Functional Units: Abstractions for Web Service Annotations

Paolo Missier, Katy Wolstencroft, Franck Tanoh, Peter Li, Sean Bechhofer, Khalid Belhajjame, Steve Pettifer, Carole Goble
School of Computer Science, University of Manchester
Oxford rd, Manchester, UK
Email: {firstname.lastname}@cs.manchester.ac.uk

***Abstract*—Computational and data-intensive science increasingly depends on a large Web Service infrastructure, as services that provide a broad array of functionality can be composed into workflows to address complex research questions. In this context, the goal of service registries is to offer accurate search and discovery functions to scientists. Their effectiveness, however, depends not only on the model chosen to annotate the services, but also on the level of abstraction chosen for the annotations. The work presented in this paper stems from the observation that current annotation models force users to think in terms of service interfaces, rather than of high-level functionality, thus reducing their effectiveness. To alleviate this problem,we introduce *Functional Units* (FU) as the elementary units of information used to describe a service. Using popular examples of services for the Life Sciences, we define FUs as configurations and compositions of underlying service operations, and show how functional-style service annotations can be easily realised using the OWL semantic Web language. Finally, we suggest techniques for automating the service annotations process, by analysing collections of workflows that use those services.**

I. Introduction

The popularity of Web Services for e-science applications is not without its problems, as scientists find themselves exploring a large space of available and potentially useful services, with only a limited understanding of the functions they offer. Part of the problem is that, despite a wealth of research over the past few years, service annotations still reflect a interface-oriented view, rather than a functional view of the service. This is a central problem for large registries of services, and for Biocatalogue in particular. Launched in 2009, the Biocatalogue registry[1] caters primarily to the bioinformatics and biomedical community. Building upon previous work on service registries and semantic annotation from the ^{my}Grid project[2] and the Embrace registry[3], Biocatalogue positions itself as a high-quality, annotation-rich clearinghouse for domain-specific service descriptions on a large scale (hundreds or thousands of services). Its primary goal is to provide descriptions of services that can be used both for accurate search and discovery, and to support the process of composing service invocations as part of larger workflows. The project is now at a critical juncture, where the choice of annotation model may determine its adoption by the Life Sciences community.

[1]http://www.biocatalogue.org
[2]http://www.mygrid.org.uk/
[3]http://www.embraceregistry.net/

A variety of conceptual models have been proposed over the years for service annotation. These include SAWSDL[4], a W3C recommendation containing syntactic conventions for augmenting a WSDL interface specification with semantic annotations; OWL-S [8][5], and WSMO[6], amongst others. While these are potentially viable annotation models, they are all based on the common implicit assumption that the two core tasks mentioned earlier are best served by annotating services at the level of their individual operations, and of the data types of the operations' messages.

The OWL-S model, for example, includes three perspectives on a service, namely its profile (what the service does), its model (how it works), and its grounding (how to access the service). Focusing on the profile, consider for example the `DDBJ` database[7]. Its `Blast` service can be annotated to specify that the operation `searchSimple` accepts a *biological sequence* as input, as follows:

```
<profile:serviceOperation>
    <operation:name> searchSimple</operation:name>
    <process:input id resource="#biological_sequence" />
</profile:serviceOperation>
```

In this example, the annotation makes a reference to the `biological_sequence` concepts in the ^{my}Grid ontology [15] (with namespace omitted).

The WSMO annotation model [10], [3], along with its WSMO-Lite counterpart [14], adds expressivity to annotations by accounting for *functional descriptions*. These are represented as capabilities, which define pre- and post-conditions that must hold before and after the invocation of a service's operation, or as functionality classications that define the service functionality using some classication ontology.

While these models abound in expressivity, their main shortcoming is that they apply rigidly either to the entire service, or to its individual operations, following the underlying WSDL-based structure of service description (a second shortcoming is that they do not apply at all to REST-style services, which lack a WSDL interface altogether. Recent proposals like SA-REST[13] address this but suffer from the same narrow perspective on annotations.) In this paper we

[4]http://www.w3.org/2002/ws/sawsdl/
[5]http://www.w3.org/Submission/OWL-S/
[6]http://www.w3.org/Submission/WSMO/
[7]http://www.ddbj.nig.ac.jp/

contend that, for the purpose of service discovery in the context of a registry like BioCatalogue, this level of abstraction is not always suitable, because the set of operations exposed by the service may not easily translate into a scientific task, or *unit of work*, that the users understand as being part of their application domain. Two main elements contribute to the problem, namely *polymorphic* operations, which are defined in a generic way to perform a number of possible functions, and operations that are used as part of some invocation *pattern*, along with other operations. An example of the former is the `eSearch` operation, which is part of the `eUtils` service provided by NCBI. While it would be difficult to associate a specific bioinformatic function to `eSearch` per se, the operation becomes meaningful when it is surrounded by a context in which some of its input parameters are configured, including, in this simple case, the target `database`. The same operation also provides an example of an invocation that is defined as part of a pattern, since `eSearch` is normally used in concert with the follow-on `eFetch` operation.

