Automatic Annotation of Web Services Based on Workflow Definitions

KHALID BELHAJJAME, SUZANNE M. EMBURY, NORMAN W. PATON, ROBERT STEVENS, and CAROLE A. GOBLE
University of Manchester

Semantic annotations of web services can support the effective and efficient discovery of services, and guide their composition into workflows. At present, however, the practical utility of such annotations is limited by the small number of service annotations available for general use. Manual annotation of services is a time consuming and thus expensive task, so some means are required by which services can be automatically (or semi-automatically) annotated. In this paper, we show how information can be inferred about the semantics of operation parameters based on their connections to other (annotated) operation parameters within tried-and-tested workflows. Because the data links in the workflows do not necessarily contain every possible connection of compatible parameters, we can infer only constraints on the semantics of parameters. We show that despite their imprecise nature these so-called *loose annotations* are still of value in supporting the manual annotation task, inspecting workflows and discovering services. We also show that derived annotations for already annotated parameters are useful. By comparing existing and newly derived annotations of operation parameters, we can support the detection of errors in existing annotations, the ontology used for annotation and in workflows. The derivation mechanism has been implemented, and its practical applicability for inferring new annotations has been established through an experimental evaluation. The usefulness of the derived annotations is also demonstrated.

Categories and Subject Descriptors: H.0 [**Information Systems**]: General

General Terms: Algorithms; Experimentation

Additional Key Words and Phrases: Semantic web services; semantic annotations; automatic annotation; workflows; ontologies

ACM Reference Format:

Belhajjame, K., Embury, S. M., Paton, N. W., Stevens, R., and Goble, C. A. 2008. Automatic annotation of Web services based on workflow definitions. ACM Trans. Web 2, 2, Article 11 (April 2008), 34 pages. DOI = 10.1145/1346237.1346239 http://doi.acm.org/10.1145/1346237.1346239

This article is an extended version of the paper presented in the International Semantic Web Conference, 2006 [Belhajjame et al. 2006].
The work presented in this article was funded by a grant from the BBSRC e-Science program.
Authors' address: School of Computer Science, University of Manchester, Oxford Road, Manchester, UK; e-mail: {khalidb, sembury,norm,rds,carole}@cs.man.ac.uk.

1. INTRODUCTION

Semantic annotations of Web services have several applications in the construction and management of service-oriented applications [McIlraith et al. 2001]. As well as assisting in the discovery of services relevant to a particular task [Maximilien and Singh 2004; Lord et al. 2004; Benatallah et al. 2005], such annotations can be used to support the user in composing workflows, both by suggesting operations that can meaningfully extend an incomplete workflow [Cardoso and Sheth 2003; Wroe et al. 2004; Sirin et al. 2004], and by highlighting inappropriate operation selections [Belhajjame et al. 2006; Medjahed et al. 2003]. As yet, however, few publicly accessible semantic annotations exist. Manual annotation is a time-consuming process that demands deep domain knowledge from individual annotators, as well as consistency of interpretation within annotation teams. Because of this, the rate at which existing services are annotated lags well behind the rate of development of new services [Wilkinson 2006; Goble et al. 2006].

Since resources for manual annotation are both scarce and expensive, some means by which annotations can be generated automatically are urgently required. Those have been recognized by a handful of researchers who have proposed mechanisms by which annotations can be inferred using machine learning algorithms [Heß and Kushmerick 2003; Heß et al. 2004] and schema matching techniques [Patil et al. 2004]. In this article, we explore the potential uses of an additional source of information about semantic annotations, namely repositories of trusted data-driven workflows. A workflow is a network of service operations connected together by data links describing how the outputs of some operations are to be fed into the inputs of others. If a workflow is known to generate sensible results, then it must be the case that the operation parameters connected by the workflow are compatible with one another (to some degree). In this case, if one side of a data link is annotated, we can use that information to derive annotation information for the parameter on the other side of the link.

Because the data links in the workflows do not necessarily contain every possible connection of compatible parameters, we unfortunately cannot derive exact annotations, but a looser form of annotation that specifies constraints on the semantics of operation parameters. Despite their imprecise nature, these *loose annotations* are useful. If a parameter is not annotated, its derived annotation can be used for supporting its manual annotation: an annotator can choose a concept from a (hopefully small) subset of the ontology indicated by the loose annotation, rather than from the full set of ontology concepts. Derived annotations can also be used in checking the compatibility of connected parameters in workflows, and in discovering service operations using the loose annotations derived for their inputs and outputs.

Deriving loose annotations for those operation parameters that are already annotated is also useful. Indeed, existing and newly derived annotations can be conflicting. These conflicts are manifestations of errors in existing annotations, the ontology used for annotation or the workflows used for deriving annotations. Therefore, by automatically detecting annotation conflicts, we can

provide support in detecting the presence of these errors. Note that we say "presence of errors," as further manual inspection may be required to detect the actual errors responsible for the identified annotation conflicts.

We previously proposed an annotation algorithm that implements the annotation inference method described above [Belhajjame et al. 2006]. In this article, we extend this annotation algorithm to cater for the automatic detection of conflicts between existing and derived annotations. We also analyze the errors responsible for annotation conflicts and specify the operations that can be used to correct them. Furthermore, we expand the preliminary evaluation reported in Belhajjame et al. [2006] to demonstrate the usefulness of derived annotations.

The article is organized as follows. We begin by formally defining the concept of a data-driven workflow (Section 2) and characterize parameter compatibility in workflows (Section 3). We then introduce the annotation derivation method (Section 4) and construct the annotation algorithm that implements it (Section 5). We analyze the conflicts detected by the annotation algorithm and specify the means by which they can be resolved. To demonstrate the applications that can benefit from derived annotations, we developed a tool that implements the annotation algorithm and uses the annotations the algorithm infers to support annotators in the manual annotation task, as described in Section 6. The tool also detects annotation conflicts and provides the operations that can be used to resolve them. In addition, it exploits derived annotations for inspecting mismatches in workflows. To further assess the proposed derivation method and gather experimental evidence that demonstrate its usefulness, we applied the annotation algorithm to a repository of bioinformatics workflows and a set of real (manual) service annotations (Section 7). The objective of this experimental evaluation is two-fold. Firstly, to show that the annotation algorithm is able to derive new annotations from a small set of existing annotations, and, secondly, to show that, despite their loose nature, derived annotations are of value in practice. To show this point, we measured the extent to which the use of derived annotations improves service discovery in terms of recall and precision. We analyze and compare work related to ours (in Section 8), and conclude by highlighting our main contributions (Section 9).

