Automatically Identifying Bounds on Semantic Annotations for Bioinformatics Web Service Input Parameters

Ravinder Singh, Sean Bechhofer, Khalid Belhajjame, and Suzanne M. Embury

School of Computer Science, University of Manchester
Oxford Road, Manchester, UK
{singhr,seanb,khalidb,sembury}@cs.man.ac.uk

Abstract. Semantic annotation of web services is achieved by associating parts of service interface descriptions to concepts defined in a semantic model, such as an ontology. Annotating web services with semantic metadata is currently a manual procedure that is conducted by human annotators. In practice, however, service annotation can be both time-consuming and error-prone to implement on real-world web services. We have investigated this problem in the bioinformatics domain, where certain characteristics shared by many web services make them a difficult breed to annotate with semantics. By taking these characteristics into consideration, we present a probing-based technique that identifies upper and lower semantic bounds for the annotation of web service input parameters. The resulting bounds can serve as a useful starting point for human annotators and could help reduce the required effort in the semantic annotation process.

Keywords: semantic web services, semantic annotation, bioinformatics.

1 Introduction

Semantic web services [1] are seen as the next step in the life cycle of web service technology. The primary goals driving semantic web service research are to automate the discovery, composition, monitoring and execution of web services. To achieve these goals, web services must first be annotated with semantic metadata. Semantic metadata for web services is provided alongside a semantic model, such as an ontology. Semantic annotations for a web service can then be created by mapping parts of a web service description file (hereafter referred to as WSDL file) to appropriate concepts in an ontology that represents the domain of the web service.

Semantic annotation of web services is a task that is primarily conducted by human annotators. Typically, annotators were not involved in the development of the web services they annotate. As a consequence, they face certain hindrances in the annotation process. First, to annotate a web service, a human annotator must be familiar with the functionality that the web service provides. Such information is obtained by reading any available documentation for the web service

F. Daniel and F.M. Facca (Eds.): ICWE 2010 Workshops, LNCS 6385, pp. 232–243, 2010.

to be annotated. In cases where no documentation exists, annotators can inspect the WSDL file for potential clues to the web service functionality. However, due to the high degree of automation that many web service development platforms provide, there is no guarantee of the quality of a WSDL file. This problem is particularly evident in the bioinformatics domain, where many web service elements are inappropriately named in the WSDL file. For example, an operation parameter may automatically be named "`In0`" rather than a more meaningful mnemonic such as "`protein_sequence`". Secondly, the annotator must also be familiar with the domain ontology to be confident that the best possible annotations are provided. In scenarios where the ontology is large, as is the case with some domain ontologies, it can become highly impractical for a human annotator to know each concept in an ontology. At the time of writing, for example, the ^{my}Grid bioinformatics domain ontology[1] that was used in our experiment contains a total of 475 concepts. Such an ontology is, however, relatively small when compared to others such as the Gene Ontology[2] that contains well over 27,000 concepts. Thus, a consequence of these hindrances is that ontology-based annotation becomes a time-consuming and error-prone task when conducted manually.

In our work, we have sought to address these issues in the annotation of bioinformatics web services. Using a probing-based technique, our method seeks to assist human annotators in the annotation of bioinformatics web services (or web services from other domains that possess similar characteristics). More specifically, the presented method identifies upper and lower bounds on the possible annotation (ontology concepts) that a web service input parameter can be annotated with. We argue that for every ontology concept an annotator examines that is not the correct annotation is wasted effort. Thus, we envisage that the output bounds from our process can be a step towards reducing the required effort to annotate bioinformatics web services.