By annotating these services on individual operations (or for the service as a whole), a gap remains between the users' perspective of service operations as tasks with a well-defined function in the context of broader a scientific goal, and the service providers' technological view. We argue that this gap can be filled by choosing to annotate at a higher level of abstraction, taking into account both polymorphic operations, and operation patterns. We refer to these abstract descriptions as *Functional Units* (FU), and we show that commonly used bioinformatics services can be annotated at this higher level.

A. Paper organisation

The paper is structured into three part. Firstly, in Sec. II we analyse a number of service examples to derive a general definition of FU that is applicable to a broad class of public Web services. In this definition, we stick to the principle that the scope of a FU is limited to the set of operations that are part of the *same service*, i.e., we assume that FUs do not cross service boundaries. This is consistent with our primary goal to facilitate the search and discovery in Biocatalogue, where each service represents one atomic granule of information.

Secondly, in Sec. III we provide examples of concrete, functional, semantic service annotations, to show how the current OWL-based annotation style adopted in the past by the ^{my}Grid project, applies with minimal additional effort to functional units.

Thirdly, we address the problem of curating large collections of services. We start from the observation that FUs are "sensible combinations" of usage of configured service operations, and thus, they are possibly application- or even user-dependent, complicating their automated discovery. In many cases, however, a service is designed so that its operations "fit together", i.e., can be effortlessly composed according to some architectural pattern, in order to realise a domain-specific function. In this case, eliciting functional units amounts to recognising the original "intent" of the service as a cohesive unit, as well as the patterns that govern its proper usage. This observation suggests that a catalogue of tried-and-tested service compositions, typically workflows, can be an important asset when trying to automate, at least in part, service annotations at the FU level. In Sec. IV we propose preliminary ideas on how to partially automate the curation process, by leveraging prior work on mining collections of worklows [1].

The approach presented in this paper is being implemented as part of the ongoing development process for the Biocatalogue service registry.

B. Related work

To the best our knowledge, the idea that comes closest to the notion of functional units is described in the SemBOWSER project [11], where a high level functional description of services is advocated. Although similar in purpose, our approach goes one step further, by proposing to describe a service in terms of a whole collection of functional units, and furthermore, by accounting for different granularity, and hierarchical composition, of those FUs.

Another approach to abstracting the level of service description to better match the user's functional requirements can be found in Hashmi *et al.* [5]. Their method for facilitating the specification of bioinformatics workflows involves the scientists specifying the steps required for performing some analysis. The specification needs not mention either the computation units (web services) responsible for the execution of each of the steps, nor the order in which they should be executed. Instead, these elements are specified using semantic annotations that describe the capabilities of web services and their data dependencies.

Large-scale semantic service registries have been receiving a lot of attention in recent years. Among these, the *ServiceFinder* European project, based on the experience of the commercial SeekDA registry[8], stands out in that it aims at scaling up the size of their curated service registry, by automating the semantic annotation task [4].

Finally, there is an important distinction, in both scope and goals, between service registries that try to accommodate a large number of community-contributed services, like Biocatalogue, and those that collect a "closed" world of services, whose design and interface specification must adhere to project-specific rules. Ostensibly, the latter scenario simplifies the curation task, as the BioMoby project has shown over the years [2]. It would be unfair, however, to put those two on the same field, as services "in the wild" exhibit broader variations even when restricted to the Life Sciences domain.

II. From service operations to Functional Units

In this section we define Functional Units as service description abstractions that may exist in latent form within a service, and that can be elicited by service curators. Our commonsense definition of FUs is driven by a cross section of service examples from the BioCatalogue. These range over a variety of scenarios, from the simplest case of a single functional

[8]http://seekda.com

unit mapping to a single service operation, to functional units as patterns of composite services, and finally, as composition of lower-level FUs, in a hierarchical fashion. For clarity, the definitions in this section are organised along the four possible cases of a many-to-many relationship between service operations and FUs.

A. One operation, one FU

To begin, we consider services that are designed to expose a set of functions that relate directly to the user's domain, so that the operations on their interface reflect precisely those functions. In this simple case, FUs are exactly aligned with service operations. Example services for this class of FUs include SABIO-RK[9], KEGG[10], and BioMoby[11].