2. DATA-DRIVEN WORKFLOWS

The method for inferring annotations that we present in this article uses as inputs the data links that connect operation parameters within tried-and-tested workflows. Thus, for our purposes, we regard a workflow as a set of service operations connected together using data links. Formally, we define a data-driven workflow as a triple:

$$wf = \langle nameWf, OP, DL \rangle,$$

where $nameWf$ is a unique identifier for the workflow, OP is the set of operations from which the workflow is composed, and DL is the set of data links connecting the operations in OP.

Operations. An operation $op \in OP$ is a pair:

$$\langle nameOp, loc \rangle,$$

where *nameOP* is the unique identifier for the operation and *loc* is the URL of the web service that implements the operation.

Parameters. An operation parameter is a pair:

$$\langle op, p \rangle,$$

where *op* is an operation and *p* is a pair:

$$p = \langle nameP, type \rangle,$$

nameP is the parameter's identifier (unique within the operation *op*) and *type* is the parameter's data type. For Web services, parameters are commonly typed using the XML Schema type system[1], which supports both simple types (such as *xs:string* and *xs:int*) and complex types constructed from other simpler ones. Given an operation *op*, we use *inputs(op)* and *outputs(op)* to denote the input parameters and the output parameters of the operation *op*.

Data links. A data link describes a data flow between the output of one operation and the input of another. Let *IN* be the set of all input parameters of all operations present in the workflow *wf* and *OUT* the set of all output parameters, that is:

$$IN = \bigcup_{op \in wf.OP} inputs(op)\ OUT = \bigcup_{op \in wf.OP} outputs(op)$$

The set of data links connecting the operations in *wf* must then satisfy:

$$DL \subseteq OUT \times IN,$$

Notation. In the remainder of this article, we will use the following notation:

—*WF* is the set of trusted workflows given as input to the annotation process.
—*OPS* is the set of all operations used in *WF*, that is, $OPS = \{op \mid op \in OP \ \wedge \ \langle \_, OP, \_ \rangle \in WF\}$
—*DLS* is the set of all data link connections in *WF*, that is,
$DLS = \{\ dl \mid dl \in DL \ \wedge \ \langle \_, \_, DL \rangle \in WF\}$.
—*INS* is the set of all the inputs of the operations in *OPS*, i.e.,
$INS = \bigcup_{op \in OPS} inputs(op)$.
—*OUTS* is the set of all the outputs of the operations in *OPS*, that is,
$OUTS = \bigcup_{op \in OPS} outputs(op)$.

We also assume the existence of the function *connectedParams()* that given a parameter ⟨*op,p*⟩ returns the set of parameters that are connected to ⟨*op,p*⟩ by

[1]http://www.w3.org/TR/xmlschema-1

data links in *DLS*:

$$connectedParams(\langle op, p\rangle) = \{\langle op', p'\rangle \in INS \cup OUTS \mid (\langle op, p\rangle, \langle op', p'\rangle) \in DLS \text{ or } (\langle op', p'\rangle, \langle op, p\rangle) \in DLS\}.$$

3. PARAMETER COMPATIBILITY

If a workflow is well-formed, then we can expect that the operation parameters connected by the workflow data links are semantically compatible. Exactly what this means depends on the form of annotation used to characterize parameter semantics, although the basic principles should be the same in most cases. For the purposes of this article, we will consider a particular form of semantic annotation that was developed to facilitate the identification and correction of parameter mismatches in workflows [Belhajjame et al. 2006]. These semantic annotations are based on three distinct ontologies, each of which describes a different aspect of parameter semantics, and each of which is defined using the Web Ontology Language (OWL) [McGuinness and v. Harmelen 2004]. These are the *Domain Ontology*, the *Representation Ontology* and the *Extent Ontology*.

The Domain Ontology describes the concepts of interest in the application domain covered by the operation. This is the commonest form of semantic annotation for services, and several domain ontologies have been developed for different application domains. An example is the myGrid ontology that describes the domain of bioinformatics [Wroe et al. 2003]. Typical concepts in this ontology are *ProteinSequence* and *ProteinRecord*. The gene ontology[2] and the Galen medical ontology[3] are other examples of domain ontologies.

Although useful for service discovery, the Domain Ontology is not sufficient by itself to describe parameter compatibility within workflows, hence the need for the two additional ontologies. The first of these, the Representation Ontology, describes the particular representation formats expected by the parameter. In an ideal world, the data type of the parameter would give us all the information required about its internal structure. Unfortunately, however, it is common for the parameters of real Web services to be typed as simple strings, on the assumption that the operations themselves will parse and interpret their contents. This is partly a legacy issue (for services that wrap existing file-based applications [Senger et al. 2003]), but it is also partly caused by the weak type systems offered by many current workflow management systems, which do not encourage Web service authors to type operation parameters accurately. Because of this, to determine parameter compatibility, it is necessary to augment the information present in the WSDL data types with more detailed descriptions of the representation formats expected, using concepts from the Representation Ontology. An ontology of this kind for molecular biology representations has already been developed under the aegis of the myGrid project [Wroe et al. 2003], containing concepts such as *UniprotRecord*, which refers to a well known format for representing protein sequences, and *UniprotAC*, which refers to the accession number format dictated by the *Uniprot* database.

[2]http://www.geneontology.org/

[3]http://www.openclinical.org/prj_galen.html

The final annotation ontology that we use is the Extent Ontology, which contains concepts describing the scope of values that can be given to some parameter. Although in general it is not possible to accurately describe the extents of all parameters, in some cases this information is known. For example, the TrEMBL database[4] is known to contain information about a superset of the proteins recorded in the SwissProt database[5], and there are several species-specific gene databases that are known not to overlap. Information about the intended extents of parameters can help us to detect incompatibilities of scope in workflows that would otherwise appear to be well-formed. An example concept from the Extent Ontology is *UniprotDatastore*, which denotes the set of protein entries stored within the *Uniprot* database. It is perhaps worth noting that the use of the three ontologies to describe operation parameters is justified as it was experimentally shown that the data links of real workflows are subject to incompatibilities in terms of each of domain, representation and extent [Belhajjame et al. 2006].

