Bioinformatics: From Disparate Web Services to Semantics and Interoperability

Mikael Åsberg, Lena Strömbäck
Department of Computer and Information Science
Linköpings Universitet
Linköping, Sweden
Email: m.asberg.watch@gmail.com, lena.stromback@liu.se

***Abstract*—In the field of bioinformatics, there exists a large number of web service providers and many competing standards regarding how data should be represented and interfaced. However, these web services are often hard to use for a non-programmer and it can be especially hard to understand how different services can be used together to create scientific workflows. In this paper we have performed a literature study to identify problems involved in developing interoperable web services for the bioinformatics community and steps taken by other projects to address them. We have also conducted a case study by developing our own bioinformatic web service to further investigate these problems. Based on our case study we have identified a number of design issues important to consider when designing web services. The paper is concluded by discussing current approaches aimed at making web services easier to use and by presenting our own proposal of an easy-to-use solution for integrating information from web services.**

***Keywords*-bioinformatics; XML; web services; interoperability; semantics**

I. Introduction

In our previous work [1] we studied the interoperability of web services within the bioinformatics community. This article extends the work by providing a more comprehensive literature review, more details on our work, and a first description of our current work in the field.

In the field of bioinformatics, there has been an explosion of data over the past years. For example, in the Molecular Biology Database Collection: 2008 update [2], 1078 databases are listed, 110 more than in the previous update [3], which itself also contained 110 more databases than the update before that. These databases are being maintained and hosted by a large number of autonomous service providers. Many of them are only concerned with a single database and its tools. Regarding how data should be represented and formatted and how its corresponding tools and algorithms should be interfaced, there exists a large number of standards [4] [5]. As an example, P. Lord et al. say in [6] that ”there are at least 20 different formats for representing DNA sequences, most of which have no formal specification”. Many standards and specifications evolved in an ad-hoc fashion and there is no wide-spread agreement on when a particular standard should be used.

Remotely accessing the resources maintained by the service providers can often be done in more than one way. Many service providers have constructed www-based interfaces to their resources, meant to be used by humans. The user does not have to use any specialized software or use a specific platform to access the resource, a common web browser is sufficient. As an alternative to browser-based interfaces, many service providers in the bioinformatics community offer programmatic access by using web services [7]. A web service is any service available over the Internet using XML as its messaging system, and it is independent of platform and programming language. The web services we specifically mean here are those falling under the category of RPC (remote procedure call) web services, being parts of Internet API:s. Such web services allow for programmatic and batch access. An example web service method from the bioinformatics community is *get_genes_by_enzyme*, which is part of the web service API that allows access to KEGG (Kyoto Encyclopedia of Genes and Genomes) [8]. This particular method takes an organism and an enzyme ID in string form and returns its corresponding genes. Another example is EBI Soaplab Web Services for EMBOSS programs [9], which is a large set of web services providing web access to many EMBOSS programs.

Given the nature of biology, a bioinformatician often has to use multiple service providers to conduct his or her research. By combining several resources and processes, local or remote, bioinformaticians can create scientific workflows [10]. There exists a number of different tools for creating scientific workflows, for example Taverna [11] [12] [13] and VisTrails [14] [15]. These tools allow the user to create a data pipeline, a workflow, using a number of different resources. Taverna caters specifically to bioinformaticians while VisTrails was originally targeted at visualization. An alternative to flexible workflow systems like Taverna and VisTrails are specialized bioinformatic grids [16] that tie remote and local resources together in a client to provide a unified view. Such grids can often be successful on a small scale, but development costs and network restrictions prevented them from becoming the de-facto standard for integration of bioinformatic services [16].

However, the task of creating bioinformatics workflows can be very difficult because of the fact that there are so many service providers and there is little consensus about data formatting and interfacing. Thus, bioinformaticians face a massive interoperability problem. A browser-based form is a convenient way to work if one just wants to do a few

stand-alone look-ups in a given database, but when one needs to perform many look-ups and queries on several disparate resources it becomes unfeasible. This is true not just because of the fact that these web-based interfaces may be poor at supporting programmatic and batch access, but also of the intricate data reformatting that may be involved in using the output from one resource as input to another. Web services, by themselves, do not solve the interoperability problem. They are simply a way to access remote resources programmatically. A problem facing users is to determine how different sets of web services can be used together to create the desired workflows.

In this paper we have performed a literature study to give an overview of other projects that have been introduced to enhance semantic interoperability and discoverability for bioinformatic web services. Section II discusses the problem with designing interoperable web services and Section III discusses semantic frameworks that have been introduced in bioinformatics community to alleviate the problem of lacking interoperability. In Section IV we discuss scientific workflow systems that are becoming a more and more popular way of interacting with multiple bioinformatic resource providers to form workflows. This way of working has many benefits, but introduces new problems that need to be considered. Section V discusses our case study, where we developed our own web service to get a hands-on experience of the problems involved with web service design. The case study was also aimed at designing a web service capable of performing data integration on the fly. Section VI summarizes the problems and design issues we encountered. In Section VII we end the paper by discussing other approaches to interoperability and also present our continued work, BioSpider.

II. Designing interoperable web services

Our main issue is how to design web services for the bioinformatics community that are interoperable with other services. We focus on technologies used in this community.

SOAP-based web services are sometimes called XML-based web services. Everything that is passed between the users of a web service and the web service itself is in XML form. This means that input parameters and output data for a given web service method are also serialized in XML. In order for clients to use a particular web service, they need a description, the *WSDL*, to learn about, which methods are available, what parameters those methods require and what kind of data is returned.

As an example, let us create a web service of the Java method (showing its signature only) *String getXItem(String id)*. It accepts a single string and returns a single string value. In Figure 1 we see what the created data types for our example method looks like in the WSDL description.