Most of the operations in the SABIO-RK service are of the form `get`X, i.e., they retrieve data from a database, where X incorporates a domain-specific term for the entity being retrieved, possibly along with the indication of an access method, for example: `searchEnzymesByECNumber`, `getCompoundID`, `getCompoundName`, etc. With this highly disciplined organisation of the service interface, the terms that appears in the operation's name can simply be used directly to describe the FUs, for instance: `searchEnzymesByECNumber`: *performs a task* (retrieving) on a resource (`Kegg_Gene_database`), has input parameters of some given type (`ECNumber`), and output parameters (a simple formalisation of these annotations using OWL are presented in the next section).

A similarly clean design can be observed in the KEGG service[12], where most of its over 70 operations perform searches over the KEGG family of databases concerning genes, proteins, ligands and pathways. Each operation defines one specific type of query over one of the databases, with its name often exhibiting an informational structure, for instance `getEnzymeByGene`, used to perform a simple database query to return enzyme identifiers encoded for by the genes specified. There is one input, which is the gene identifier, and one output, which is the enzyme identifier. In this case, the functional unit is data retrieval and the operation runs over the KEGG gene database.

Finally, in the case of BioMoby, the structure of a service operation is dictated by compliance to the BioMoby service model and to the annotation model prescribed at service registration time. The natural abstraction of BioMoby service descriptions using this model is at the functional unit level, typically with one operation performing one function.

B. One operation, multiple FUs

In the services cited above, each operation performs precisely one function. Commonly in Life Science services, however, operation are *polymorphic*, i.e., they can perform multiple functions depending upon the combination and values of their input parameters. For example, the `searchSimple` operation in the DDBJ database mentioned earlier can perform a variety of BLAST similarity searches of a biological sequence over selected biological sequence databases. This operation is polymorphic because its behaviour depends on the input parameters for (a) one of several BLAST programs to be executed, for instance `blastn` for a DNA query, `blastp` for a protein query, etc.; and (b) one of several available databases to align the input sequence against, for instance a particular DNA database rather than a Protein database. Not all of the combinations are valid, however. For instance `tblastn` expects a Protein sequence to be aligned against a DNA database: searching Uniprot in combination with a nucleotide sequence would result in an invalid query. This suggests that one can elicit all and only the Functional Units by enumerating all the *legal* combinations for the input configurations. This service, for example, offers five different FUs, corresponding to the five legal combinations of its inputs (see Table I). Finding legal combinations is the service curator's job. Whilst this is potentially a time-consuming task, it injects precious additional knowledge into BioCatalogue, potentially making it possible, for example, to automatically generate workflow processors for a given FU, by driving the configuration of the operations' invocation.

C. Multiple operations, one FU

Unlike the examples in the previous section, in some services the operations are designed to be orchestrated, and it is only when they are executed as part of a particular pattern that they produce a functional unit. In the case of many asynchronous services, for instance, one operation will submit the job to a queue and return a JobID, another will poll the execution with that JobID to check the status, and a third one will retrieve the results when they are ready. The functional unit of the service is only realised by the orchestration of all three operations in that particular order. The EBI Web Services work in this way. The InterproScan service[13], for example, has one functional unit composed from the interaction of three operations, `runInterProScan`, `CheckStatus` and `getResults`. This service compares a protein sequence against a collection of protein motif and domain databases to identify interesting domains within the sequence. The scientist can control which data is analysed, the format in which the data is returned, and the values of certain algorithmic parameters. The underlying mechanism for execution, however, should not be part of the functional unit definition.

An example of this pattern in action as part of a Taverna workflow involving InterProScan is shown in Fig. 1. Each processor in a Taverna workflow represents the invocation of a *single service operation* (but note that normally a workflow involves multiple services). Note that multiple concrete realisations of this common pattern are possible, depending on the available workflow language primitives. In particu-

[9]http://sabio.villa-bosch.de/webservicedoc.jsp.
[10]http://www.genome.jp/kegg/
[11]http://www.biomoby.org/
[12]http://www.genome.jp/kegg/
[13]http://www.ebi.ac.uk/interpro/

FU	input parameters		
	Blast Algorithm	**Sequence Database**	**Query Sequence**
proteinBlast	Blastp	Protein database (e.g Uniprot)	ProteinSequence
nucleotideBlast	Blastn	Nucleotide (e.g. DDBJ)	nucleotideSequence
proteinNucleotideBlast	tBlastn	Nucleotide (e.g. DDBJ)	proteinSequence
nucleotideProteinblast	Blastx	Protein (e.g. UniProt)	nucleotideSequence
nucleotideBlast with frame translation	tBlastx	Nucleotide (e.g. DDBJ)	nucleotideSequence