In order to characterize parameter compatibility in terms of the above three ontologies, we assume the existence of the following functions for returning annotation details for a given parameter

$$\begin{aligned} domain &: INS \cup OUTS \rightarrow \theta_{domain} \\ represent &: INS \cup OUTS \rightarrow \mathcal{P}(\theta_{represent}) \\ extent &: INS \cup OUTS \rightarrow \theta_{extent} \end{aligned}$$

where θ_{domain} is the set of concepts in the Domain Ontology, $\theta_{represent}$ the set of concepts in the Representation Ontology, and θ_{extent} the set of concepts in the Extent Ontology. Note that an operation parameter can support more than one representation. For example, the operation *SimpleSearch*[6] supplied by the DNA Database of Japan[7] for aligning protein sequences accepts inputs that are formatted using either *Uniprot*[8] or *Fasta*[9]. These are two widely used bioinformatics sequence formats.

We also assume the existence of the function *coveredBy()* for comparing extents (since the standard set of OWL operators is not sufficient for reasoning with Extent Ontology concepts). Given two extents *e1* and *e2*, the expression *coveredBy(e1, e2)* has the value *true* if the space of values designated by *e1* is a subset of the space of values designated by *e2* and the value *false* otherwise.

Using annotations of the form described, we can automatically check a variety of forms of parameter compatibility that go beyond simple data type compatibility and that allow us, as we shall see later, to infer new annotations for operation parameters based on their connections to other annotated parameters within tried-and-tested workflows. We now present a classification of these compatibility types and define the criteria for verifying each one.

[4]http://www.ebi.ac.uk/trembl/
[5]http://www.ebi.ac.uk/swissprot
[6]http://xml.nig.ac.jp/wsdl/Blast.wsdl
[7]http://www.ddbj.nig.ac.jp/
[8]http://expasy.org/sprot/userman.html
[9]http://www.ncbi.nlm.nih.gov/blast/fasta.shtml

Domain Compatibility. This refers to compatibility in terms of semantic domain between connected output and input parameters. In order to be compatible, the domain of the output must be equivalent to or a subconcept of the domain of the subsequent input. Formally, the output parameter ⟨*op1,o*⟩ is domain compatible with the input parameter ⟨*op2, i*⟩ if and only if[10]:

$$domain(\langle op1,\ o\rangle) \sqsubseteq domain(\langle op2, i\rangle).$$

For example, consider a data link (⟨*op1,o*⟩,⟨*op2,i*⟩) such that $domain(\langle op1, o\rangle) = DNASequence$ and $domain(\langle op2, i\rangle) = ProteinSequence$. According to the molecular biology ontology mentioned earlier [Wroe et al. 2003], $DNASequence \not\sqsubseteq ProteinSequence$, therefore, ⟨*op1,o*⟩ and ⟨*op2,i*⟩ are incompatible in terms of semantic domain.

Representation Compatibility. Two operation parameters which belong to compatible semantic domains can be represented using different data formats. In order to be compatible the input parameter should support all the representations adopted by the output parameter. Specifically, the output ⟨*op1,o*⟩ is compatible with the input ⟨*op2,i*⟩ in terms of representation if and only if:

$$\begin{aligned}(domain(\langle op1, o\rangle) &\sqsubseteq domain(\langle op2, i\rangle)) \quad and\\ (represent(\langle op1, o\rangle) &\subseteq represent(\langle op2, i\rangle)).\end{aligned}$$

For example, suppose that $domain(\langle op1, o\rangle) = domain(\langle op2, i\rangle) = ProteinRecord$, $represent(\langle op1, o\rangle) = \{Uniprot\}$, and $represent(\langle op2, i\rangle) = \{Fasta\}$. The output and the input parameters are compatible in terms of semantic domain. However, they are incompatible in terms of representation: $Uniprot \notin represent(\langle op2, i\rangle)$.

Extent Compatibility. This refers to compatibility in terms of the space of possible values between two connected output and input parameters. In order to be compatible, the input's extent must cover the output's extent. Formally, the output ⟨*op1,o*⟩ is compatible with the input ⟨*op2,i*⟩ in terms of extent if and only if:

$$\begin{aligned}(domain(\langle op1, o\rangle) &\sqsubseteq domain(\langle op2, i\rangle)) \quad and\\ (represent(\langle op1, o\rangle) &\subseteq represent(\langle op2, i\rangle)) \quad and\\ coveredBy(extent(\langle op1, o\rangle), &extent(\langle op2, i\rangle)).\end{aligned}$$

For example, consider a data link (⟨*op1,o*⟩,⟨*op2,i*⟩) such that $domain(\langle op1, o\rangle) = domain(\langle op2, i\rangle) = ORF$[11] and $represent(\langle op1, o\rangle) = represent(\langle op2, i\rangle) = \{Fasta\}$. The two parameters are therefore compatible in terms of domain and representation. Suppose now that $extent(\langle op1, o\rangle) = FlyBase$ and $extent(\langle op2, i\rangle) = SGD$. *FlyBase*[12] is a database that stores information on the genetics and molecular biology of *Drosophila*. *SGD*[13] is a scientific

[10]The symbol ⊑ stands for subconcept of.
[11]*ORF* stands for open reading frame: a fragment of a DNA sequence potentially able to encode a protein.
[12]http://flybase.bio.indiana.edu/
[13]http://www.yeastgenome.org/

