a workflow of services:

1. the identification and expression of the goal of the workflow. This will be determined by the immediate and long-term goals of the scientist's research. Ideally, this externalisation of scientific goals would be made directly at a semantic level;

2. the identification of the major processing tasks, that is, developing (at the semantic level) a practicable sequence of tasks for arriving at the goal given the data currently available. This will necessarily involve some awareness of the types of service (and data) that are available so as to construct a description of this desired sequence;
3. the identification of actual (and currently active) services able to enact this sequence, including any necessary shim services. This will involve discovery of services, with perhaps a selection from among competing matching services. The discovery will typically involve searching for instances of a specified type of service and/or which produce some specified output;
4. the invocation of the workflow. At this point, the semantic description must be bound to the underlying 'syntactic' computational description of the services involved (this need not be exposed to the scientist);
5. the storage of the workflow for later reuse. For this purpose, some language rich enough to capture the semantics of the developed process would be required.

Steps 1–3 are unlikely to be wholly independent and to conform strictly to this ordering; identification of the major tasks in step 2, for instance, is likely to be influenced by and determined to some extent by knowledge of the services that are available. However, if our aim is to faciliate and enrich workflow construction of this sort, the above steps allow us to outline the semantic properties we would like in this domain, namely service *description*, *discovery*, *selection* and *invocation*, along with the *capture of workflow process*.

Currently, however, the WSDL descriptions of services are primarily syntactic in nature, inasmuch as they describe the types of the service inputs and outputs but not the semantics of these parameters or of the task that the service performs. This means that prospective clients usually need to gain an understanding of the behaviour of and interface to the service from some other source (for example, by emailing its authors or reading their web pages) in order to use the service. Furthermore, it restricts the possibilities for dynamic discovery of appropriate services to meet a client's needs (Taverna, for example, provides its users with lists of 'known' web services from which to choose; this list can be augmented if the user knows of the URIs of other services, but no automatic discovery is attempted).

Given these requirements, and the desire to apply emerging semantic web technologies rather than invent new ones for this domain, in the following sections we outline our approach to introducing semantics to bioinformatics web services.

4 A Semantic Bioinformatics Service Environment

In order to introduce semantics into the current practice of service-based bioinformatics and satisfy the above desiderata, we have introduced the following:

– The semantic description of services using OWL-S, plus a dedicated domain ontology described in OWL (described below in sections 4.1– 4.3);
– An automated discovery service using a description logic reasoner (section 4.4);
– A semantic workflow tool, which acts as a user interface to the discovery service, and allows the invocation of services (section 4.5).

4.1 Introducing Semantics: OWL and OWL-S

In an attempt to move towards a more ‘semantic’ environment, we have chosen to introduce OWL-S descriptions of the existing services. OWL-S [5] is a generic upper ontology for specifying web services; it is intended to allow providers to describe (using an XML document) their services in such a manner as to allow their discovery, selection, composition and invocation; and, where appropriate, allow for monitoring, mediation and failure recovery. OWL-S is specified in OWL, the Web Ontology Language [7], which provides a language (built on the RDF data model) for specifying Description Logic (DL) constructs in the syntax of XML. DLs form a subset of first-order logics that are particularly suited to the description of hierarchical ontologies of entities, and possess appealing tractability characteristics.

The OWL-S ontology is divided into three principal areas: the *Profile*, *Model* and *Grounding*. The Profile is used to describe the purpose of the service, and so primarily has a role in the initial discovery of candidate services for a particular task. The Model describes how the service is performed, and is intended to allow simulation of and mediation with the service, to enable the execution of the service to be monitored, etc. Finally, the Grounding specifies in concrete terms how the service is actually invoked.

The role of the Profile, then, is to describe the essential capability of the service by characterizing it in functional terms (in addition, non-functional aspects of the service can be specified through ‘service parameters’). This functional characterisation is expressed by specifying the class of the service and by detailing the inputs it expects, the outputs it produces, the preconditions that are placed on the service and the effects that the service has. The description of preconditions and effects presents something of a problem for OWL-S; their expression requires, in effect, variables and rule-like constructs, which are outside the expressive capabilities of DLs (and hence, outside the capabilities of DL reasoners). In this domain, however, as noted in section 2 above, we assume that web services are essentially stateless; accordingly we do not need to model preconditions and effects, but this will not be the case for all domains. As well as characterising services, the Profile has an additional use: to allow potential clients to specify their desired services (the descriptions of which may be partial or more general in nature where certain specifics are irrelevant to the client).