TABLE I
FUNCTIONAL UNITS AS LEGAL COMBINATIONS OF INPUT PARAMETERS TO AN OPERATION

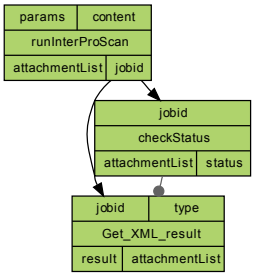


Fig. 1. Asynchronous operation pattern for the EBI InterPro service

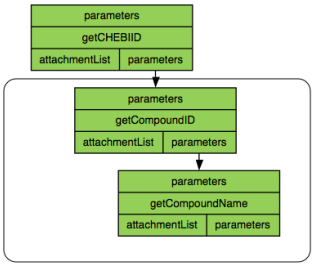


Fig. 2. Composite functional unit in the SABIO-RK service

lar, the figure shows a typical realisation done in Taverna, where the `checkStatus` processor encapsulates a while-loop construct, whereby the processor periodically polls the server using the native operation `checkStatus` until the job result becomes available. Thus, the three operations are used in concert to realise the complete pattern.

We observe a similar pattern in the `eUtils` service from the NCBI[14], which also requires asynchronous communication. A set of records, for example, is obtained by first specifying search criteria using the `eSearch` operations, and then retrieving the actual records using `eFetch`. The service is stateful, i.e., a user session is created as a result of the first invocation, which is available to subsequent operations. Note that these operations are highly polymorphic, as noted earlier.

The SoapLab tool[15] provides a final, but not less important example of operations that are designed for orchestration [12]. SoapLab generates service wrappers for "legacy" applications that are natively available only through a command-line interface from an OS shell. Examples of services generated using this tool are available from the EBI, for example[16]. As a consequence of the "cookie-cutter" approach to automated service generation, all SoapLab services expose the same set of operations, namely `getResults`, `getStatus`, `run`, `runAndWaitFor`. This is an example of a *server-oriented* approach to service design, whereby none of the service operations' domain semantics, and thus none of the latent functional units, are exposed by the service interface. While this service style provides basic primitives to interact with any service, it stands in contrast with the initial examples of this section, where a functional style of service design facilitates the curator's annotations task and the user's understanding.

[14]http://eutils.ncbi.nlm.nih.gov/
[15]http://soaplab.sourceforge.net/soaplab2/
[16]http://www.ebi.ac.uk/soaplab/

D. Composite FUs

The final, and more general case of functional units involves the hierarchical composition of existing FUs into new, higher-level FUs. Prime examples of these compositions are service-based workflows, often used to describe complex data pipelines as part of *in silico* experiments. We continue to use Taverna as an example of intuitive workflow model that accounts for hierarchical composition. As mentioned, at a basic level a workflow processor represents the invocation of a single service operation. When such operations are themselves FUs, as described in Sec. II-A, we expect the resulting workflow to perform a meaningful function to the user, as well, and thus potentially forming a new FU. Remember, however, that we limit the scope of FUs to compositions of operations *within a single service*, consistent with the service-level granularity of BioCatalogue. Even with this constraint, many services can be found that contain operations that form autonomous functional units, but can also be combined into composite FUs. Both KEGG and SABIO-RK provide good examples. The workflow fragment in Fig. 2 illustrates a composite functional unit from SABIO-RK. The first operation in the highlighted box retrieves a compound identifier from the ChEBI database, the second operation maps this identifier to a compound name. Either operation can be used independently, but they are designed and intended to be called as part of the same workflow, which forms a new FU. A hybrid case, for KEGG, of a pattern that contains an FU is shown in Fig. 3, where the `search_compounds_by_name` operation is an FU, but `bget`, a generic operation for retrieving data given their ID, only becomes a FU when it is properly configured.