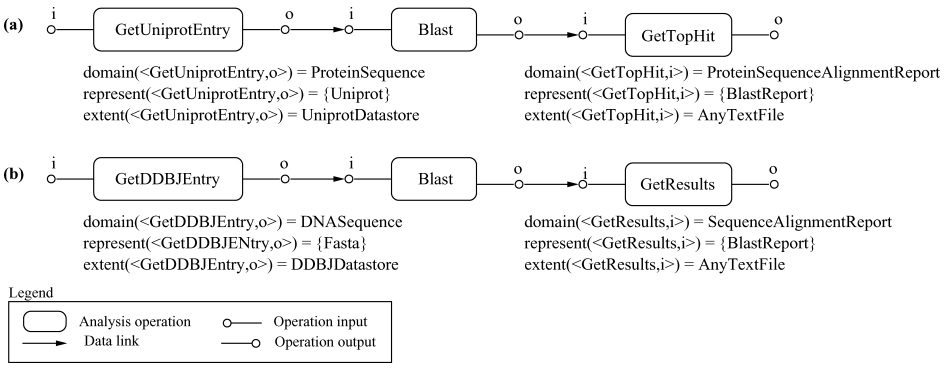


Fig. 1. Example workflows.

database of the molecular biology and genetics of the yeast *Saccharomyces cerevisiae*. The two databases are non-overlapping: none of the *ORFs* found in *FlyBase* are present in *SGD* and thus *coveredBy(FlyBase,SGD) = false*. Therefore, ⟨*op1,o*⟩ and ⟨*op2,i*⟩ are not compatible in terms of extent. Even though the two parameters are compatible in terms of domain and representation, the workflow will still not be able to produce valid results.

4. DERIVING PARAMETER ANNOTATIONS

Using the rules for parameter compatibility presented in the previous section, we can infer information about the semantics of linked parameters in workflows that the user believes to be error free. We will use a simple example to illustrate this idea. Consider the pair of workflows shown in Figure 1. Both these workflows are intended to perform simple similarity searches over biological sequences. The first finds the most similar protein to the one specified in the input parameter. To do this, it retrieves the specified protein entry from the *Uniprot* database, runs the *Blast* algorithm to find similar proteins, and then extracts the protein with the highest similarity score from the resulting *Blast* report. The second workflow finds similar sequences to a given DNA sequence. It retrieves the DNA sequence from the DDBJ database[14], searches for similar sequences using Blast and finally extracts the sequences of all matches from the *Blast* report.

Notice that the parameters of the *Blast* operation have not been annotated, while the parameters of the other operations have. However, since these are thoroughly tested workflows, their data links should all be compatible and we can therefore infer some information about the annotations that the *Blast* operation ought to have. For example, if we focus on just the domain annotations, we can see that the input of the *Blast* operation must be compatible with both *ProteinSequence* and *DNASequence*, and its output must be compatible with both *ProteinSequenceAlignmentReport* and *SequenceAlignmentReport*. In fact, by the rule of parameter domain compatibility, given in Section 3, we can infer

[14]http://www.ddbj.nig.ac.jp

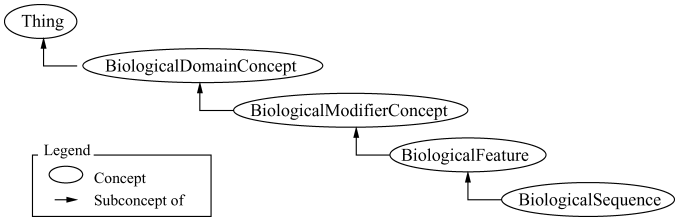


Fig. 2. Fragment of the domain ontology.

that[15]:

$$ProteinSequence \sqcup DNASequence \sqsubseteq domain(\langle Blast, i\rangle)$$

and

$$domain(\langle Blast, o\rangle) \sqsubseteq ProteinSequenceAlignmentReport \sqcap SequenceAlignmentReport.$$

Since, according to the domain ontology, *ProteinSequenceAlignmentReport* is a sub-concept of *SequenceAlignmentReport* we can further conclude that:

$$domain(\langle Blast,o\rangle) \sqsubseteq ProteinSequenceAlignmentReport$$

Note that, unfortunately, we cannot infer the exact annotation as we may not have been given a complete set of workflows (by which we mean a set of workflows that contains every possible connection of compatible parameters). All we can safely do is infer a lower bound on the annotation of the input parameters and an upper bound on the annotation of the output parameters. Thus, in the case of the *Blast* input parameter, we can use the derived lower bound just given to indicate the fragment of the ontology that must contain its true domain annotation (shown in Figure 2). In this case, all the super-concepts of the union of *ProteinSequence* and *DNASequence*[16]. As far as the output of *Blast* operation is concerned, there exists only one concept in the domain ontology that satisfies the derived upper bound condition: *ProteinSequenceAlignmentReport*.

We call these lower and upper bounds *loose annotations*, to distinguish them from the more usual (*tight*) form of annotation in which the exact concept corresponding to the semantics of the parameter is given by an annotator. All manually asserted annotations at present are tight annotations (though in the future annotators may prefer to assert loose annotations for difficult cases where they are unsure of the correct semantics).

[15]In the rest of the article, we use the symbol $\cup$ to denote the union set operator and the symbol $\sqcup$ to denote the operator for constructing the union of concepts in an ontology. Similarly, we use the symbol $\cap$ to denote the intersection set operator and the symbol $\sqcap$ to denote the operator for constructing the intersection of concepts in an ontology.

[16]The ontology fragment shown in Figure 2 does not contain the lower bound concept $ProteinSequence \sqcup DNASequence$ since it is not a (named) concept within the ontology. However, since the OWL language allows the formation of new concepts using, among others, the union and intersection operators, the true annotation may in fact be the lower bound itself (i.e. $ProteinSequence \sqcup DNASequence$). Other, less expressive, ontology languages such as RDFS, do not allow this possibility.

Based on this reasoning, we can derive a method for inferring loose annotations for operation parameters, given a set of tested workflows *WF* and a set of (tight) annotations for some subset of the operations that appear in *WF*. Since the compatibility relationship between input and output parameters is not symmetrical, we must use a different method for deriving input parameter semantics from that used for deriving output semantics.

4.1 Derivation of Input Parameter Annotations

Given an input parameter of some operation, we can compute three forms of loose annotation, based on the compatibility rules for each of the three annotation ontologies described previously as follows.