The WSDL description shows that for our example method, two complex schema types have been declared, one for the input parameter and one for the result. However, these types are simply wrappers around the XML Schema primitive type *xs:string*. Herein lies the interoperability problem, because these types have no semantics attached to them. We do not know the semantic meaning of the input and output data, only their syntactic meaning!

```
<xs:complexType name="getXItem">
   <xs:sequence>
      <xs:element minOccurs="0"
                  name="arg0" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="getXItemResponse">
   <xs:sequence>
      <xs:element minOccurs="0"
                  name="return" type="xs:string"/>
   </xs:sequence>
</xs:complexType>
```

Fig. 1. Type Declarations in WSDL-file

When programming in a strongly-typed language like Java or C# and one needs to have semantics for syntactically simple types (like strings or integers), the common practice is to introduce new classes or hierarchies of classes. By doing that, we can enforce semantic rules at compile time and prevent, for example, scenarios where users are passing strings that represent journal abstracts to methods expecting database identifiers in string form. However, this will not work in a web service context, because, in order to achieve interoperability between different programming languages and platforms, the types used in the host language must have corresponding XML Schema types like integers or strings.

The WSDL description for a web service can be seen as the grammar of that particular web service. Thus, all providers have their own grammars, albeit specified using the same XML language constructs. In order to achieve true semantic interoperability we need a common grammar, a common stock of reference. Within the bioinformatics community, a number of different semantic frameworks have been proposed to provide a common grammar, for use as a common stock of reference regarding data types, naming, operations etcetera.

Another thing to consider is that not all parameters for a given web service method may hold the same merit. Some parameters constitute the fundamental data needed by the service. These are the important parameters. But many methods also accept additional parameters, whose purpose is simply to tweak the behavior of the service. When looking for semantically suitable web services, a bioinformatician is not interested in these kinds of parameters as they would only serve to complicate the task at this point [6].

A problem with the web services themselves can also be that a given method is semantically simple and has a limited and well-defined purpose. This programming paradigm is mostly considered a good thing, but if the building blocks of a workflow become too fine-grained it will be difficult to create more advanced workflows without requiring a lot of ”glue” in the form of data formatting or even programming. However, it may be the case that possible interoperability between two web services is easier to establish if their parameters are fundamentally simple and well-defined. This leads to the main point of this article: how to design web services with clear

semantics that can be easily used in bioinformatic workflows.

In the following section we will give a presentation of some of the more notable semantic frameworks that have been introduced to provide a single grammar to describe a host of autonomous service providers.

III. Semantic Frameworks in Bioinformatics

Several projects have attempted to address the above problems and here we give a brief overview of the three most important in the field of bioinformatics today: BioMoby [17], ^{my}Grid [18], and the more recent BioCatalogue [19] [20]. These projects tackle not only the problem of helping users to determine how services from different providers can work together to form workflows, but also assist in the discovery of relevant services.

A. BioMoby

The BioMoby project was initiated in 2001 and has a main branch, MOBY-Services (MOBY-S) and a sub-branch known as SSWAP (Simple Semantic Web Architecture and Protocol). However, according to the BioMoby web page [21] these two branches are to be merged in the future.

In the **MOBY Services** branch three ontologies provide semantic discoverability and interoperability along with strictly enforced naming rules, and we present them briefly below:

- **Service ontology** - this ontology contains operational classifications used to label web service methods. There is a root type called *Service* and a tree of sub-types, such as *alignment* or *rendering*. The purpose of this ontology is to assist in the discovery of web service methods that are useful or interesting for a given task or problem.
- **Namespace ontology** - this ontology can be seen as a flat list of different namespaces that can be used to semantically describe data that is consumed and/or produced by a web service method. It also enforces a method for how to name identifiers. A problem in the bioinformatics field is that identifiers have not been named in a consistent and reliable way. The namespace ontology derives information from Cross-Reference Abbreviations List from the Gene Ontology consortium [22] and defines distinct naming rules that are used dependably and reliably. For example, *Antirrhium majus* (Snapdragon) gene names live in the *DragonDB_gene* namespace.
- **Object ontology** - this ontology is similar to the service ontology in the sense that it can be viewed as a tree. The root node is called *Object* and between all nodes and its children there exists an *IS-A* relationship, e.g., an *AlignedSequence* is an *Object* and an *AlignedDNASequence* is an *AlignedSequence*. Two nodes, in different parts of the tree, can also have a *HAS-A* relationship (either one-to-one or one-to-many), e.g., the type *Annotation* is a direct child of *Object* and it has two *Integers*, but is itself not an *Integer*. An instance of an object that is returned from a web service method is serialized into XML and contained in an envelope (that is also XML). Envelopes can contain cross-references that are supplied by the data provider and they describe related pieces of information. Since cross-references are valid objects by themselves they may be used directly, without additional formatting or computation to discover services from other providers that operate on them [23]. In Figure 2 we can see a small fraction of this ontology.

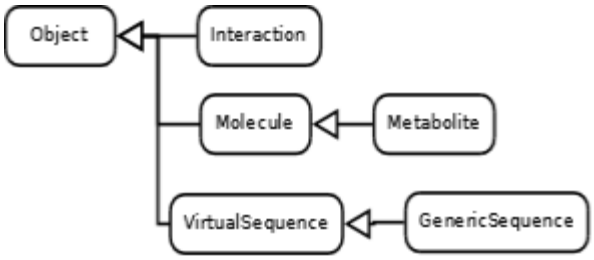

Fig. 2. Fraction of BioMoby object ontology

The BioMoby team runs Moby Central, which is a web service registry where all service definitions can be found, expressed in the ontologies described above. When a new actor wants to participate in the BioMoby project they register their objects, namespaces, and service classifications in the registry. This implies something that is very important and fundamental to BioMoby: the ontologies described above are *end-user extendable*.

SSWAP stands for Simple Semantic Web Architecture Protocol and was previously called Semantic-MOBY or simply S-MOBY. The approach taken by this sub-branch differs from the one taken by the MOBY-Services project. Instead of maintaining three user-extendable ontologies centrally, it defines a minimal messaging structure and relies on the wealth of the available third party ontologies, such as OBO [24], to define meaning, syntax and interoperability between services. In [17] the authors say that *"SSWAP has shown exciting early success in achieving interoperability between a small number of participating providers. It remains to be seen, however, if the complexity of reasoning over an open-world system, and/or the potential dilution of compatibility between resources due to an increasing number of ontological possibilities, will interfere with the desired goal of straight-forward, maximum interoperability between bioinformatics Web resources"*.