2 Context

Annotating bioinformatics web services with semantic metadata would be a natural step towards achieving the goals set out for semantic web services in the bioinformatics domain. It has, however, been observed that bioinformatics web services have certain characteristics that can further hinder the process of their semantic annotation [2]. Since many services in the bioinformatics domain are developed on an as-needed basis, such web services typically have poor quality descriptions [3]. For example, many web services have been created by converting existing command-line programs using automated conversion software such as SoapLab[3]. Such software can output poor quality WSDL files in which operations, parameters and datatypes are given meaningless identifiers. Also, as a result of their *ad hoc* creation, such web services can be deployed with little or no

[1] http://www.mygrid.org.uk/ontology
[2] http://www.geneontology.org
[3] http://www.ebi.ac.uk/soaplab

associated documentation [2]. The lack of an agreed upon data model in bioinformatics also means that such web services usually have weakly typed web service operation parameters [4]. For example, many parameters for existing bioinformatics web service operations accept input parameters that are typed using the XML-schema datatype `xsd:string`. In many cases, such operations accept data that typically consists of a complex underlying structure that would be more suitably encoded in a complex XML-schema datatype. Parameters defined using complex data-types can potentially give better clues to the type of data the web service accepts and hence give an annotator a better chance of understanding the functionality of the web service. Collectively, these characteristics result in a manual semantic annotation process which can be both error-prone and time-consuming.

The time-consuming aspect arises because: (a) multiple sources of information may need to be searched for and acquired to understand the functionality of a web service and (b) the annotator must be familiar with the entire ontology, if this is not the case, browsing or navigating a large ontology for the correct concept can become a highly time-consuming task in itself.

Furthermore, the error-prone aspect arises because: (a) Sources of information about a web service's functionality may be scarce, which may lead to an incorrect understanding of the service's functionality and (b) the information in such sources may also be scarce, which may also similarly lead to an incorrect understanding of a services' functionality.

To acquire an understanding of a typical real-world semantic annotation process for web services in the bioinformatics domain, we interviewed an expert annotator on the BioCatalogue project [5]. The first step in the annotation process described by the annotator is concerned with understanding the functionality of the web service operation being annotated. Here, the annotator typically has to use multiple sources of information. One source of information is the annotator's own background knowledge of the domain. If such background knowledge does not exist or does not help to determine the functionality of the web service, the annotator can refer to the documentation of the web service. In cases where no documentation exists, the annotator can also inspect the WSDL file for any potential clues on the functionality of the web service. However, if the web service is described poorly, even this last step becomes a fruitless option. When such sources of information are unavailable or do not help in determining the functionality of the web service, the annotator can attempt to contact the web service provider to acquire information that would help in annotating the web service. Finally, if no contact details are available, the annotator can manually experiment by invoking the web service operation, in the hope that by observing the output of the operation some clues can be derived to the functionality of the web service. Once the functionality of a web service operation has been determined, the next step is to map the various parts of the operation to appropriate concepts in the domain ontology. If the annotator is not completely familiar with the concepts in the ontology, the process of locating the best concept for each annotation can also be a time-consuming task.

To avoid time costs and yet still produce accurate annotations, it is important that human annotators are assisted in the manual annotation process. Various tools and methods have been proposed. In [6], Heß *et al.* introduce the ASSAM web service annotation tool. ASSAM utilises a machine learning method to assist in the annotation for a set of similar web services. Using previously annotated web services as training data, ASSAM is able to suggest candidate annotations for new web services. The machine learning algorithm uses the elements in a WSDL file (operations and their parameters) as features for classification. In [7], Lerman *et al.* also present a machine learning approach to identify semantic labels for input and output parameters of web services. Both these machine learning approaches, however, are dependent on how well the parts of a web service are named in the WSDL file. In [8], Patil *et al.* present a schema matching algorithm to suggest annotations for web services. In this work, the XML-based web service description and the domain ontology are both treated as schemas to be matched to each other. Elements in the WSDL file, such as operation and parameter names, are mapped onto concepts in the ontology using string similarity matching techniques. In [9], Afzal *et al.* use a semi-automated text-mining approach to extract semantic descriptions of bioinformatics services. Using scientific corpora such as journals, they apply various natural language processing techniques to create semantic profiles of web services that can later help in the manual annotation effort. Finally, in [10], Belhajjame *et al.* show that semantic annotations can be inferred from partially annotated workflows. They show that it is possible to infer parameter level annotations based on the data flow connections between operations in the workflow. In scenarios where a non-annotated operation is connected to an annotated operation, constraints can be inferred on the annotations that the non-annotated operation can potentially have.