—*getInputDomain: INS* $\rightarrow \theta_{domain}$. This function computes a loose domain annotation, by locating a concept that is equivalent to or a subconcept of the semantic domain of the given input parameter. It first finds all operation outputs that are connected to the given parameter in *WF*. It then retrieves the domain annotations for these outputs and returns the concept obtained by their union. Formally,

$$getInputDomain(\langle op,i\rangle) = \bigsqcup_{(\langle op_x,o_x\rangle,\langle op,i\rangle) \in DLS} domain(\langle op_x, o_x\rangle).$$

In our example, the domain of *Blast* input must be a super concept of $ProteinSequence \sqcup DNASequence$.

—*getInputRepresentations: OPS* $\times$ *INS* $\rightarrow \mathcal{P}(\theta_{represent})$. This function computes the set of representations that should be supported by a given input parameter. It first finds the operation outputs that are connected to the given input. It then returns the set of representations obtained by unioning the sets of representations that are supported by such parameters. Formally, use different.

$$getInputRepresentations(\langle op,i\rangle) = \bigcup_{(\langle op_x,o_x\rangle,\langle op,i\rangle) \in DLS} represent(\langle op_x, o_x\rangle)$$

In Figure 1, the representation annotation that is inferred for the *Blast* input parameter is {*Uniprot, Fasta*}.

—*getInputExtent: INS* $\rightarrow \theta_{extent}$. This function computes a loose extent annotation, by constructing a concept that designates an extent that is covered by the extent of the given operation input. It first finds all output parameters that are connected to the input by workflows in *WF*. It then retrieves their extent annotations and returns the concept obtained by their union. Formally:

$$getInputExtent(\langle op,i\rangle) = \bigsqcup_{(\langle op_x,o_x\rangle,\langle op,i\rangle)) \in DLS} extent(\langle op_x, o_x\rangle)$$

In our example, the extent of the *Blast* input parameter is an extent which covers the extent designated by the union of *UniprotDatastore* and *DDBJDatastore*.

4.2 Derivation of Output Parameter Annotations

Derivation of annotations for output parameters follows much the same pattern as for input parameters. However, parameter compatibility rules require us to infer upper bounds on the semantics of output parameters rather than lower bounds as is the case for input parameters.

—*getOutputDomain: OUTS* $\rightarrow \theta_{domain}$. This function computes a loose domain annotation for the given output parameter by locating a concept that is equivalent to or a super concept of the semantic domain of the parameter. It first finds all input parameters that are connected to it in the workflows in *WF*. It then retrieves the domain annotations of these inputs and returns the concept obtained by their intersection. Formally,

$$getOutputDomain(\langle op,o\rangle) = \bigsqcap_{(\langle op,o\rangle,\, \langle op_x, i_x\rangle) \,\in\, DLS} domain(\langle op_x, i_x\rangle)$$

In our example, the output parameter of the *Blast* operation must be a subconcept of *ProteinSequenceAlignmentReport*.

—*getOutputRepresentations: OUTS* $\rightarrow \mathcal{P}(\theta_{represent})$. According to the representation compatibility rule (Section 3), the set of representations of an output parameter should be a subset of the set of representations supported by its connected input parameter. Therefore, when an output $\langle op, o\rangle$ is connected to more than one input, its set of representations should be a subset of the set obtained by the intersection of the sets of representations supported by its connected inputs. The function *getOutputRepresentations*$(\langle op,o\rangle)$ computes this intersection set. Formally,

$$getOutputRepresentations(\langle op,o\rangle) = \bigsqcap_{(\langle op,o\rangle,\, \langle op_x, i_x\rangle) \,\in\, DLS} represent(\langle op_x, i_x\rangle).$$

In our example, the annotation inferred for the *Blast* operation output parameter is {*BlastReport*}.

—*getOutputExtent: OUTS* $\rightarrow \theta_{extent}$ This function computes a loose extent annotation by locating an extent that covers the extent of the output parameter. It first finds all input parameters that are connected to the given output and retrieves the concepts representing their respective extents. It then returns the concept obtained by the intersection of these concepts. Formally,

$$getOutputExtent(\langle op, o\rangle) = \bigsqcap_{(\langle op,\, o\rangle,\, \langle op_x, i_x\rangle) \,\in\, DLS} extent(\langle op_x, i_x\rangle).$$

In our example, that the extent of the *Blast* operation output must be contained within the *AnyTextFile* extent.

In addition to these functions, we assume the existence of the subroutines *assertLooseDomain*, *assertLooseRepresentation*, and *assertLooseExtent*, the signatures of which are presented below, for asserting derived loose domains, representations, and extents of an operation parameter, respectively. They return true if the annotation has been successfully asserted (that is, entered to the annotation repository), and false, otherwise.

```
Algorithm DeriveAnnotations
inputs OPS
outputs OPS
begin
1     for each op ∈ OPS do
2         for each ⟨op, i⟩ ∈ inputs(op) do
3             c_domain = getInputDomain(⟨op, i⟩)
4             c_represent = getInputRepresentations(⟨op, i⟩)
5             c_extent = getInputExtent(⟨op, i⟩)
6             ActOnDerivedAnnotations(⟨op, i⟩, c_domain, c_represent, c_extent)
7         for each ⟨op, o⟩ ∈ outputs(op) do
8             c_domain = getOutputDomain(⟨op, o⟩)
9             c_represent = getOutputRepresentations(⟨op, o⟩)
10            c_extent = getOutputExtent(⟨op, o⟩)
11            ActOnDerivedAnnotations(⟨op, o⟩, c_domain, c_represent, c_extent)
end
```

Fig. 3. Annotation algorithm.

$$\textit{assertLooseDomain: (INS} \cup \textit{OUTS)} \times \theta_{domain} \rightarrow \textit{Boolean}$$
$$\textit{assertLooseRepresentation: (INS} \cup \textit{OUTS)} \times \mathcal{P}(\theta_{represent}) \rightarrow \textit{Boolean}$$
$$\textit{assertLooseExtent: (INS} \cup \textit{OUTS)} \times \theta_{extent} \rightarrow \textit{Boolean}$$

To avoid confusion between tight and loose annotations, the functions *domain*(⟨*op,p*⟩), *represent*(⟨*op,p*⟩) and *extent*(⟨*op,p*⟩), presented in Section 3, are used to retrieve only the asserted tight annotation of the parameter ⟨*op,p*⟩. As such, they return a null value when ⟨*op*, *p*⟩ does not have an asserted tight annotation, even when it has an asserted loose annotation. Clarifying this point is important for understanding the annotation algorithm presented in the next section. Similar functions to those used for retrieving asserted tight annotations can be defined for retrieving asserted loose annotations. However, we do not define these functions as we will not make use of them in this article. For the sake of simplicity, in the rest of the article, we use the term *asserted annotation* to refer to the asserted tight annotation of a parameter.