B. ^{my}Grid

The ^{my}**Grid** project is a part of the UK government's e-Science program and it seeks to enable bioinformaticians to discover and chain together disparate bioinformatics services to create workflows, *in silico* experiments. For describing a bioinformatics domain and the properties of its services there exists a domain ontology that is stored centrally and maintained and generated by an expert. Semantic descriptions of services are made using a lightweight RDF data model using terms from the domain ontology. These descriptions are extendible by users. The project makes a distinction between domain services and shim services. Domain services are the centerpieces, those that perform scientific functions. The archi-

tecture should not automatically select those for the scientist, because there could be alternatives that only the scientist doing the experiment should select among. Shim services, on the other hand, does not have a scientific function per se, but are only used as glue to connect two domain services that are not directly compatible. The architecture should automatically insert available shim services where they are needed without the scientist having to intervene. The Taverna software can interact with ^{my}Grid-based sources, and also with BioMoby-ones through a plugin [25].

C. BioCatalogue

The BioCatalogue project [19] [20] was launched in June 2009 and is a joint venture between EMBL-EBI and the ^{my}Grid project. It aims to assist web resource users within the Life Sciences community to find relevant services for their research and assist in determining how different services can operate together. Another goal is to act as a registry for suppliers of services, which will allow any given supplier to increase the size of its user base.

The project tries to tackle the problem of services becoming stale, disappearing, or changing by monitoring both their actual availabilty and their supposed function. To put it another way, the four main issues listed on their homepage (http://www.biocatalogue.org/) effectively point out the objectives of the project:

- *"Web Services are hard to find."*
- *"Web Services are poorly described"*
- *"My Web Services are not visible"*
- *"Web Services are volatile"*

By finding solutions to the problems listed above, the BioCatalogue project aims to become the central hub for web services in the Life Sciences community. It doesn't aim to become a supplier of scientific web services themselves, only to bring the vast variety of already existing ones under one umbrella to the benefit of the community.

The BioCatalogue project has noted that finding an interesting web service is only part of the problem. It's not enough just to have a central registry of services where suppliers can register their service. Often the documentation for a service is mediocre or outdated and comes with few or no examples, making the service hard to use. Some services also require certain operations to happen in sequence to be able to produce a meaningful result, and this fact is not always clear from the get-go.

To minimize this problem of understanding a given web service, the BioCatalogue project employs rich annotations for all registered services. The annotations for a given service are not derived from a single source (e.g., the supplier of the service). Instead it's comprised of information from several parties: the supplier of the service, a domain expert curator employed full-time who has sub-curators to assist him or her, the user base itself, and usage patterns that are automatically collected. Together, these entities evolve the annotations for a given service over time to make it better and more consistent. The role of the curator is to oversee the process to ensure that guidelines are followed in order to avoid annotations to be given in an inconsistent manner. Annotations for services can be divided into four main categories:

- Functional - these annotations describe the purpose of the service, what kind of operations it can perform and what data it will operate on and produce. This category is further divided into sub-categories, pin-pointing the task of the service more explicitly, e.g., alignment or text mining. Example input data and other usage scenarios often accompany these annotations in order to help users.
- Operational - these annotations detail any particular considerations that must be taken into account in order to successfully use the service. As noted above, some services require operations to happen in sequence (i.e., the individual methods of the service cannot be seen as separate islands) in order to produce a sensible result. Such things are annotated in this category.
- Profile - here automatically collected data and other comments by users regarding the service are maintained.
- Provenance - contains information about the supplier of the service and an audit trail detailing any changes to service over time are kept here.

BioCatalogue can be accessed through a web portal [20], but an API is also provided to allow programmatic access. A user using the web portal can look for services by searching for scientific function, data types, provider, country etcetera. If the user is unsure about the type of some data he or she has, BioCatalogue can analyze an excerpt of it to determine its type. Regarding data, input and output data are tagged with ontological terms from the ^{my}Grid project and if the users know that information it's straightforward to find services that fit perfectly semantically. All services from the ^{my}Grid project have been imported into BioCatalogue and work is currently underway to merge with other big repositories like BioMoby.

IV. Scientific Workflow Systems

The main aim of this paper is to study interoperability of web services, i.e., how web services can be designed to be easily used together to solve an information integration task. From a technical point of view there are several ways to combine web services into more complex tasks, however, one approach in common use within the bioinformatics community is scientific workflows. Scientific workflow and workflow-based systems [11] [12] [13] [14] [15] [26] [27] [28] have emerged as an alternative to ad-hoc approaches for documenting computational experiments and designing complex processes. They provide a simple programming model whereby a sequence of tasks (or modules) is composed by connecting the outputs of one task to the inputs of another. Workflows can thus be viewed as graphs, where nodes represent modules and edges capture the flow of data between the processes.

The actual features and representation of a scientific workflow differ between the systems, due to varied needs from application areas and users. There is ongoing work to create one common model for provenance, e.g., the Open Provenance Model [29] and a mediation approach [30]. However, currently