Since the OWL-S ontology is essentially domain independent, in order to provide this sort of ‘semantic typing’ of both services and queries, we need to extend the ontology with ontological concepts from, in this case, the domain of bioinformatics.

4.2 A Bioinformatics Ontology Extension

The approach taken is to extend the basic generic description of an OWL-S Profile, hierarchically subclassing it with the various types of bioinformatics service that can be identified (figure 2). In addition, we also generated a hierarchy of the various conceptual data types that describe the inputs and outputs.

Being essentially simple taxonomies of services and data, these ontology extensions are not as rich as one might expect, due in part to the practicalities of expressing these concepts in DLs. For instance, when coming to define the class *ProteinSearchService*,

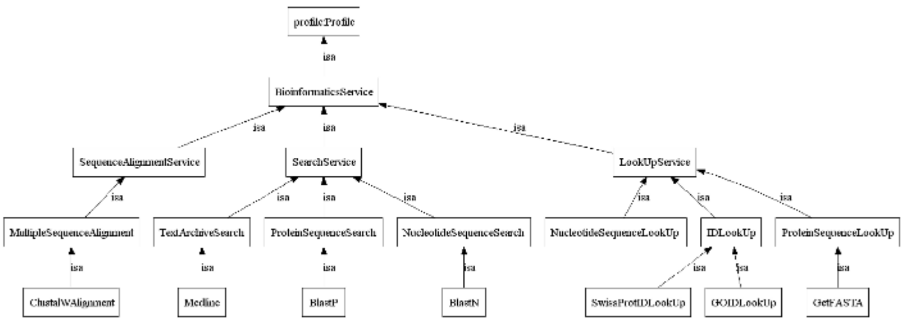


Fig. 2. A fragment of the bioinformatics ontology showing the hierarchical arrangement of some types of service. Note that the root of this subtree is the (OWL-S) Profile concept

one might reasonably attempt to express as restrictions on this class the features of its inputs and outputs that are common to all service instances of the class:

$$\mathit{ProteinSearchService} \equiv \mathit{SearchService} \sqcap \exists\, \mathit{hasInput.ProteinSequence} \sqcap \exists\, \mathit{hasOutput.Report}$$

In other words, that instances of *ProteinSearchService* have some input of class *ProteinSequence* and give some output of class *Report*. Now, one might want use this definition to define the more specialised service concept *BlastP*:

$$\mathit{BlastP} \equiv \mathit{ProteinSearchService} \sqcap \exists\, \mathit{hasInput.DatabaseName} \sqcap \exists\, \mathit{hasInput.ProgramName} \sqcap \exists\, \mathit{hasOutput.BlastSearchReport}$$

Now, unless we introduce appropriate disjointedness axioms and cardinality constraints, from this definition it is impossible to infer how many inputs and how many outputs a *BlastP* service should have. Introduction of these appropriate axioms and constraints can have implications for the tractability of automated reasoning (e.g., the assertion of cardinalities other than 0 or 1 takes us from the OWL-Lite subset of the language to OWL-DL). A more practical concern, though, is the engineering implications of these sort of definitions; for a bioinformatician wishing to describe his service (who cannot be assumed to be familiar with DLs), complex definitions of this sort are unwieldly and difficult to use, and can result in inappropriate conceptualisations having unintended implications. The client who wishes to state her requirements will face similar difficulties. To a certain extent, these problems are due to the unsuitability of DLs, with their lack of arbitrary variables and rule expression, for representing processes.[7] As a consequence, the ontology extensions we have created are relatively sparse, being little more than definitions of taxonomies of concepts. The intention is that the service provider and client are able to use these 'naively' to express the service and data types they provide or require.

[7] Note that the most recent version of OWL-S (version 1.1) includes the concept of a variable, and suggests ways by which to introduce rule-like expressions; however, it is not entirely clear how these should be used or reasoned with.