Unfortunately, approaches based on string-similarity matching and machine learning rely on WSDL files where the elements of a web service are meaningfully named. Thus, the use of such approaches produce ineffective results on web services that have the characteristics described earlier. Some method is therefore required to assist annotators of bioinformatics web services by taking the characteristics of such web services into explicit consideration.

3 Automatically Identifying Annotation Bounds

Our aim is to provide assistance in the annotation of web services that possess the characteristics outlined in section 2. The goal is to see if accurate candidate annotation sets for web service input parameters can be determined by probing operations with annotated input data. The experimented method outputs an upper and a lower bound concept, both of which belong to the ontology being used for the annotation (the domain ontology). Each ontology concept which falls within the upper and lower bound (inclusive) is a potential candidate annotation for the web service input parameter. To arrive at the upper and lower bound concepts, we use a probing technique that consists of three steps.

In the first step, a web service operation is probed with a pool of annotated instance data i.e., raw data values that are mapped to concepts from a

domain ontology (e.g. the value "AB000100" would be mapped to the concept `DDBJ_accession` from the ^{my}Grid ontology). Thus, the pool will consist of a set of `<instance_value, concept>` pairs. Given a web service operation that is to be annotated, each instance in the pool of annotated instances is used as a value for the input parameter to execute the web service operation. Once executed, an operation typically returns a result. Thus, the next task in the automated process is to use the returned result of the operation to determine if the operation naturally accepted or rejected the input annotated instance. Here, we define acceptance of an instance by a web service operation to mean that the operation was developed to expect such an instance as input (discussed further in section 3.2). In the second step, we use the acceptance results for each instance to label each concept in the ontology. The labels provide an abstraction over the number of instances accepted for a particular concept. In the final step, the labels are used to identify the upper and lower bound concepts that can be used for the annotation of the input parameter. To identify the upper and lower bounds, our process is dependent on the constructs provided by the Web Ontology Language (OWL); thus, the domain ontology that we use is hierarchically modeled in OWL.

In the subsequent sections, we describe each of the stages of the process in more detail. First, in section 3.1 we further discuss the notion of annotated instance data and how such data can be collected.

3.1 Annotated Instance Data

Annotated instances are used to probe a web service operation and thus are a key input to the process. Ideally, for each concept in the domain ontology being used for the service annotations, we would have a set of instances that represent the concept as broadly as possible. For example, for the `protein_sequence` concept in the ^{my}Grid ontology, we were able to obtain protein sequence instances in different formats such as FASTA and GenPept, as well as the Plain sequence format. As one of the characteristics of the web services in this domain is that the operation parameters are weakly typed (i.e., as plain strings even when the internal structuring of the data passed through the parameters is complex), we did not need to concern ourselves with storing any additional data type information for each instance.

We were able to gather such data from two sources. As proposed in the QuASAR project[4], one source of such instance data are provenance logs for workflows. Some existing workflow execution engines may keep execution traces (provenance logs). If such logs are kept from workflows that contain annotated web service operations, intermediate data values that pass through the parameters of such operations can be retrieved through these logs. A second method, which typically results in more manual labor, is to browse online databases for such instances. We were able to retrieve instances for concepts by performing manual deep web searches on bioinformatics databases. For example, instances

[4] http://img.cs.manchester.ac.uk/quasar

for concepts such as `DDBJ_accession` and `SWISS-PROT_accession` were easily obtained from online databases such as DDBJ[5] and UniProtKB[6], respectively.

3.2 Probing and Instance Classification

The first stage of the process is primarily concerned with categorising each annotated instance as either being accepted or rejected by the web service operation. For the purpose of this initial investigation, we have only focused on identifying annotations for operations with single input parameters and do not yet consider the problem of attempting to identify candidate annotations for operations with multiple input parameters.