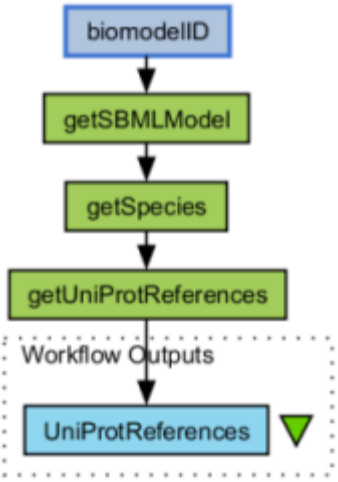

Fig. 3. Sample workflow illustrating iteration in Taverna

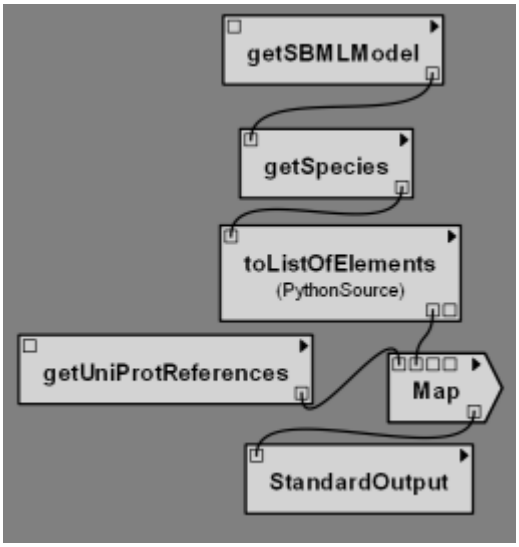

Fig. 4. Sample workflow illustrating iteration in VisTrails

systems tend to work in their own internal format albeit it is becoming common to provide conversion to other formats. In practice this means that a web service can be more or less easy to use within the scientific workflow framework dependent on its design and the available features for the chosen tool.

In this work we have chosen to work with two different workflow systems, VisTrails [14] [15] and Taverna [11] [12] [13]. VisTrails is a workflow system that supports exploratory computation tasks. It has a graphical user interface that is used for the composition and execution of workflows. Data and workflow provenance is uniformly captured to ensure reproducibility of results by others. Workflows can be composed by program libraries (Python) or by external web services. VisTrails has been used in the fields of biology and earth science. Taverna is designed specifically for bioinformatics applications. As VisTrails, Taverna has a graphical user interface for creating and executing workflows. Workflows are composed by making use of external services such as web services, BioMart, BioMoby and SoapLab services.

There are several differences between VisTrails and Taverna in terms of how they represent provenance and what functionality they offer. However, for this work the most important difference is how they represent iteration, which is a common task in bioinformatics. Taverna offers a straightforward solution. Figure 3 shows an example of a Taverna workflow. Whenever a module returns a list of results the next module is iteratively applied on all results in the list. In this case *getSpecies* returns a list of all species in the model and *getUniProtReferences* is applied on every species in the list. VisTrails does not offer this feature, instead they offer a number of control flow modules. This includes control flow modules such as conditions and a map module for iteration. By using the map module we can apply the next module on all results in a list. The resulting VisTrails workflow corresponding to the Taverna version is shown in Figure 4.

V. Case Study

To further explore the design issues encountered when designing bioinformatic web services, we have performed a case study where we implemented our own service. The main objectives of the service were that it should be easy to use, it should help with data integration, and the semantics should be possible to model in frameworks such as MOBY-Services. Regarding data integration, we wanted to investigate ways of following links between different data sources using a single service. Right now, our service links three databases together: the (curated) BioModels Database [31], the UniProt Knowledgebase (UniprotKB) [32], and the RCSB Protein Data Bank (PDB) [33]. The service is not intrinsically bound to just these three databases forever, but could be expanded to work on others.

A. Design of the web services

The web service is written in Java [34] using the Eclipse Web Tools Platform [35], but it is platform neutral in the sense that it can be used from any WSDL-enabled language and platform without any additional dependencies. Using the SBML library [36], the service loads SBML models from the BioModels database, and the loading of a model can be seen as an entry point to the web service. From this model the user can extract UniProt references that can be found in the annotations for some species in the BioModels data. The UniProt references are used, by the server, to obtain the corresponding UniProtKB XML-files and, from those PDB-references can sometimes be extracted. A PDB file is returned to the caller who can use it for visualization. The service can be seen as having several stateful subsets: when you ask questions about BioModels or its species you need a BioModel or species ID, but when you obtain a UniProt reference then that becomes the key you use and likewise the PDB reference

```
getSBMLModel(biomodelFileName) : BioModel ID
getSpecies(modelID) : Species IDs
getNthSpecies(modelID, n) : Species ID
getNumberOfSpecies(modelID) : Number of Species
getSpeciesSBMLID(speciesID) : Species SBML ID
getSpeciesSBMLName(speciesID) : Species SBML name
getUniProtReferences(speciesID) : UniProt references
getPDBReferences(uniprotID) : PDB references
getPDB(pdbReference) : PDB file
```

Fig. 5. Web Service Method Listing

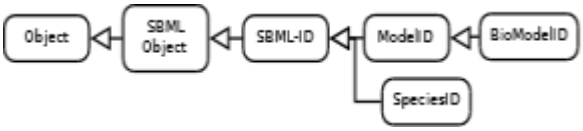

Fig. 6. Type ontology for our web service

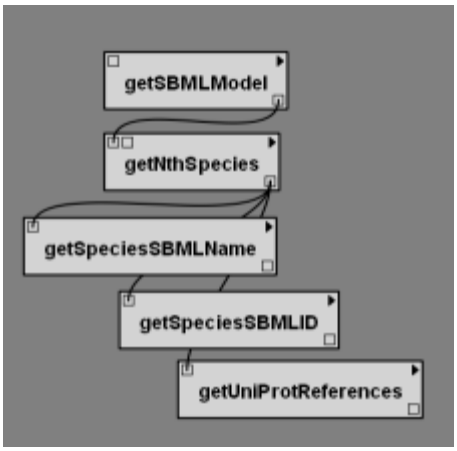

Fig. 7. Fetching name, id, and references for a given species

becomes the identifying token that is used for the PDB part of the service. In figure 5 we see a listing of the core methods.

Figure 3, from the previus section, is an example of a Taverna workflow utilizing the web service. It's a straightforward workflow that loads a BioModel. From the BioModel a list of ID:s to its species is obtained and for each species any UniProt references are determined. The method *getUniProtReferences* that is called only takes a single species ID so Taverna automatically iterates over our list of species ID:s and calls the method once for each. The resulting output is a list of lists that shows how species are annotated with UniProt references. In Figure 4, the previous section also showed the same workflow modeled using the VisTrails system. The VisTrails variant of the workflow is a bit more complicated than its Taverna counterpart. This is due to the fact that iteration is not automatic in VisTrails, but modeled in the workflow itself, using special control flow modules like map.