There are now several tools available which enable the user to create and extend OWL ontologies; here we use the Protégé ontology editor.[8] The extensions were created by an informatics researcher who also has experience of bioinformatics, and since we restricted ourselves to modelling only certain — but hopefully representative — scenarios, these extensions represent partial views of the domain for this purpose. The creation of the 'right' ontologies is an issue of obvious importance, not only here but throughout the semantic web community and beyond. A good ontology in this case would both allow service providers an appropriate degree of expression to capture accurately and completely the behaviour of their services and enable clients to express their needs in as specific or as general a manner as is appropriate. In this sense, assessing the value of an ontology is a pragmatic question. Moreover, the use of domain-specific ontologies in this manner places certain practical obligations on agents in this domain: for service discovery to be possible, there must be a certain degree of consensus in the content and use of ontologies by both the service providers and potential clients.

4.3 Describing Bioinformatics Services

Now we can use the ontology extensions outlined above to describe the Profiles of bioinformatics web services in the manner suggested. Each particular service is an instance of the appropriate subclass of *BioinformaticsService*, and each of its inputs and outputs are typed[9] with the appropriate data class expressions. We do not, however, make use of OWL-S service parameters to try to capture the non-functional qualities of these services; factors such as trust, efficiency and availability of services will undoubtedly play a major role in service selection, but remains an area of future research.

Now we need to consider how to express the other constituents of an OWL-S description, namely the model and the grounding. In each case, this is relatively straightforward. Since, as discussed in section 2, these services are generally modelled as 'black box' atomic processes, this naturally leads us to describe their models using the OWL-S concept of *AtomicProcess*. However, note that, while appropriate for the existing web services, this choice means that the services have rather inexpressive models, and as a result the possibilities for simulation, mediation, monitoring, etc. of services is limited. As also stated in section 2, the interfaces to these services are usually described using WSDL documents; hence this becomes the obvious choice for their OWL-S grounding.

The construction of an OWL-S document describing a particular service is a semi-automatic process. From its WSDL description, the OWL-S API [6] allows the automatic construction of a basic OWL-S outline document, having a grounding that refers to its WSDL, an atomic process model, and a profile which has the appropriate number of inputs and outputs. Using the bioinformatics ontology extension, these inputs and outputs, along with the class of profile itself must then be manually annotated with the appropriate semantic terms from the extended ontology.

[8] `http://protege.stanford.edu/`

[9] Through the use of the OWL-S *parameterType* relationship.

Although in this case we have provided the OWL-S descriptions of others' web services,[10] ideally it would be the service providers who would generate these. This would require appropriate tools to be available and, since the task is always likely to have a manual component, the ontological descriptions to be 'usable' (a subject touched upon in the previous section).

4.4 Semantic Discovery

Among the fundamental capabilities of DL reasoning engines are the subsumption of class terms and the classification of individuals into their appropriate categories or classes. The use of OWL-S and OWL allows us to exploit their underpinnings in DLs to construct discovery services for this domain. Here, we have constructed a simple generic discovery service based on the RACER DL engine [10], which loads and maintains the current Profile ontology (including its bioinformatics extensions) in memory. On receipt from a service provider of a service advertisement in the form of (the URL of) an OWL-S document, the Profile of the service is used to classify this instance into its appropriate location in the ontology.

Subsequent queries (also in the form of OWL-S descriptions) can be interpreted as defining a class of services; the instances of those classes that are equivalent to or subsumed by this class are considered to satisfy this query. Note that queries can be as specific or as general as required, and there may be any number of services that meet a particular query, details of all of which are returned to the client. (Others have proposed similar reasoning mechanisms for discovering services — for example, see [12].) It is easy to imagine applying more elaborate reasoning here, perhaps involving aspects of automated composition to formulate and return sequences of services. In addition, the discovery service described above will miss potential service solutions that are more general than (i.e., that subsume) the current query. While this functionality could easily be provided, it raises a problem with the interpretation of the intended semantics of services: should the claim of a service to take input of some class I be interpreted as meaning it can handle *every* instance of (every subclass of) I, or merely some of these instances, of which I represents the 'least general generalisation'? (While the former interpretation would allow for more definitive reasoning about services and is probably the more 'correct' approach, at a pragmatic level the latter use would appear more natural.) Another difficulty surrounds queries which stipulate, for example, a relatively general input class and a relatively specific output class (as would be used when 'probing' the available services for methods that produce a particular desired output). In this case, any particular service (with, say, more specific input and more general output typings) is unlikely to either subsume or be subsumed by the query: a more sophisticated matching algorithm would be required, one which considers these different constituents of the profile description separately. Problems such as these suggest that service discovery based on the simple subsumption of Profile descriptions is unlikely to be adequate for many domains (and undermines some of the rationale for expressing OWL-S in a DL-based language).