An important part of this stage is determining whether a particular annotated instance was accepted or rejected by the web service operation. To distinguish accepted instances from rejected instances, we use the returned result of the web service operation. Determining whether an input to a web service operation is accepted or rejected, based solely on observing the operations output, is a difficult task to automate. It is difficult primarily because there is no widely adopted convention on how a web service operation handles normal termination compared with abnormal termination. To signify abnormal termination, some operation's return the `null` data value or an empty string value, whereas others may return a custom string error message such as "Error: Invalid input" or even throw an exception. Furthermore, we also came across operations with weak input validation. Such operations typically attempted to perform their computations even with invalid input values, and returned string message results like "No results found". In such cases, it is extremely difficult, even manually, to determine whether the input instance was valid but the operation produced no result or whether the input instance was invalid and hence there was no result. As our aim was to automate this process, we formulated a heuristics-based criteria for classifying each annotated instance's effect on a web service. Based on the effect of a web service operation, input annotated instances are assigned to one of two classes, namely, `accepted` or `rejected`. The criteria are outlined below:

Rejected: An annotated instance is considered rejected if the operation output is a null data value, an empty string value or a thrown exception.

Accepted: Conversely, an annotated instance is considered accepted if the operation output does not match the criteria for `Rejected`.

Using the above criteria, each annotated instance can then be associated with a class that indicates whether the operation accepted or rejected it. Hence, the output from this stage of the process is a collection of `<<instance, concept>, class>` pairs.

[5] http://www.ddbj.nig.ac.jp
[6] http://www.uniprot.org/help/uniprotkb

3.3 Concept Labeling

In the next stage of the process, the level of abstraction is raised from the instance level to the concept level. To accomplish this, each concept in the domain ontology is labeled depending on the number of accepted instances for the concept. The labels provide guidance on arriving at the upper and lower bound concepts (next step in the process). Below, we further discuss each label and the information that it provides.

BEST CASE: At any time, only one concept in the domain ontology would be labeled as `BEST CASE`. In scenarios where instances are only accepted for a single concept in the ontology, we have reason to speculate that the best case concept is a good candidate for annotating the input parameter. In such scenarios, it is possible that the web service performs strong input validation and thus only accepts instances belonging to the identified concept.

GENERALISE: A concept labeled as `GENERALISE` indicates that the parent concept (of the labeled concept) is a more promising candidate annotation. For a concept C to be labeled as `GENERALISE`, all of its instances (including the inferred instances from its descendants) must be accepted by the operation and there exists another concept in the ontology that is not a descendant of C, for which at least one instance is accepted.

SPECIALISE: A concept is labeled as `SPECIALISE` whenever a child concept (of the labeled concept) is a possible candidate annotation. For a concept C to be labeled as `SPECIALISE` a non-empty subset of the instances associated to it must be accepted.

3.4 Identifying Upper and Lower Bounds

In the final stage of the process, the labeled concepts from the previous stage are used to identify the upper and lower bound concepts in the ontology. We define an upper bound concept as the most general concept that can be considered for the annotation of the web service input parameter. Conversely, the lower bound concept is defined as the most specific concept that can be considered an annotation of the input parameter. Let LC be the set of labeled concepts from the ontology, *ubound* and *lbound* to be the upper and lower bound concepts, then the process of locating the upper and lower bound concepts can be expressed using the algorithm in listing 1.

As shown in the pseudo code, the upper and lower bounds are easily identified if a best case concept resulted from the probing (i.e., all the instances for a single concept and only those instances were accepted). In such a scenario, both the upper and lower bound will be the same concept (the best case concept).