5. ANNOTATION ALGORITHM

Using the functions for deriving annotations for individual parameters presented in the previous section, we can construct an algorithm (shown in Figure 3) that derives all annotations automatically from a set of tested workflows and an incomplete repository of semantic annotations. The algorithm iterates over the parameters present in the workflows, deriving new loose annotations for each of them, using the functions given in the previous section. The resulting annotations are then examined by the subroutine presented in Figure 4. If there is no existing asserted annotation for a parameter, then the derived annotation is asserted and the subroutine returns the value true. If an asserted annotation is already present, then this is compared with the derived annotation to check for any conflicts. If the two are compatible, then no further action need be taken and the subroutine returns the value true. If not, then the conflict is flagged to the user and the subroutine returns the value false. A conflict is detected in the following cases:

```
Algorithm ActOnDerivedAnnotations
inputs ⟨op, p⟩ ∈ INS ∪ OUTS,
        c_domain ∈ θ_domain, c_represent ⊆ θ_represent, c_extent ∈ θ_extent
outputs isAsserted ∈ Boolean
begin
1       isAsserted = true
2       if (c_domain ≠ null) then
3           if (domain(⟨op, p⟩) = null) then
4               assertLooseDomain(⟨op, p⟩, c_domain)
5           else
6               if (⟨op, p⟩ ∈ INS ∧ c_domain ⋢ domain(⟨op, p⟩)) ∨
7                  (⟨op, p⟩ ∈ OUTS ∧ domain(⟨op, p⟩) ⋢ c_domain) then
8                   display(conflictingParams(⟨op, p⟩, "domain"))
9                   isAsserted = false
10      if (c_represent ≠ null) then
11          if (represent(⟨op, p⟩) = null) then
12              assertLooseRepresentation(⟨op, p⟩, c_represent)
13          else
14              if (⟨op, p⟩ ∈ INS ∧ c_represent ⋢ represent(⟨op, p⟩)) ∨
15                 (⟨op, p⟩ ∈ OUTS ∧ represent(⟨op, p⟩) ⋢ c_represent) then
16                  display(conflictingParams(⟨op, p⟩, "representation"))
17                  isAsserted = false
18      if (c_extent ≠ null) then
19          if (extent(⟨op, p⟩) = null) then
20              assertLooseExtent(⟨op, p⟩, c_extent)
21          else
22              if (⟨op, p⟩ ∈ INS ∧ !coveredBy(c_extent, extent(⟨op, p⟩))) ∨
23                 (⟨op, p⟩ ∈ OUTS ∧!coveredBy(extent(⟨op, p⟩), c_extent)) then
24                  display(conflictingParams(⟨op, p⟩, "extent"))
25                  isAsserted = false
26      return isAsserted
end
```

Fig. 4. Subroutine that examines and asserts derived annotations.

—*Domain conflict*. An input suffers from a domain conflict if its asserted domain annotation is not a super concept of its derived domain annotation (line 6, Figure 4). An output suffers from a domain conflict if its asserted domain annotation is not a subconcept of its inferred domain annotation (line 7, Figure 4). Consider, for example, that the asserted domain annotation of the input ⟨*op,i*⟩ specifies that it is a protein sequence, $domain(\langle op,i\rangle) = ProteinSequence$, and that its derived domain annotation indicates that it should be a super-concept of *Sequence*. Since *ProteinSequence* is not a superconcept of *Sequence*, according to the Domain Ontology, we conclude that $\langle op, i\rangle$ suffers from a domain annotation conflict.

—*Representation conflict*. An input parameter suffers from a representation conflict if its asserted representation annotation is not a superset of its derived representation annotation (line 14, Figure 4). Conversely, an output parameter suffers from a representation conflict if its asserted representation annotation is not a subset of its derived representation annotation (line 15, Figure 4). Consider, for example, that the asserted representation annotation of the output ⟨*op,o*⟩ specifies that its instances are formatted according

to either *PIR*[17] or *Uniprot* representations, $represent(\langle op,o\rangle) = \{PIR, Uniprot\}$, and that its derived representation annotation indicates that its instances are formatted according to *BioPax*[18] representation. The asserted and derived representation sets are not overlapping, and as such, they are conflicting. Moreover, they are used for representing data instances that belong to incompatible semantic domains. *Uniprot* and *PIR* are used for representing protein entries whereas *BioPax* is a data exchange format for biological pathway data.

—*Extent conflict*. An input suffers from an extent conflict if its asserted extent does not cover its derived lower extent (line 22, Figure 4). An output suffers from an extent conflict if its asserted extent is not covered by its inferred upper extent (line 23, Figure 4). Consider, for example, that the asserted extent annotation of the output $\langle op, o\rangle$ specifies that its instances belong to the *Uniprot* protein database, $extent(\langle op,o\rangle) = Uniprot$, and that its derived domain annotation indicates that its instances should belong to the *Swissprot* database. Since *Swissprot* does not contain all the protein sequences in *Uniprot*, $coveredBy(Uniprot, Swissprot) = false$, we conclude that $\langle op, o\rangle$ suffers from an extent annotation conflict.

When a conflict is identified for a parameter, the set of parameters having conflicting annotations are displayed to the user for inspection *(lines 8,16,24)*. These parameters are retrieved using the function *conflictingParams()* with the following signature:

$$conflictingParams: (INS \cup OUTS) \times ANT \rightarrow \mathcal{P}(INS \cup OUTS).$$

where *ANT* contains possible annotation types: *ANT* = {*"domain", "representation", "extent"*}. Given a parameter $\langle op, p\rangle$ together with an annotation type *ant*, $conflictingParams(\langle op,p\rangle, ant)$ returns the set of parameters that are connected to $\langle op, p\rangle$ and whose asserted annotations of type *ant* are conflicting with that of $\langle op, p\rangle$.