BioModel-files and UniProtKB-files are never passed verbatim between the client and the server, instead ID:s are used to identify a particular file. The only large data object that is passed is the output of the *getPDB*-method that represents the PDB itself. The server stores the latest release of the curated biomodels in a database and it will fetch UniProtKB-files and PDB-files when they are needed from their respective resource providers. Any downloaded files are cached for performance reasons.

All input and output parameters to the methods that make up the web service are simple types like strings or integers. Since all types used can be modeled using XML Schema primitives we are not tied to a particular platform. It also means that our interface is fit for modeling in MOBY-Services. A question that arises when modeling an interface in MOBY-Services is if one should use namespaces or types for new items. We have chosen to define new types. Figure 6 shows how the types can be realized for our service. For UniProt and PDB references there are already existing namespaces defined that can be used. See also the descriptions of input and output parameters in Figure 5.

Determining the proper level of granularity of your interfaces is hard. If it's too fine-grained, you could end up with an explosion in the number of methods that might require glue to perform complex tasks. If you design a service where the methods are complex and operate on complex data structures the service will tend to be tied to a very limited set of tasks and it might require the user to perform data-reformatting tasks, especially if he or she wants to use other services as part of his or her workflow. In our implementation the goal was to design a service that ties together several different databases by following links in the data while at the same time having a simple interface to allow for interoperability and to severely limit the need for the user to reformat data. If the primary goal of the service was to obtain PDB data then a much more compact interface could have been made, where the methods *getSpecies*, *getUniProtReferences*, and *getPDBReferences* could have been replaced with a method that given a BioModel ID returns a list of PDB:s. Such a method would be very easy to use for that particular task, but it could not be used in some other context.

B. Using the web services

The design of the web services is important for its ease of use and the possibility to combine the services into extended functionality. In this section we will exemplify and discuss two of the major design choices that we considered in our case study. These were to design a service that ties together several different databases by following links in the data and providing a simple interface to allow for interoperability and to severely limit the need for the user to reformat data. This is demonstrated by the two examles in Figure 4 and 7.

Here we use the services to find references to information in the UniProt database, but by combining them in another way the user can select other information about each species.

If the primary goal of the service was to obtain PDB data then a much more compact interface could have been made, where the methods *getSpecies*, *getUniProtReferences*, and *getPDBReferences* could have been replaced with a method that given a BioModel ID returns a list of PDB:s. Such a

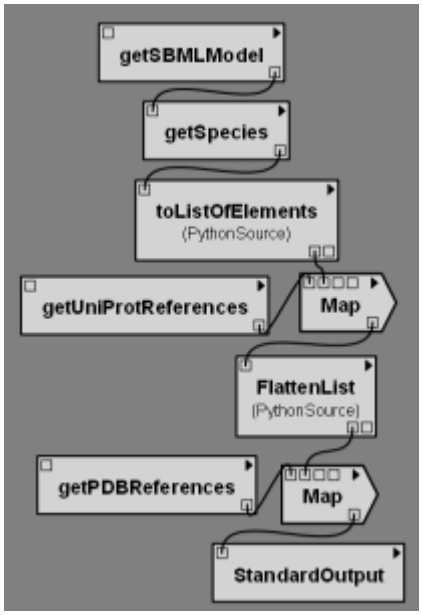

Fig. 8. Find PDB references in VisTrails

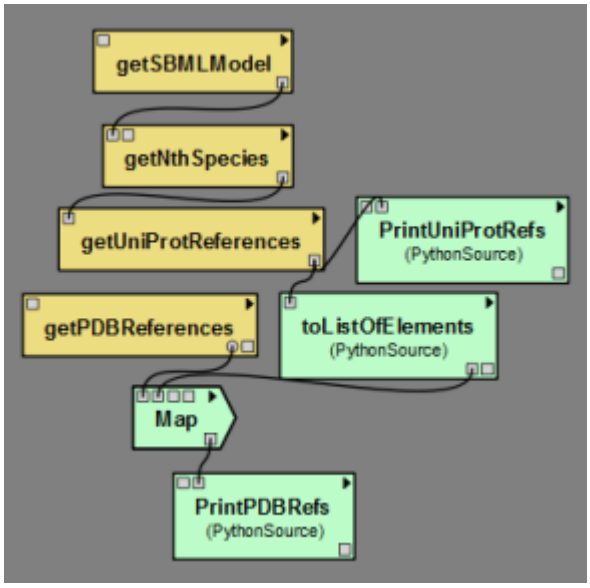

Fig. 9. Find PDB references using parameter exploration and iteration

method would be very easy to use for that particular task, but it would not have allowed the variation demonstrated by the previous example.

As shown by pervious examples, in bioinformatics, it is common to have lists of data as the result of some operation that one might want to pass to another operation. Even though both Taverna and VisTrails have facilities for creating iterative control flows it still remains a complex task. As an example, we can study the workflow where the user loads a BioModel, obtains an ID for each of its species and for each species determines any UniProtreferences. So far, this is the same example as before, but what if we want to follow the UniProt links to find any PDB references? The more straightforward solution is shown in Figure 8.

Here we reformat the list of UniProt references and find a set of PDBreferences for each of them. The drawback with this solution is that the connection between each species and the resulting PDBreference is lost. Maintining them requires building complex datastructures during the iteration. In Figure 9 we show an alternative solution in VisTrails. For iteration, the workflow utilizes a combination of VisTrails features for parameter exploration and control flow modules like maps.

This implementation prints the desired results of the workflow. In Figure 10, we see the output of the workflow from VisTrails when we are using BioModel BIOMD0000000003. Three short Python scripts are used for printing results and for data type conversion. A main objective when constructing this workflow was to preserve information regarding links, i.e., we want to know exactly how species are annotated with UniProt references and how UniProt references links with PDB references. This goal has been fulfilled. A careful study of the output reveals that only the species with index one has UniProt references and gives *findPDBReferences* something to work on. Since map returns a list of lists we can tell how UniProt references are connected to PDB references. In this case they both point to the same PDB file.