If the best case scenario is not realised, then the process of locating the upper and lower bound is dependent on the concepts labeled as generalise and specialise, respectively. The upper bound concept is the first common ancestor

ofthe generalise concepts (GC). In practice, this is accomplished by constructing a new anonymous OWL concept that represents the union of each generalise concept. Once constructed, the super concept of the constructed concept is the upper bound concept (first common ancestor). Conversely, the lower bound concept is the first common descendant of the specialise concepts (SC). In practice, to find the lower bound concept, we construct a new OWL concept that represents the intersection of each specialise concept. Once constructed, the sub concept of the constructed concept gives the lower bound concept (first common descendant).

```
inputs: LC - outputs: ubound, lbound
1   foreach concept ∈ LC
2      if (labelOf(concept) = "BEST CASE") then
3         ubound := concept
4         lbound := concept
5         exit-foreach
6      else if (labelOf(concept) = "GENERALISE") then
7            GC := GC ∪ concept
8      else if (labelOf(concept) = "SPECIALISE") then
9            SC := SC ∪ concept
10  end-foreach
11  if (ubound = null and lbound = null) then
12     if (GC ≠ ∅)
13        ubound := getLowestCommonAncestor(GC)
14     if (SC ≠ ∅)
15        lbound := getGreatestCommonDescendent(SC)
16  end-if
```

Listing 1. Pseudo code for identifying the upper and lower bound

The candidate concepts for the annotation of the input parameter are those concepts which fall within the upper and lower bound concepts (inclusive) in the ontology. In the possible scenario where we do not have an upper bound (i.e., no concepts meet the generalise label criteria) but have a lower bound, then the upper bound is all the ancestor concepts of the lower bound. If we do not have a lower bound (i.e., no concepts meet the specialise label criteria) but have an upper bound, then the lower bound is all descendant concepts of the upper bound. Finally, in the case where we do not have an upper and a lower bound, then the upper bound is the most general defined concept in the ontology (i.e., the direct sub concept of `owl#Thing`) and the lower bound is all the descendant concepts of the upper bound – i.e., the entire ontology becomes the candidate set.

4 Results and Qualitative Analysis

To conduct our experiment we used a set of 19 semantically annotated bioinformatics web service operations. Each operation was annotated manually by an

expert annotator from the bioinformatics domain. The existing annotations served as a gold standard for us to measure the accuracy of the output bounds from the automated process. The input parameters for each operation are annotated using concepts from the ^{my}Grid bioinformatics domain ontology (consisting of 475 concepts). In total, we were able to collect 183 raw instances for a total of 84 concepts in the ontology.

4.1 Experiment and Results

In our experiment, we tested the effectiveness of the process by running each web service operation through the process using the aforementioned instance data. We hypothesise that when it is possible to automatically distinguish between normal and abnormal termination of operations, then accurate candidate annotation sets which are significantly smaller than the domain ontology can be identified. To test this hypothesis, we measure how accurate the resulting candidate concept sets are (i.e., do the sets contain the gold standard annotation?) and how much of the ontology we are able to rule out for the annotator (i.e., is the size of the candidate concept set significantly less than the size of the ontology?). A summary of the results is presented in Figure 1.

For each operation, the graph shows the number of candidate concepts identified for each operations input parameter, i.e., the concepts that appeared within the upper and lower bounds (inclusive). The candidate sets all contained the gold standard annotation concept, thus giving a recall of 100%. For 10 of the operations we were able to obtain candidate sets that contained significantly fewer concepts when compared to the size of the full ontology. In two of these cases, the best case scenario was realised, since the operations only accepted instances for a single concept. Upon analysis of the probing results for these two operations, we observed that these operations performed strong validation of input values