[10] A task which presented some difficulties, since — of course — these services lacked any semantic description!

Notwithstanding these shortcomings, we have chosen to construct and deploy the relatively simple algorithm described above to provide a basic functioning discovery service for our environment. This discovery service is itself implemented as a web service, with a WSDL description specifying the appropriate SOAP messages for publishing services and posting queries (these functions can also be performed through a web browser form). Hence, a further assumption about this environment is that the location of this service is known *a priori* to clients and providers alike.

4.5 Semantic Workflow Tool

The architecture outlined above can be used by clients that are able to parse WSDL and generate, send, receive and parse SOAP messages over HTTP, and, for the purpose of semantic discovery and invocation, generate and parse OWL-S descriptions. However, provision of these abilities currently places quite a burden on any prospective user. Consequently, we decided that, in order to ease this burden and provide a means to construct the workflows of services in the manner described in section 3, we would also provide a client-side interface to this architecture, in the form of a domain-independent semantic workflow tool.

Through a graphical interface, this tool allows its users to specify, using the ontology extensions of the domain in question, their (perhaps partial or general) service needs at a semantic level; these are used to generate the appropriate OWL-S queries, which are then sent to the discovery service. If matching services are returned, their descriptions are then used to fill in details (such as additional inputs and their characterisations). By specifying that the output document of one service is to form the input document of the next, 'pipelined' sequences of services can be constructed dynamically. Finally, values can be provided for inputs to the systems, and outputs channelled into specified outputs and the created workflow can be invoked. (This tool is domain-independent; among its parameters are details of the domain ontology extensions to use.)

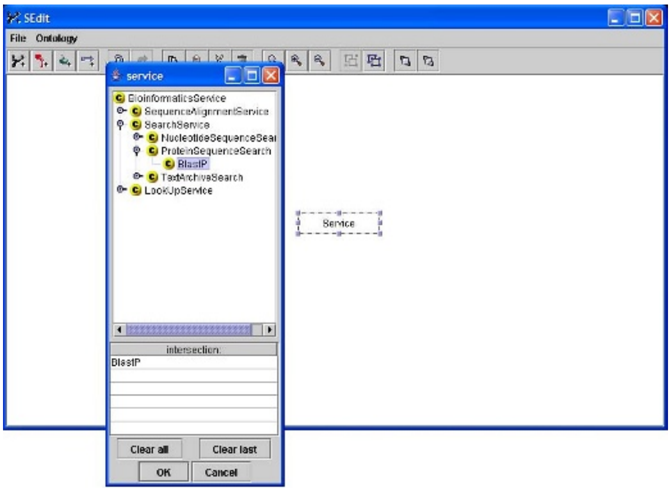


Fig. 3. Using the semantic workflow tool to define a *BlastP* service step

As an example of the use of this tool — and of the use of the semantic environment — we now describe the construction of a sequence alignment workflow as in the scenario of section 2.1. For reasons of brevity, however, we restrict this example to the definition of only the first two service steps in this sequence. The aim of this workflow, then, is to perform a BLAST search over a protein sequence database. Accordingly, the user first introduces a generic service node, and then, using the bioinformatics ontology extensions, browses the hierarchy to specify the class of desired service (figure 3). Having done this, the user then places a call to the discovery service to find if there are any services available which conform to this (partial) service definition.

The response indicates that there is a single available instance of a *BlastP* service (called "SEARCHSIMPLE"), and the user selects this to instantiate this step. This has the effect of elaborating the workflow with the three inputs and single output of this particular service, all named and typed appropriately (figure 4).