VI. Design issues

In this section we summarize our experiments identifying the important design issues to consider when designing web services for bioinformaticians. The goal is to allow bioinformaticians to create complex workflows using multiple service providers that are easily located and those workflows should require a minimum amount of glue in the form locally-performed programming. We have divided these issues into the following categories (in no particular order): semantics, chainability, granularity, data representation, and data passing.

A. Semantics

When creating workflows, bioinformaticians need to be able to automatically discover services that perform some scientific function or find services that either produce or consume data in some given format. The key to enabling this automatic discovery is semantics and here is where frameworks such as MOBY-Services or ^{my}Grid come into the picture. Registering your service in frameworks such as MOBY-Services or ^{my}Grid will allow users to automatically find it when it fits at the semantic level and not just at the syntactic level.

```
Species 0 UniProt refs: []
Species 0 PDB refs: []
Species 1 UniProt refs: ['P24033', 'P35567']
Species 1 PDB refs: [['1P2A'], ['1P2A']]
Species 2 UniProt refs: []
Species 2 PDB refs: []
```

Fig. 10. Find PDB references workflow in VisTrails - output

B. Chainability

Chainability is closely related to semantics, because we need semantics to be able to attain it. Chainability means the ability to automatically list consumers for a given type of data, consumers that fit semantically. Sometimes a service cannot work on your data directly but may be able to work on some subset of the data or requires re-formatting of the data. This requires so called shim services that perform extraction or formatting on some existing data to make it usable by another service, but have no scientific domain function by themselves. Shim services are important to discover automatically. One step towards realizing semantically correct chainability has been made through frameworks such as BioMoby and ^{my}Grid where the semantics of the function of a web service and of its input and output data is specified. However, the framework must also be clever enough to suggest shim services automatically. Say a user has some piece of data in a given format and wants to perform some scientific function on it. The data might need to be reformatted before the methods performing a scientific function can operate on it. The reformatting is done by shim services, and those should be discovered automatically by the framework even though they themselves do not perform the scientific function the user was searching for per se. In [37], D. Koop et al. presents a method of suggesting suitable services by using predictions.

C. Granularity

Should we make stateless or stateful web services? How fine-grained should they be? Very fine-grained methods could be seen as following the programming paradigm of divide-and-conquer. A large problem is broken down to a large number of very small, restricted, and well-defined mini-problems. In a procedural context one could then image writing a procedure (a function, a method) for each mini-problem and this would allow for modularity, ease of testing, re-usability, and ease of documentation. It might not always be the best way when catering to non-programmers because it becomes a complex task of putting all the pieces together in a workflow if there are lots of them. However, if the building blocks in the toolbox are all fundamentally simple, this might lead to a lot of glue in workflows in the form of shim services or programming. On the other hand, if the function of a service is fundamentally simple, it will be easier to describe its semantics. This allows for interoperability. A complex web service where too much functionality has been shoe-horned into a few methods will be very hard to use outside the particular purpose it was designed for. This kind of web service would be hard to use as a tool in a toolbox.

Another issue to take into account is whether the service should operate on lists or single items. Operating on lists can be tempting for performance reasons due to overhead associated with web service calls. However, the output from such methods will become more complex and it may be difficult or impossible to tell how results are connected to input data. The capabilities of workflow tools regarding support of iterative control flows would influence the design of the web service.

D. Data representation

One very important issue when designing a web service operating on complex data is the data format, such as the SBML model. The choice of representation is important when passing complex objects as arguments between web services, but also to enable an understanding of the semantics of the services. In our case we have chosen to use available XML standards for bioinformatics [4] [5], such as SBML and UniPROT. This is a benefit, as it makes the functionality of the service transparent to anyone familiar with the standard, e.g., in our case the naming and the functionality of the SBML services have a direct relationship to the entities defined by SBML.

Another aspect is that data representation for web services is closely related to data formats available for export on the web. Therefore it is natural to reuse the work already invested in this area instead of inventing new representations. Our case study shows that using available data formats works well. In addition we avoid unnecessary conversions of data by using formats where data is already available.

E. Data passing

When using web services we want to avoid passing large amounts of data back and forth when we do not need to. If the data is not generated by the client and is available in the public domain and it is the responsibility of the web service to manipulate or study this data in order to compute some result that is to be returned to the caller, then that data should stay on the web service server as much as possible. The clients should only see the results they are interested in. In our example implementation we found that using IDs worked well as links to pieces of data.

VII. Other Approaches

In the previous sections we have discussed interoperability issues regarding bioinformatic web services, we followed with a look at some semantic frameworks that have been introduced to tackle those issues. We also presented a detailed case study where we identified and discussed design issues related to the construction of web services.

In this section we will present three other approaches that aim to make web services easier to use for bioinformaticians. First we present TogoWS [38] [39], which is a web service and data-integration proxy for a number of service providers. We follow that with a presentation of SeaHawk [40], that serves as a front-end to the BioMoby framework presented earlier. The section is concluded with a presentation of our future work, the BioSpider, which in short can be described as plugin-based framework for modelling disparate, but yet connected bioinformatic web resources.

A. TogoWS

The TogoWS project [38] [39] sprung out from ideas seeded and problems recognised at the BioHackathons in 2008 and 2009. In these BioHackathons it was concluded that interoperability was a major problem in bioinformatics, not only because of data representation but also because technical decisions made by some certain service providers, forcing clients to use a particular programming language or environment. It was also noted that there are projects aimed at tackling this problem, such as BioMoby, but that many providers have not made their services compatible with these frameworks due to the hefty investment in server-side work required, making these proposed solutions not completed.

Due to the amount of data reformatting required to be able to use data from one service provider with another provider, a large number of third-party, client side libraries have been developed. There are libraries for different programming languages aimed at different tasks and these include the set of libraries (BioPerl, BioPython, BioRuby, BioJava) provided by the Open Bioinformatics Foundation [41]. However, the team behind TogoWS decided that their service should itself provide the data reformatting features offered by these libraries. This would relieve the user of the burden of having to install these libraries and write code to interface with them.