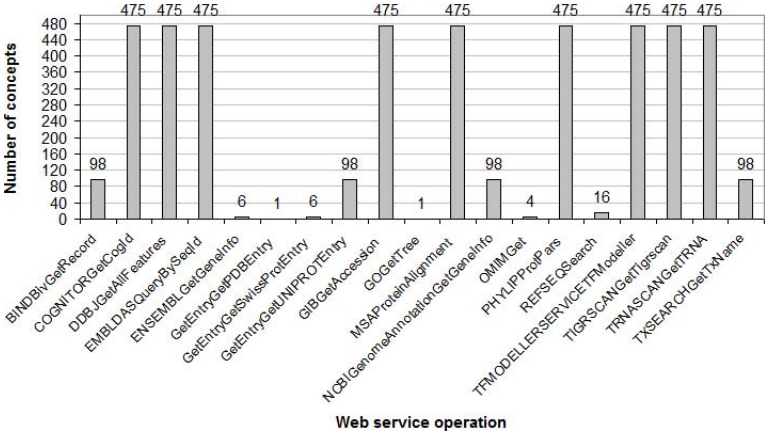

Fig. 1. Results from the experiment

and abided by the acceptance criteria that we used. For example, both operations returned an empty string response when terminating abnormally.

In the four cases that produced a candidate set size of 98, we found that although these operations performed strong input validation (i.e., accepted a low number of instances) the format of the instances of their gold standard annotation overlapped with the format from the instances of other "similar" concepts. For example, the gold standard annotation for the `BIND` web service operation `BivGetRecord` is the concept `BIND_id`. In total, for this concept, we had two raw instances "240947" and "185171". Although both of these instances were accepted, the `BivGetRecord` operation was unable to discriminate between similarly formatted instances for other "id" concepts. For example, the operation also accepted instances from other concepts such as `chEBI_term_id`, `KEGG_record_id` and `MaizeGDB_id`. The instances for these concepts have the same format as the instances for `BIND_id` (positive integers), thus the `BivGetRecord` operation incorrectly—but understandably—recognised such instances as valid BIND id's. As a result, each of the four operations that had a candidate concept set size of 98 obtained the same upper bound annotation of `bioinformatics_record_id` (first common ancestor of the id concepts) and did not produce a lower bound.

The results also show that for 9 of the operations the process was ineffective as the identified upper and lower bounds resulted in the entire ontology being returned (475 concepts). The reason for these large candidate concept set sizes was that these operations did not abide by the acceptance criteria that was used for classifying the instances as accepted or rejected. For some of these operations, there is evidence that they attempted to validate input values. However, their method of returning an abnormal response was typically using a proprietary string error message which we could not have anticipated when defining our generic acceptance criteria. In other cases, due to a lack of validation of the input parameters, the operations attempted to continue their computation regardless of whether the input value was valid or not. In these particular cases, the operations did not strictly terminate abnormally and hence returned a proprietary response that did not match the defined criteria for rejecting an instance.

In summary, the results show that the success of such a probing technique is highly dependent on being able to automatically distinguish between normal and abnormal termination of web service operations. When strict conventions on how to terminate an operation are adhered to, such a probing-based technique has the potential for becoming a cost effective solution for acquiring parameter-level semantic annotations.

4.2 Instance Pool Design

In addition to the experiment, an investigation was conducted to determine if characteristics of a "good pool" of annotated instances can be identified. The aim of this investigation was to attempt to identify guidelines for collecting annotated instances as they are an integral input to the process. To do this, we generated 200 instance pools using the original collection of 183 annotated instances. Using a total of 8 different pool sizes (20, 40, 60, ..., 160) which gave us a spread of

pool sizes, 25 random instance pools of each size were created. We reduced the number of web service operations used to 10, by disregarding the 9 operations which achieved a candidate concept set size equal to the size of the ontology. The reason behind this decision was that these particular operations did not abide to the acceptance criteria, and hence would only cause noise in the output of this investigation. Each generated instance pool was then used as input into the process for each web service operation. Using the output candidate concept sets, we selected the instance pools that resulted in good candidate concept sets and the pools that resulted in bad candidate concept sets. To determine the instance pools that performed well (the "good" pools), the following criteria was applied: (1) all candidate concept sets from a good instance pool must contain the gold standard annotation for each operation and (2) all candidate concept set sizes are less than half the size of the ontology. The bad instance pools were selected using the criteria: (1) more than half of the candidate concept set sizes are equal to the size of the ontology and (2) there exist some candidate concept sets that do not contain the gold standard annotation for an operation.