These examples show higher-level FUs as mini-workflows

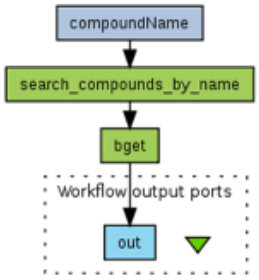


Fig. 3. KEGG composite functional unit

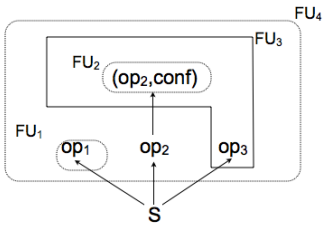


Fig. 5. Summary of functional unit definitions relative to service operations

where the participating operations (processors) are simply connected to each other through data dependency links. This tidiness, however, can only be achieved when the operations are indeed designed to fit together according to a natural pattern. In general, workflows designers often need to program additional bespoke adapters, encoded as local scripts, which perform necessary data transformations. While this would not be surprising when trying to connect operations from heterogeneous services, single-service workflows that require adapters seem indicative of a poor service interface design or to perform a complex query using data from a single database. The following example illustrates one of these complex composite FUs involving SABIO-RK. Fig. 4(a) shows a composite FU as an ideal sequence of processors. The purpose of this biochemical FU is to find chemical reactions that are associated with a given metabolite, and the kinetics associated with those reactions. This is an *ideal* workflow in the sense that it "skips over" the adapters that are required to make the data pipeline work in practice for identifying chemical reactions for a given set of metabolites using data within SABIO-RK. The relevant fragment of the actual workflow is shown in Fig. 4(b). The additional processors are scripts that perform local data manipulation (in this case, set intersection, parsing of lines in a text file).

When these composite functional units are properly annotated, the significant effort required for their design translates into high added value for third party users who discover them through BioCatalogue. In the next section we use some of these FUs to provide examples of such FU-style annotations. Fig. 5 shows a summary of the hierarchical definitions of functional units, from a single operation FU_1, to a configured single operation (FU_2), to a pattern potentially involving other FUs (FU_3), and finally, to composite FUs (FU_4).

III. Specifying Functional Units

Earlier in the paper, we made the point that the main distinction between operation-level and functional-level service annotations is one of abstraction, rather than of annotation model or language. For FU-level service annotations to be successful in practice, this change in abstraction level should not require curators to learn a new and unfamiliar annotation model. In particular, in the previous section we have identified the need to describe *combinations* of input parameters to a polymorphic operation, something that standard, attribute-level notations like SAWSDL are not designed for.

In this section we present functional annotations for our previous working examples, to show that such requirements can be met by adopting an OWL-based semantic annotation model. As we have noted earlier, this is by no means a new idea. Importantly, however, we show that functional-style annotations only require a slight extension to the annotation model used extensively in the ^{my}Grid project, and thus, it will keep Biocatalogue curators on familiar ground.

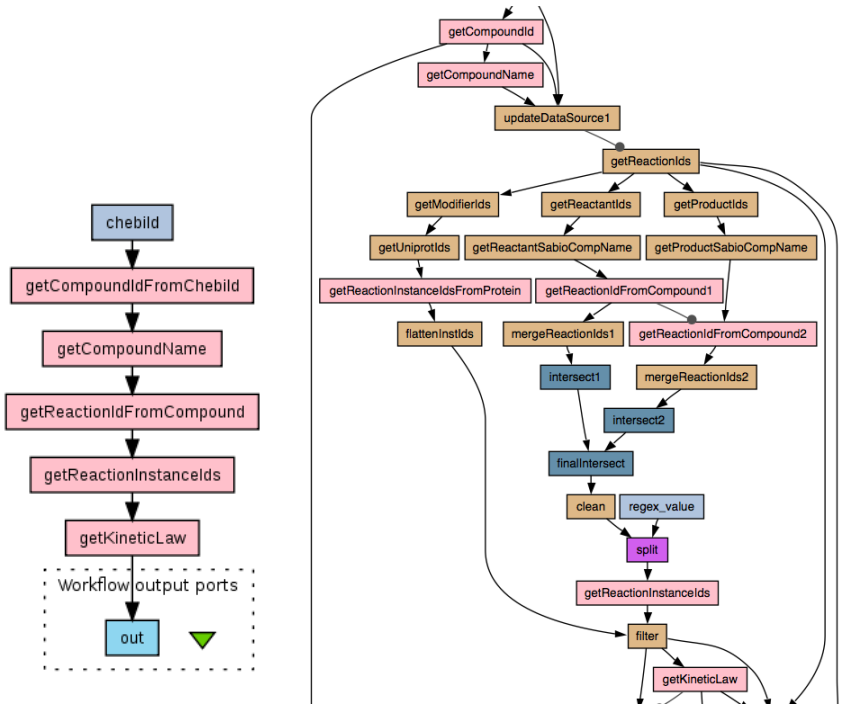
In ^{my}Grid, services were annotated using terms from the ^{my}Grid ontology [15] and descriptions stored and searched through the Feta semantic discovery component [7]. The ^{my}Grid ontology describes the bioinformatics research domain and the dimensions with which a service can be characterised from the perspective of the scientist. The ontology provides an annotation vocabulary, including descriptions of core bioinformatics data types and their relationships to one another. It also describes the physical and operational features of web services, such as, inputs and outputs and where the service is hosted. In ^{my}Grid, the ontology is currently used to describe individual properties of services and operations. Over 800 operations have been annotated in this way and their descriptions are available in the BioCatalogue. The ^{my}Grid ontology is expressed in OWL, which means it could be used to fully describe an individual service or operation as a collection of all of its properties instead of as individual properties. As a proof of concept, we used this ontology to describe some of the functional units identified previously.