The user's next task is to acquire the desired protein sequence that forms one of these inputs; the user does not have this data directly, and so would like to search for a service that will supply it; hence, the user replaces the input with a new generic service concept, and places a call to the discovery service. This corresponds to a request for any service that produces an output of type *ProteinSequence*. This time, the response indicates that two alternatives are available, namely "GETFASTA_SWISSENTRY" and "GETFASTA_PIRENTRY", look-up services which access the PIR and the SwissProt protein databases respectively. Since, in this case, the user wishes to use the SwissProt database, the latter of these is selected, and its input added to the model.[11]

Now, the user can associate appropriate values with each of these inputs (or else read the values from files) and invoke the workflow; the results are displayed to the screen and written to a file (figure 5).

To encourage interoperability and reusability, the workflow is also saved as a file conforming to the SCUFL XML workflow language used by the Taverna Workbench, into which it may then be loaded, executed, modified, etc.

4.6 Summary

It will be useful at this point to reiterate the steps that we performed in order to create this bioinformatics semantic web service-oriented environment from existing computational resources:

[11] Inevitably, the description of services is more complex than is suggested by this example. In particular, the 'pipelining' of services, as in this example, is complicated by the variety of formats used to describe what are conceptually the same input and output data, and frequently results in the need to resolve these mismatches using shim services. (In this example, the pipelined data is — conveniently enough — in the same format.) Ideally, since we are trying to operate at a 'semantic' level, we would like to defer consideration of such 'syntactic' questions until invocation-time. However, since the absence of even a single necessary shim service will prevent successful execution the entire workflow, such matters cannot be so easily ignored. Consequently, we currently model them as first-class semantic services, but the appropriate manner of describing and reasoning with data formats remains an open question.

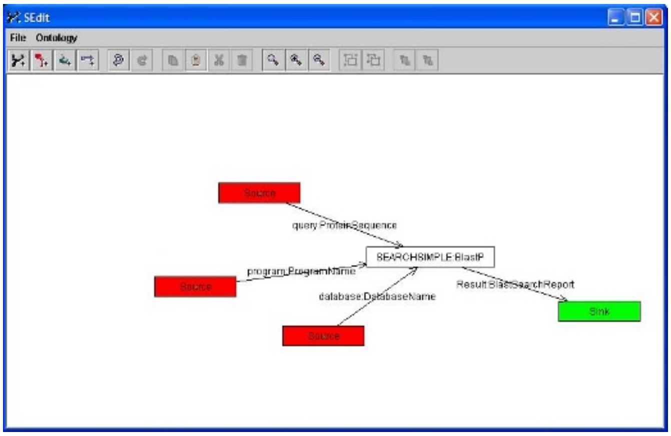
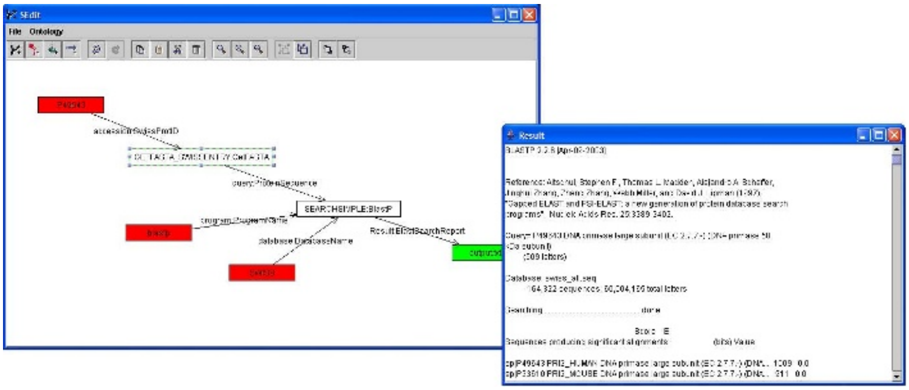


Fig. 4. A service instantiation of the *BlastP* step



```
BLASTP 2.2.6 [Apr-09-2003]


Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs", Nucleic Acids Res. 25:3389-3402.

Query= P49643 DNA primase large subunit (EC 2.7.7.-) (DNA primase 58
kDa subunit)
        (509 letters)

Database: swiss_all.seq
          164,322 sequences; 60,204,136 total letters

Searching..............................done

                                                   Score    E
Sequences producing significant alignments:       (bits) Value

sp|P49643|PRI2_HUMAN DNA primase large subunit (EC 2.7.7.-) (DNA...  1009   0.0
sp|P33610|PRI2_MOUSE DNA primase large subunit (EC 2.7.7.-) (DNA...   911   0.0
```