Using the above criteria, 18 good instance pools and 32 bad instance pools were found. In our analysis of these two sets of pools, we found that on average the instances in the good pools represented more concepts in the ontology (77) than the bad pools did (26). By representation of concepts, we mean the number of concepts for which instances existed in the pool. We also found that the good pools contained more instances than the bad pools did; for good pools the average number of instances per represented concept was approximately two, whereas for the bad pools the average number of instances per concept was approximately one. Perhaps unsurprisingly, these results provide some evidence to suggest that the more concepts represented and the more instances per concept within instance pools, are two potentially useful guidelines to follow when collecting such instance data.

5 Conclusions

In this paper, we have presented a technique to obtain candidate annotations for bioinformatics web service operations. We envisage that the technique could help to reduce the possibility of inaccurate annotations during the manual semantic annotation of such web services. The conducted experiment showed that for a sample of 19 bioinformatics web services, the process achieved a 100% recall with the gold standard annotation always being a member of the candidate annotation set. For the 10 web services that abided to the acceptance criteria, we were able to rule out more than half of the ontology through the output bounds. In some cases, even achieving candidate concept sets of less than 20 in size.

In cases where our acceptance criteria for distinguishing between the two did well, we were able to achieve candidate concept sets that were on average an approximate 90% reduction of the ontology. However, in cases where the acceptance criteria failed, the resulting output bounds were equal to the size of the ontology, and thus would have been no help to a human annotator. Thus,

for such a technique to be useful in practice, an effective means of being able to automatically distinguish between abnormal and normal termination of web service operations must be realised. One solution to this problem is to actively encourage the web service development community to follow strict guidelines when designing and developing web services. Furthermore, the success of this technique is also dependent on how a good a set of annotated instances can be obtained cost effectively. Our investigation provided some evidence to suggest characteristics of good collections of annotated instances, we envisage that these observed characteristics can be used as guidelines when collecting such data.

References

1. Mcilraith, S., Cao, T.S., Zeng, H.: Semantic web services. IEEE Intelligent Systems 16, 46–53 (2001)
2. Lord, P., Alper, P., Wroe, C., Goble, C.: Feta: A light-weight architecture for user oriented semantic service discovery. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 17–31. Springer, Heidelberg (2005)
3. Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. Nucleic Acids Res. 34(Web Server issue) (July 2006)
4. Hull, D., Stevens, R., Lord, P.: Describing web services for user-oriented retrieval (2005)
5. Belhajjame, K., Goble, C., Tanoh, F., Bhagat, J., Wolstencroft, K., Stevens, R., Nzuobontane, E., McWilliam, H., Laurent, T., Lopez, R.: Biocatalogue: A curated web service registry for the life science community. In: Microsoft eScience conference (2008)
6. Heß, A., Johnston, E., Kushmerick, N.: Assam: A tool for semi-automatically annotating semantic web services. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 320–334. Springer, Heidelberg (2004)
7. Lerman, K., Plangprasopchok, A., Knoblock, C.: Automatically labeling the inputs and outputs of web services. In: AAAI 2006: proceedings of the 21st national conference on Artificial intelligence, pp. 1363–1368. AAAI Press, Menlo Park (2006)
8. Patil, A.A., Oundhakar, S.A., Sheth, A.P., Verma, K.: Meteor-s web service annotation framework. In: WWW 2004: Proceedings of the 13th international conference on World Wide Web, pp. 553–562. ACM, New York (2004)
9. Afzal, H., Stevens, R., Nenadic, G.: Mining semantic descriptions of bioinformatics web resources from the literature. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 535–549. Springer, Heidelberg (2009)
10. Belhajjame, K., Embury, S.M., Paton, N.W., Stevens, R., Goble, C.: Automatic annotation of web services based on workflow definitions. ACM Trans. Web 2(2), 1–34 (2008)