A. One operation, one or multiple FUs

In the simplest case, FUs map exactly to a non-polymorphic operation. For example, the KEGG operation `getEnzymeByGene` mentioned in Sec. II-A performs a simple database query to return enzyme identifiers encoded for by the genes specified. There is one input, which is the gene identifier, and one output, which is the enzyme identifier. An human-readable rendering[17] of its formal OWL annotation is shown in Fig. 6. Intuitively, the annotation specifies the semantic types of the input and output parameters, and it specifies that the function is to retrieve data from the KEGG database. More precisely, an axiom like

`inputParameter` *some* `KEGG_gene_id`

[17]The rendering is provided by the Protegé OWL editing tool, available at http://protege.stanford.edu/.



(a) FU as an ideal Taverna workflow

(b) The actual workflow fragment

Fig. 4. Composite FU for SABIO-RK

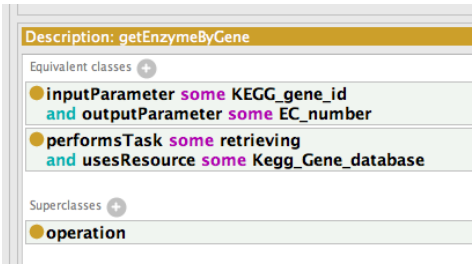


Fig. 6. OWL description of the `getEnzymeByGene` functional unit

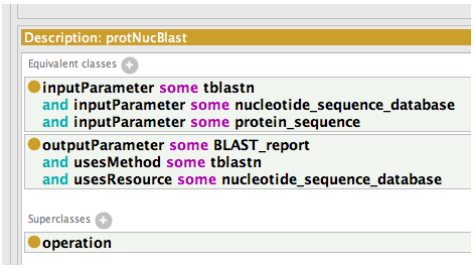


Fig. 7. OWL description of the `protNucBlast` functional unit

in the context of concept `getEnzymeByGene` defines the constraint: "at least one of the input parameters of `getEnzymeByGene` must be a `KEGG_gene_id`".[18] This is interpreted as a constraint, which allows input parameters to have any type, as long as `KEGG_gene_id` is one them. Note that the definition in Fig. 6 also asserts that `KEGG_Gene_database` must be one of the resources used by the functional unit, but others may be used as well.

When operations are polymorphic, as in the DDBJ example of Sec. II-B, the annotations need to describe legal combinations of values for the input parameters. An example of these combinations, for operation `SearchSimple`, was shown earlier in Table I. Fig. 7 shows the OWL specification for one of these functional units, namely `protNucBlast`. It generalises the previous cases by specifying a *set* of types for the input parameters, as well as a method (`blastn`). Note that now the specification defines the necessary types for the input parameters,[19] by asserting that the types of the input parameters form a superset of {`tblastn`, `protein_sequence`, `nucleotide_sequence_database`}.

B. Pattern-based FUs

We now present OWL definitions for the last two types of functional units, namely those that involve multiple operations arranged in a pattern. The first is for the `InterproScan` service, which is represented by a FU when three of its operations, `InterProScan`, `checkStatus` and `getResult`, are orchestrated to realise the asynchronous job submission

[18]Formally, this Protegé syntax renders the OWL axiom:

$\texttt{getEnzymeByGene} \sqsubseteq \exists\, \texttt{inputParameter}\,.\,\texttt{KEGG\_gene\_id}$

[19]However, it does not specify which type is associated to which input parameter.

pattern shown in Fig. 1. The corresponding FU definition (Fig. 6) requires the service to exhibit all three operations. Note that the inputs and outputs are now those of the FU, rather than those of each operation. Also, the new concept `wsdl-asynch` has been added to the ^{my}Grid ontology to characterise the type of pattern required by this FU. Note that this definition includes the operations and type of the pattern, but not its structure. We are planning to complement the OWL-based definitions with Taverna workflows that formalise the pattern.

Finally, Fig. 8 shows the definition for the composite FU of Fig. 3, involving the two units `compoundName` and `compoundRetrieval`, the latter being the FU resulting from the combination of `search_compound_by_name` and `bget`.