The team behind the TogoWS project decided to build a web service frontend for different suppliers, acting as a proxy between them and the users. Several major service providers were included under this umbrella. Operations were divided into two main categories: data retrieval and analysis. It was decided that using a REST-based API was best for data retrieval because then a uniform URI-scheme for the participating databases could be devised. For analysis operations, it was decided that SOAP was the best option because here results returned and parameters can be very complex and running time can be substantial, making REST a less than ideal choice. In the first phase, a unified SOAP-based interface was developed for several service providers located in Japan, and certain technical limitations found in some services were worked-around.

By developing a front-end that is itself a web service too, the clients can continue to use whichever tools they like to call these services and instead of waiting for service providers to agree on interoperability issues, the TogoWS project makes it happen for them.

B. SeaHawk

SeaHawk [40] is a front-end for BioMoby (more specifically, MOBY Services), written in Java, and developed at the University of Calgary (Canada). The people behind the project reviewed numerous other front-ends available for MOBY-S, categorised them, and evaluated what their respective strengths and weakness were, in order to build, as they see it, a better client. A major problem they identified with other front-ends is how they deal with actual data. MOBY-S employs several ontologies to deal with service specification, naming, data type hierarchies and relationships etcetera and all communication payload is wrapped in XML structures called Moby Envelopes. Many of the other front-ends either required the user to have extensive knowledge of the layout of those ontologies or were very limited in their expressiveness (in order to reduce complexity).

These findings inspired the idea of developing a front-end that was data-centric, the bioinformatician using the tool should focus on his or her actual data and not worry about implementation details. Since ontological terms do have to be specified in order to operate within MOBY-S, the SeaHawk client generates the required Moby Envelopes under the hood for the user. Data can be anything from service output, formatted HTML, rich-text files, text files in certain biologicial formats, or subsets and through its interface SeaHawk offers many ways to access the data.

While making it easier for novice programmers to work with the tool and focus on the data, the focus is also on using the system in a workflow manner and SeaHawk was recently enhanced to be able to generate Taverna workflows from its operations [42].

C. BioSpider

Our ongoing work to address the above problems is a tool that goes under the name of BioSpider. It's inspired by one of the core research ideas that we investigated in this paper: having a web service perform on-the-fly data integration on a few databases we knew had references to each other in one way or another.

In BioSpider, we do not focus on a particular technology like web services, but the emphasis is on the data itself. As we have discussed in this article, there are several autonomous providers of data (along with tools corresponding to that data) in the bioinformatics community. Even though there is an abundant number of data and service providers in the bioinformatics community, the actual data in databases themselves are not self-contained but full of references to other databases. These references represent important information for bioinformaticians performing their research, but the sheer intricacy and massiveness makes it very diffult for them to get a bird's eye view of how different data sets and entries are connected.

This is where BioSpider comes in. BioSpider is at its core a framework, a model for creating a single graph, representing data from disparate sources that have references to each other in some way. This graph is in effect the result of data integration operations performed between two connected sets of data.

The framework also comes with a rule-set for displaying this graph in a graphical user interface. It's through this user interface the user sees a unified view of the data, and he or she can follow links in the data to more sources, or apply actions on different data items (nodes) in the graph. These actions include visiting web pages, invoking web services or even running third-party tools. Figure 11 shows an example of the what the graph can look like for the user.

Fig. 11. Graph of connected data sources as visualized by BioSpider

In technical terms, BioSpider is written in Java. It consists of a set of classes that make up the core framework where connections between data items and actions that are available are specified. We have separated the actual framework from the data to allow the rule-set used by the framework to be extendible by users. One extension is the introduction of support for a completely new database, i.e., completely new functionality, or it can be an extension of the functionality of an already known data source. Another possible extension is additional actions that can be performed on data items or new references to other sources that can be exploited.

Our goal with the framework is to avoid hard-coded connections to any given data set and just provide a set of rules for describing data sets, actions that can be performed on items in the data and connections to other sources. The framework should be expressive to be able to capture all kinds of different data and actions but still be easy to extend.

We forsee two kinds of users. The first user is the normal user, using the framework as an ordinary desktop application to explore data in a unified way and perform actions on the data. The second user is a power user who will extend the rule set, making the tool even more powerful. These extensions should be easy to feed back to the community, or to the authors of the framework. Recently, we evaulated the extensibility of the framework and the results were encouraging.

In the future, we envision being able to in one way or another incorporate or utilize work done by other projects, like the BioCatalogue project, to further extend and improve our framework for the benefit of the user.

VIII. Conclusion

In this article we performed a literature and case study to examine the situation for bioinformaticians with a need for creating complex workflows using multiple service providers. They face severe interoperability problems because it can be very difficult to discover appropriate services and determine how they can be used in conjunction. When designing web services for the bioinformatics community we have identified several issues that need to be addressed to achieve a high discoverability and interoperability. The individual web service methods should be fundamentally simple so the semantics is easy to describe and the method should operate on data that is semantically well defined. This will allow the service to be registered in frameworks such as BioMoby and ^{my}Grid. Registration of services will assist users in discovering the service and decide how it can be used with other services.

References

[1] M. Åsberg and L. Strömbäck, "Interoperable and easy-to-use web services for the bioinformatics community - a case study," in *The Second International Conference on Advances in Databases, Knowledge, and Data Applications DBKDA 2010*, 2010.

[2] M. Y. Galperin, "The molecular biology database collection: 2008 update," *Nucleic Acids Research*, vol. 36, 2008.

[3] ——, "The molecular biology database collection: 2007 update," *Nucleic Acids Research*, vol. 35, 2007.

[4] L. Strömbäck, D. Hall, and P. Lambrix, "A review of standards for data exchange within systems biology," *Proteomics*, vol. 7, no. 6, pp. 857–867, 2007.

[5] L. Strömbäck, V. Jakoniene, H. Tan, and P. Lambrix, "Representing, storing and accessing molecular interaction data: a review of models and tools," *Briefings in Bioinformatics*, vol. 7, no. 4, pp. 331–338, 2006.

[6] P. W. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. A. Goble, and L. Stein, "Applying semantic web services to bioinformatics: Experiences gained, lessons learnt," *International Semantic Web Conference*, pp. 350–364, 2004.