Fig. 5. Finally, the workflow is invoked and the results can be inspected

1. Analysis of existing web services and of their use by bioinformaticians has allowed us to define the desirable properties of any such environment to act as drivers for the engineering effort;
2. An OWL bioinformatics ontology extension to the OWL-S ontology was engineered; this step involves deciding how best to characterise the domain and differentiate services, from the perspectives of both service providers and potential clients.
3. Using this ontology and the WSDL description of an available service, an OWL-S description of the service is generated (in part manually).
4. Using an existing DL reasoner, a semantic discovery (web) service was constructed. This uses the ontology to classify available services, whose OWL-S descriptions can now be published to this service.
5. In this case, we provided a client-side workflow tool interface that would allow its users to interact with this environment.

By way of an aside, one might expect many of the characteristics noted above to be found in other domains in which existing computational resources are invoked from the command-line using UNIX-pipelines. In its embrace of web services, bioinformatics is a relatively advanced domain; when considering how to 'servicify' similar but less evolved domains, there are a number of issues that arise. These include deciding what should constitute a service in the domain (a convenient rule of thumb might be to consider what constitutes a 'minimal unit of reusability'), how best to expose its inputs and outputs (since command-line programs will often refer to local files and directories) and how to provide the appropriate computational (HTTP, SOAP, WSDL) wrapping around the service. (With others we have addressed some of these issues when creating a semantic environment similar to that described in this paper for the domain of natural language processing; see [8] for details.)

5 Related Work

In this section we highlight some current work that is closely related to that presented in this paper. Within the bioinformatics community, the myGrid project[12] has investigated the use of DAML-S (the precursor to OWL-S) for service description, and one thread of the BioMOBY project mentioned above concerns the semantic description of services, with, as here, the use of OWL ontologies of service and data types. Rather than using the kind of service-oriented architecture adopted here, though, this work is experimenting with an alternative model in the form of the joint development between agents of a 'negotiated' service description. This approach is intended to help counter some of the problems that would occur whenever ontologies or service descriptions alter. However, since this work is still under development, it is not yet possible to judge the merits of this approach.

The work presented here has some overlap with the Task Computing project [14], at least in terms of the underlying semantic web technologies that are adopted, and that project's STEER interface is somewhat similar to the workflow tool developed here. However, there are differences: Task Computing is an ambitous project concerned with pervasive computing, and adopts an appropriate non-centralised architecture for service discovery; moreover, it is aimed at a wider range of potential users than the work here, which is focused on the needs of a particular community.

The WSMO working group[13] is directly concerned with providing semantic service environments, and in some respects represents an alternative to the OWL-S approach and the architecture that it suggests; however, its work is still at a relatively early stage, and does not yet allow a full appraisal of its applicability to this domain. The METEOR-S/WSDL-S [17] project is an attempt to integrate semantic 'type' descriptions (expressed using OWL constructs) more directly into WSDL documents. This sort of approach might be appropriate in this particular case, since the assumptions we make about bioinformatics services mean that, in effect, OWL-S is used for little more than typing of this sort.

[12] `http://www.mygrid.org.uk`
[13] `http://www.wsmo.org`

6 Conclusions

The environment described above supports the interactive construction and execution of workflows, i.e., their realisation in an orchestrated sequence of web services. From the user's perspective, the creation of a workflow can take place at the 'knowledge level' of service types, with calls to the discovery service used to try to ground the workflow in actual computational resources. The choice of a particular candidate service for a given workflow step also has the effect of introducing any additional input and output parameters associated with it into the model. Data flow is achieved by 'pipelining' an output of one service into the input of another. When actual services instantiate all the steps, the workflow input values can be supplied, either as literals or from files, and the workflow can be invoked. Hence, the discovery of services plays an important role in providing access to active services. In comparison with conventional look-up approaches such as UDDI [1], which rely on generic service taxonomies, our discovery mechanism can perform more detailed matching using subsumption over service requests and advertisements. This mechanism, the workflow tool and the underlying architectural aspects of the environment are essentially domain-independent; the specifics of the domain are expressed through the ontology extensions and their use in OWL-S service descriptions.