C. Further FU modelling options

The OWL definitions shown earlier in this section apply to classes of services, in that they formalise the conditions that define "what it means for a service to be a certain functional unit". In particular, note that the OWL axioms in these examples are placed in the *equivalent classes* box of the class definition in the figure, indicating that they define both necessary *and sufficient* conditions for a service to be a functional unit of type (i.e., an instance of class) `getEnzymeByGene`, or `ProtNucBlast`. To see what this means, suppose that an actual service S, that appears for example in BioCatalogue, has been annotated with assertions that include "S has inputParameter X", "X has type `Kegg_gene_id`", and so forth for all the other conditions that define the `getEnzymeByGene` FU. A standard OWL reasoner will be able to infer that S is of type `getEnzymeByGene`. In general, specifying functional units in terms of necessary and sufficient conditions (*constructed classes*, in OWL parlance) provides a powerful way to automatically classify actual services based on their specific annotations.

Given this potential for automated inference, it is important to understand the exact meaning of the FU class definitions. For example, class `protNucBlast` may include services that have input parameters of arbitrary type, provided that the parameter types specified in the definition also apply to some of the parameters. This is because the definition only includes existential quantifiers, i.e., "there must exist a parameter of type X". One could, however, further constrain the set of acceptable parameter types to be all *and only* those in the set above, by adding universal quantifiers ("if parameter X has a type, it shall be t_1 or t_2 or t_3"). This has the effect of "closing" the set of possible types that appear in the list of parameters.

Which constraints are appropriate is largely a modelling issue, often based upon the trade-off between accuracy of the annotations, and potential for automated classification. For example, one may not want to completely determine the set of acceptable input types, because some of the input parameters (e.g. the max number of returned hits in Blast) do not affect the definition of the service as a functional unit. In this case, one could decide to characterise the parameters as belonging to one of two classes (FU-critical vs all the others), and then apply the stronger, closed-world type constraints to only the FU-critical ones. We plan to evaluate these and similar options in practice, using an upcoming version of BioCatalogue that supports functional-style annotations.

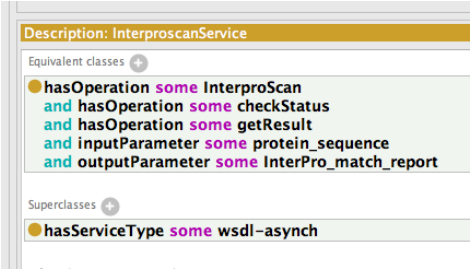


Fig. 8. OWL description of the `InterProScan` functional unit

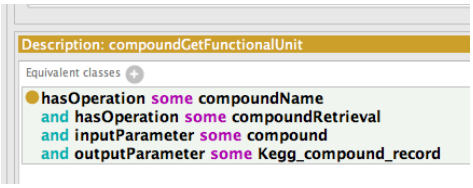


Fig. 9. OWL description of the `compoundGetFU` functional unit

IV. Eliciting Functional Units

Annotating a web service is a challenging task that requires deep knowledge of both the implementation of the web service and the domain ontology used for annotation. Asking the annotator to additionally be knowledgeable of the functional units that can be constructed out of the operations that compose the web service makes the annotation task even more challenging and prone to errors. Therefore, if the notion of functional units is to be adopted in practice, a means for assisting human annotators in identifying relevant functional units that can be obtained by aggregating the operations of a given web service is needed.

A naive approach to automating the identification of functional units would involve constructing all workflows that can be obtained by composing subsets of the service operations of the web service in question. Such an approach is, however, neither efficient nor effective. The number of workflows that can be obtained by composing the operations of a web service is large. Also, combining any subset of service operations does not necessarily guarantee obtaining a valid functional unit. Therefore, we need a means for identifying service operation combinations that are relevant to the domain in question, and that can be used as building block for constructing other applications, in particular workflows.

The idea is to exploit the specifications of a collection of tried-and-tested workflows for guiding the human annotator. To explain this idea, we will use the workflow example illustrated in Fig. 3. Given that the three operations that constitute this workflow are supplied by the same web service,

we can suggest it to the human annotator as a potential relevant functional unit that can augment the semantic description of the web service in question.

More specifically, the task of elicitation of functional units can be defined as follows. Given a repository of tried and tested workflows, the elicitation task consists in extracting sub-workflows, the component operations of which are supplied by the same web service. Note that a single workflow definition may lead to the identification of multiple functional units of the same and/or different web services. Therefore, the number of functional units that can be identified by simply parsing the workflow definitions in a repository such as myExperiment[20], can be potentially large.

Notice that we only consider tried-and-test workflows in the functional units elicitation task. This is because a workflow that suffers from errors may lead to erroneous functional units. A simple example of an error that may occur is that of mismatch between connected operations. If a workflow connects operations with incompatible parameters, then the functional unit that we extract form that workflow may inherit such an error, and its execution, therefore, can lead to execution errors.