[7] E. Germani, *Web Services Essentials*. O'Reilly, 2002.

[8] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori, "The KEGG resource for deciphering the genome," *Nucleic Acids Research*, vol. 32, 2004.

[9] M. Senger, P. Rice, and T. Oinn, "Soaplab - a unified sesame door to analysis tools," in *UK e-Science- All Hands Meeting 2003*, 2003.

[10] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[11] The Taverna Team, "Taverna - open source and domain independent workflow management system," Accessed january 16th 2011. [Online]. Available: http://www.taverna.org.uk/

[12] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services." *Nucleic Acids Research*, 2006.

[13] T. Oinn, M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency and Computation: Practice and Experience*, 2006.

[14] The VisTrails Team, "VisTrailsWiki," Accessed january 16th 2011. [Online]. Available: http://vistrails.org/

[15] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: Enabling interactive multiple-view visualizations," *In Proceedings of IEEE Visualization*, 2005.

[16] P. B. T. Nerrincx and J. A. M. Leunissen, "Evolution of web services in bioinformatics," *Briefings in Bioinformatics*, vol. 6, no. 2, pp. 178–188, 2005.

[17] The BioMoby Consortium, "Interoperability with Moby 1.0-it's better than sharing your tootbrush!" *Briefings in Bioinformatics*, vol. 9, no. 3, pp. 220–231, 2009.

[18] K. Wolstencroft, P. Alper, D. Hull, C. Wroe, P. Lord, R. Stevens, and C. Goble, "The myGrid ontology: bioinformatics service discovery," *International Journal of Bioinformatics Resesearch and Applications*, vol. 3, no. 3, pp. 303–325, 2007.

[19] J. Bhagat, F. Tanoh, E. Nzuobontane, T. Laurent, J. Orlowski, M. Roos, K. Wolstencroft, S. Aleksejevs, R. Stevens, S. Pettifer, R. Lopez, and C. A. Goble, "BioCatalogue: a universal catalogue of web services for the life sciences," *Nucleic Acids Research*, 2010.

[20] The BioCatalogue Project, "BioCatalogue.org - Home," Accessed january 16th 2011. [Online]. Available: http://www.biocatalogue.org

[21] The BioMoby Consortium, "BioMoby Semantic MOBY," Accessed january 16th 2011. [Online]. Available: http://biomoby.open-bio.org/index.php/semantic-moby/

[22] The Gene Ontology Consortium, "GO database abbreviations," Accessed january 16th 2011. [Online]. Available: http://geneontology.org/cgi-bin/xrefs.cgi

[23] M. Wilkinson, D. Gessler, A. Farmer, and S. L, "The BioMOBY project explores open-source, simple, extensible, protocols for enabling biological database interoperability," *Proceeding of the Virtual Conference on Genomic and Bioinformatics*, vol. 3, pp. 16–26, 2003.

[24] Object Management Group, "Life sciences analysis engine specification," Accessed january 16th 2011. [Online]. Available: http://www.omg.org/technology/documents/formal/lsae.htm

[25] E. Kawas, M. Senger, and M. D. Wilkinson, "BioMoby extensions to the taverna workflow management and enactment software," *BMC Bioinformatics*, 2006.

[26] Information Sciences Institute, "Pegasus:home," Accessed january 16th 2011. [Online]. Available: http://pegasus.isi.edu

[27] The Kepler Project, "The Kepler project - Kepler," Accessed january 16th 2011. [Online]. Available: http://kepler-project.org

[28] The Swift Project, "Swift," Accessed january 16th 2011. [Online]. Available: http://www.ci.uchicago.edu/swift

[29] L. Moreau, J. Freire, J. Futrelle, R. E. McGrath, J. Myers, and P. Paulson, "The open provenance model," 2008. [Online]. Available: http://eprints.ecs.soton.ac.uk/14979/1/opm.pdf

[30] T. Ellkvist, D. Koop, J. Freire, C. Silva, and L. Strömbäck, "Using mediation to achieve provenance interoperability," in *IEEE Workshop on Scientific Workflows*, 2009.

[31] EMBL-EBI, "BioModels database," Accessed january 16th 2011. [Online]. Available: http://www.ebi.ac.uk/biomodels-main/

[32] UniProt Consortium, "UniProtKB," Accessed january 16th 2011. [Online]. Available: http://www.uniprot.org/help/uniprotkb

[33] RCSB, "RCSB protein data bank," Accessed january 16th 2011. [Online]. Available: http://www.rcsb.org/pdb/home/home.do

[34] Oracle Corporation, "Oracle technology network for java developers," Accessed january 16th 2011. [Online]. Available: http://www.oracle.com/technetwork/java/index.html/

[35] The Eclipse Foundation, "Web tools platform (WTP) project," Accessed january 16th 2011. [Online]. Available: http://www.eclipse.org/webtools/

[36] B. J. Bornstein, S. M. Keating, A. Jouraku, and H. M., "Libsbml: An api library for sbml." *Bioinformatics*, 2008.

[37] D. Koop, C. E. Scheidegger, S. P. Callahan, J. Friere, and C. T. Silva, "Viscomplete: Automating suggestions for visualization pipelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1691–1698, 2008.

[38] T. Katayama, M. Nakao, and T. Takagi, "TogoWS: integrated SOAP and REST APIs for interoperable bioinformatic web services," *Nucleic Acids Research*, 2010.

[39] Database Center for Life Science, "TogoWS," Accessed january 16th 2011. [Online]. Available: http://togows.dbcls.jp/

[40] P. M. Gordon and C. W. Sensen, "Seahawk: moving beyond HTML in web-based bioinformatics analysis," *BMC Bioinformatics*, 2007.

[41] The Open Bioinformatics Foundation, "Open bioinformatics foundation," Accessed january 16th 2011. [Online]. Available: http://www.open-bio.org/

[42] P. M. Gordon, K. Barker, and C. W. Sensen, "Helping biologists effectively build workflows, without programming," in *Proceedings of the 7th International Conference on Data Integration in the Life Sciences - DILS 2010*, 2010.