5.1 Sources of Annotation Conflicts

Conflicts between asserted and derived annotations are manifestations of errors in existing annotations, the ontology used for annotation or the workflows used for deriving annotations. By automatically detecting annotation conflicts in our algorithm, we can therefore provide support in detecting the presence of these errors. Specifically, an annotation conflict may help detect the presence of the following errors.

—*Incorrect annotations*. The manually asserted annotations of the parameter in question and/or some of its connected parameters may be erroneous.

—*Incorrect annotation ontology*. Asserted and derived annotations may in reality be compatible, but such compatibility is not evident from the ontology used for annotation. To illustrate this, suppose that the asserted extent

[17]http://pir.georgetown.edu/

[18]http://www.biopax.org/

of the input $\langle op, i\rangle$ specifies that its instances belong to the *Uniprot* database, and its derived extent indicates that $\langle op, i\rangle$ should be able to consume any instance of the *TrEMBL* database. The two annotations are compatible: the *Uniprot* database contains all the protein sequences in *TrEMBL*. Suppose, however that this relationship is not specified in the Extent Ontology, *coveredBy(TrEMBL,Uniprot)* = *false*. As a result, an extent conflict will be detected.

—*Mismatched workflows*. Some of the workflows used for inferring the parameter annotation may contain connected parameters that are incompatible. The workflows used by the annotation algorithm should, in principle, be free from incompatibilities as our method is based on the assumption that only tried and tested workflows should be used for deriving parameters semantics. However, our experiments, as we shall see later, showed that even tested workflows may still suffer from a particular form of mismatch. To illustrate this kind of mismatch, consider a data link that connects an output $\langle op1, o\rangle$ to an input $\langle op2, i\rangle$ such that $\langle op1, o\rangle$ delivers bioinformatics sequences, $domain(\langle op1, o\rangle) = Sequence$, and $\langle op2, i\rangle$ expects protein sequences, $domain(\langle op2, i\rangle) = ProteinSequence$. Given that *Sequence* is not a subconcept of *ProteinSequence* according to the domain ontology, $Sequence \not\sqsubseteq ProteinSequence$, $\langle op1, o\rangle$ and $\langle op2, i\rangle$ are domain incompatible meaning that not all the instances of $\langle op1, o\rangle$ can be used to feed the execution of *op2*. However, *ProteinSequence* is known to be a subconcept of *Sequence*, $ProteinSequence \sqsubseteq Sequence$. This implies that *op2* will accept as input certain instances of $\langle op1, o\rangle$, specifically those that are protein sequences. The workflows containing this form of mismatch may successfully pass a set of tests, (their execution may, with certain inputs, deliver the expected results) and be added to the repository of tried-and-tested workflows as a result.

Identifying the actual source(s) of an annotation conflict may require some detective work on the part of the user as s/he has to examine the above three sources of conflicts and to consult, when available, other sources of information. For example, if workflow provenance that stores details about workflow executions exists then this can help the user in identifying incorrectly annotated parameters by comparing the instances of operation parameters provided by provenance logs with their annotations [Zhao et al. 2004; Bowers et al. 2006].

5.2 Resolution of Annotation Conflicts

Resolving an annotation conflict that an operation parameter suffers from means acting upon one or more of the sources of conflict presented earlier to reduce its set of conflicting parameters to the empty set. Each of the sources of conflict requires different corrective actions. The following describes the corrective actions that can be performed for resolving conflicts in domain annotations and consider for this purpose a parameter $\langle op, p\rangle$ for which asserted and derived domain annotations are conflicting. Note however that the focus on domain conflicts instead of representation or extent conflicts is arbitrary,

and that the same corrective actions that we describe here can be applied for resolving representation and extent conflicts.

Acting on Incorrect Annotations. Assume that, after inspection, the annotator discovers that the domain annotation of $\langle op, p\rangle$ is incorrect, the conflict can be resolved in this case by removing the domain annotation of $\langle op, p\rangle$. As a result of this action, the derived domain annotations of the parameters that are connected to $\langle op, p\rangle$ may have to be recomputed. Instead of removing the domain annotation, the annotator may choose to mark it as incorrect. Keeping this annotation can be useful when reannotating. This is the case, for example, when the correct domain annotation can be obtained by refining the existing one, for example, by choosing a subconcept or a super concept of the semantic domain previously used for annotating $\langle op, p\rangle$.

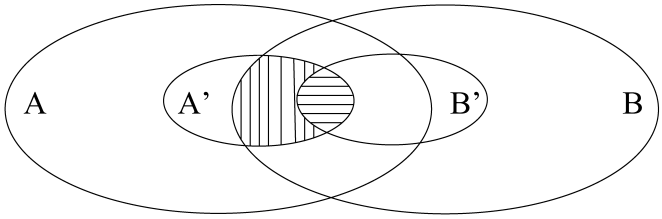
In case the annotator knows the correct domain annotation that $\langle op, p\rangle$ should have, then s/he can modify it accordingly. This action does not guarantee the resolution of the annotation conflict: the new asserted domain annotation of $\langle op, p\rangle$ may still be conflicting with its derived annotation. The domain annotations of the parameters connected to $\langle op, p\rangle$ may also need to be recomputed as a result of this action. As a real example, consider the operation *Restrict*[19] that is used for predicting cut sites in a DNA sequence. The asserted annotation of this operation specifies that it requires as input an enzyme restriction report (*EnzRestReport*). On the other hand, the annotation derived by our annotation algorithm states that its input must be a super concept of *DNASequence*. The two annotations are conflicting: $DNASequence \not\sqsubseteq RestEnzReport$. Upon manual diagnosis, the asserted annotation was found to be incorrect and the conflict was resolved by modifying the asserted annotation of *Restrict* input from *EnzRestReport* to *DNASequence*.