We conclude with some remarks about the engineering aspects. The need to wrap services with a SOAP messaging layer, and generate the corresponding WSDL and (in particular) OWL-S documents remains an obstacle to those trying to 're-purpose' existing resources as web services.[14] The end-user tools for doing this are not yet readily available, and until such time as they are, take-up of these technologies will necessarily be limited. At a more general level, the suitability of the OWL-S ontology and OWL itself, and, more specifically, their underlying grounding in DLs, for the purpose of describing services remains questionable. As discussed in section 4.2, DLs do not readily lend themselves to the description of processes. Furthermore, as seen in section 4.4, service discovery based simply on the subsumption of Profiles is not always going to be adequate. The workflow tool that we have developed currently has a rather limited vocabulary for specifying workflows; for instance, iterations over resources cannot be specified in the language, but must be encapsulated in a composite process. In part, this is a result of the desire to provide a simple graphical interface, which does not lend itself to subtleties of this sort (and, of course, users with more advanced needs could always revert to a conventional programming language to specify their workflows). However, another factor is that the contents of an appropriate workflow language(s) for bioinformatics (and for e-Science more generally) are, as yet, not entirely clear. Indeed, this — along with other aspects of this environment — is something that one might expect to evolve as semantic web services are assimilated into scientific research methodologies.

[14] To address part of this problem, the myGrid project has developed Soaplab, to help provide a SOAP wrapper for programs; see `http://industry.ebi.ac.uk/soaplab/`.

Acknowledgements

This work is supported by BBSRC grant BBSRC 15/BEP 17046 (XSPAN), and by the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University.

References

1. T. Bellwood *et al.* UDDI technical white paper. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.
2. E. Birney *et al.* An overview of ensembl. *Genome Research*, 14:925–928, 2004.
3. A. Brazma *et al.* Minimum information about a microarray experiment (miame): toward standards for microarray data. *Nature Genetics*, 29(4):365–371, 2001.
4. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL). http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.
5. The OWL Services Coalition. OWL-S: Semantic markup for web services (v1.1). http://www.daml.org/services/owl-s/1.1/.
6. The MINDSWAP Group. OWL-S API. http://www.mindswap.org/2004/owl-s/api/.
7. The Web Ontology Working Group. OWL web ontology language reference. http://www.w3.org/TR/owl-ref/.
8. C. Grover, H. Halpin, E. Klein, J.L. Leidner, S. Potter, S. Riedel, S. Scrutchin, and R. Tobin. A framework for text mining services. In Simon J. Cox, editor, *Proceedings of the UK e-Science Programme All Hands Meeting 2004 (AHM 2004)*, pages 878–885, Nottingham, UK, 2004. 31st August-3rd September.
9. M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, and H.F. Nielsen. Simple object access protocol (SOAP). http://www.w3.org/TR/soap12-part1/.
10. V. Haarslev and R. Möller. RACER system description. In *Proceedings of the First International Joint Conference on Automated Reasoning*, pages 701–706. Springer-Verlag, London UK, 2001.
11. D. Hull, R. Stevens, P. Lord, C. Wroe, and C. Goble. Treating shimantic web syndrome with ontologies. In *First AKT workshop on Semantic Web Services (AKT-SWS04), KMI, The Open University, Milton Keynes*, 2004.
12. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 331–339. ACM, 2003.
13. P. Lord, S. Bechhofer, M.D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. In *Proceedings of Third International Semantic Web Conference (ISWC2004), Hiroshima, Japan, November 2004*, pages 350–364. Springer-Verlag LNCS 3298, 2004.
14. R. Masuoka, Y. Labrou, B. Parsia, and E. Sirin. Ontology-enabled pervasive computing applications. *IEEE Intelligent Systems*, 18(5):68–72, 2003.
15. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics Journal*, 20(17):3045–3054, 2004.
16. S.C. Potter *et al.* The ensembl analysis pipeline. *Genome Research*, 14:934–941, 2004.

17. K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *Proceedings of the 1st International Conference on Web Services (ICWS'03), Las Vegas, Nevada (June 2003)*, pages 395–401, 2003.
18. L. Stein. Creating a bioinformatics nation. *Nature*, 417, 2002.
19. R. Stevens, R. McEntire, C.A. Goble, M. Greenwood, J. Zhao, A. Wipat, and P. Li. myGrid and the drug discovery process. *Drug Discovery Today: BIOSILICO*, 2(4):140–148, July 2004.