It is worth mentioning that the elicitation of functional units as discussed in this section tackles the problem of identifying the participant operations, and the way they are to be combined to deliver the functional unit in question. In addition, the annotation task involves annotating the inputs and outputs of the workflow that incarnates the functional unit by relating them to concepts from the domain ontology used for annotation. This tasks can be performed in a manual manner, or in a semi-automatic fashion by using tools developed by the semantic web service community such as Meteor-S [9], Assam [6] and QuASAR [1]. Indeed, the workflow that realises a given functional unit can be seen as "black box' service operation, in which case the task of annotating the input and output parameters can be seen as that of annotating the inputs and outputs of a web service operation.

The above discussion underlined the fact that workflow repositories can be exploited for the automatic identification of functional units. We are in the process of implementing and exploring these ideas within the context of the BioCatalogue web service registry to assist human annotators identifying functional units by parsing the workflow specifications readily accessible from the myExperiment workflow repository.

V. Conclusions

In this paper we have proposed a functional style of service annotation that involves *Functional Units* as the information elements that describe a service's functionality at a level that service consumers are likely to understand. This is in contrast to interface-style annotations, which often fail to reveal the service's intent. We have defined FUs through a number of examples as combinations of operation configurations and compositions, and have shown how they can be expressed using OWL DL. Finally, we have presented initial ideas on exploiting collections of service-based workflows for automating the discovery of functional units.

Our approach is currently being implemented as part of the Biocatalogue service registry.

[20]http://www.myexperiment.org

Acknowledgements

We would like to thank all the members of the Biocatalogue team, and in particular Jiten Bhagat in Manchester, and Eric Nzuobontane at the EBI.

References

[1] Khalid Belhajjame, Suzanne M. Embury, Norman W. Paton, Robert Stevens, and Carole A. Goble. Automatic annotation of web services based on workflow definitions. *TWEB*, 2(2), 2008.

[2] M. DiBernardo, R. Pottinger, and M. Wilkinson. Semi-automatic web service composition for the life sciences using the biomoby semantic web framework. *Journal of Biomedical Informatics*, 41(5):837–847, 2008.

[3] C. Feier, D. Roman, and A. Polleres. Towards intelligent web services: The web service modeling ontology (WSMO). In *Proc. of the Int'al Conf on Intelligent Computing (ICIC)*, Hefei, China, August 2005.

[4] Adam Funk, Holger Lausen, Nathalie Steinmetz, and Kalina Bontcheva. D3.3 - automatic semantic annotation research report - version 2. Technical report, ServiceFinder Consortium, November 2009.

[5] Nada Hashmi, Sung Lee, and Michael P. Cummings. Abstracting workflows: unifying bioinformatics task conceptualization and specification through semantic web services. In *in: W3C Workshop on Semantic Web for Life Sciences*, 2004.

[6] A. Heß, E. Johnston, and N. Kushmerick. ASSAM: A tool for semi-automatically annotating semantic web services. In *Procs. ISWC*, 2004.

[7] Phillip W Lord, Pinar Alper, Chris Wroe, and Carole A Goble. Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *Procs. ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2005.

[8] D. Martin, M. Paolucci, and S. McIlraith. Bringing semantics to web services: The OWL-S approach. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, San Diego, CA, 2004.

[9] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA*, pages 553–562. ACM, 2004.

[10] D. Roman, J. de Bruijn, A. Mocan, H. Lausen, C. Bussler, and D. Fensel. WWW: WSMO, WSML, and WSMX in a nutshell. In *Proceedings of the first Asian Semantic Web Conference (ASWC 2006)*, Beijing, China, September 2006.

[11] S S Sahoo Sheth, A.P., Hunter, B., York, W.S., Christopher J O Baker Cheung, and Kei-Hoi. *SemBOWSER - Semantic Biological Web Services Registry*. Springer, 2007.

[12] M. Senger, P. Rice, A. Bleasby, and M. Uludag. Soaplab: Open source web services framework for bioinformatics programs. In *Procs. of the 10th Annual Bioinformatics Open Source Conference*, 2009.

[13] Amit P. Sheth, Karthik Gomadam, and Jon Lathem. Sa-rest: Semantically interoperable and easier-to-use services and mashups. *IEEE Internet Computing*, 11(6):91–94, 2007.

[14] Tomas Vitvar, Jacek Kopeck, Jana Viskova, and Dieter Fensel. WSMO-lite annotations for web services. In *Proceedings of 5th European Semantic Web Conference (ESWC)*, 2008.

[15] K Wolstencroft, P Alper, D Hull, C Wroe, P W Lord, R D Stevens, and C A Goble. The myGrid ontology: bioinformatics service discovery. *International Journal of Bioinformatics Resesearch and Applications*, 3:303–325, 2007.