Suppose now that some of the conflicting parameters of $\langle op, p\rangle$ have incorrect asserted domain annotations. The same actions as earlier can be applied to those parameters. The annotation conflict that $\langle op, p\rangle$ suffers from is resolved if the asserted domain annotation of each of the conflicting parameters is removed, marked as incorrect, or modified to be compatible with the asserted domain annotation of $\langle op, p\rangle$.

Acting on the asserted domain annotations of the conflicting parameters may affect other parameters as it may raise new annotation conflicts and resolve other existing ones. In the following we specify the parameters that may be affected when removing, marking as incorrect or modifying the domain annotations of the conflicting parameters of $\langle op, p\rangle$. We also specify how those parameters are affected by specifying the situations in which their sets of conflicting parameters are reduced or enlarged. This analysis can be useful, for example, for the annotator to assess the effects of her/his corrective actions on the annotation of the conflicting parameters.

Let $\langle op', p'\rangle$ be a parameter of the same kind as $\langle op, p\rangle$, that is, $\langle op, p\rangle$ and $\langle op', p'\rangle$ are either inputs or outputs. Acting on the asserted domain annotations of the conflicting parameters of $\langle op, p\rangle$ may affect the parameter $\langle op', p'\rangle$ if and

[19]http://bioweb.pasteur.fr/docs/EMBOSS/restrict.html



Legend

A The set of parameters that are connected to (op,p)
A' The set of parameters that are conflicting with (op,p)
B The set of parameters that connected to (op',p')
B' The set of parameters that are conflicting with (op',p')
(A'∩ B) \ B': the set of parameters that are conflicting with (op,p) and connected to (op',p'), but not conflicting with (op',p')
A'∩ B': the set of parameters that are conflicting with both (op,p) and (op',p')

Fig. 5. Graphical representation of the set of connected parameters and the set of conflicting parameters of (op,p) and (op',p').

only if some of the parameters that are conflicting with $\langle op, p\rangle$ are connected to $\langle op', p'\rangle$, that is:

$$conflictingParams(\langle op, p\rangle, \text{“}domain''\text{)} \cap connectedParams(\langle op', p'\rangle) \neq \phi.$$

Figure 5 depicts the set of connected parameters and the set of conflicting parameters of both $\langle op, p\rangle$ and $\langle op', p'\rangle$. The set of parameters that are conflicting with $\langle op, p\rangle$ and connected to $\langle op', p'\rangle$ corresponds to the set $A' \cap B$. Since the parameters in this set are connected to $\langle op', p'\rangle$, the removal or modification of their domain annotations may either enlarge or reduce the set of conflicting parameters of $\langle op', p'\rangle$. More specifically, the set of parameters that are conflicting with $\langle op', p'\rangle$ may be enlarged when modifying the asserted domain annotations of the parameters that have compatible asserted domain annotations with $\langle op', p'\rangle$, that is, the parameters in[20]

$$(conflictingParams(\langle op, p\rangle, \text{“}domain''\text{)} \cap connectedParams(\langle op', p'\rangle)) \setminus conflictingParams(\langle op', p'\rangle, \text{“}domain\text{”}).$$

In Figure 5, this set corresponds to $(A' \cap B) \setminus B'$. If modified, the domain annotations of the parameters in this set may conflicting with the asserted domain annotation of $\langle op', p'\rangle$, hence enlarging its set of conflicting parameters.

On the other hand, the set of conflicting parameters of $\langle op', p'\rangle$ can be reduced when acting on the parameters whose asserted domain annotations are not compatible with those of $\langle op', p'\rangle$, that is, the parameters in:

$$conflictingParams(\langle op, p\rangle, \text{“}domain\text{”}) \cap conflictingParams(\langle op', p'\rangle, \text{“}domain\text{”}).$$

In Figure 5, this set corresponds to $A' \cap B'$. The parameters in this set may become compatible with $\langle op', p'\rangle$ if their annotations are removed, marked as incorrect or modified. Under certain conditions, the annotation conflict of $\langle op', p'\rangle$ may even be resolved, that is, its set of conflicting parameters may be

[20]The symbol\denotes the set difference operator.

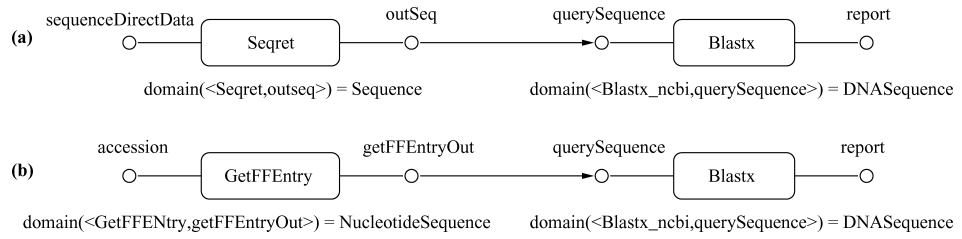


Fig. 6. Examples of conflicts.

reduced to the empty set. In case $\langle op, p\rangle$ and $\langle op', p'\rangle$ are output parameters, the resolution of the domain conflict of $\langle op, p\rangle$ by acting on the asserted domains of its conflicting parameters implies the resolution of the domain conflict of $\langle op', p'\rangle$, if the following conditions are met.

—The set of conflicting parameters of $\langle op', p'\rangle$ is a subset of the set of conflicting parameters of $\langle op, p\rangle$:
$conflictingParams(\langle op', p'\rangle, \text{“}domain\text{”}) \subseteq conflictingParams(\langle op, p\rangle, \text{“}domain\text{”})$.
—The semantic domain of $\langle op', p'\rangle$ is a subconcept of the semantic domain of $\langle op, p\rangle$:
$domain(\langle op', p'\rangle) \sqsubseteq domain(\langle op, p\rangle)$.

As an example, consider the pair of workflows shown in Figure 6, which are used for performing alignment searches over biological sequences. The outputs of the operations *Seqret* and *GetFFEntry* suffer from domain conflicts: both these parameters have asserted domain annotations that are not subconcepts of *DNASequence*. On the other hand, they have the same set of conflicting parameters, $\{\langle Blastx, querySequence\rangle\}$, and the semantic domain of *GetFFEntry*'s output is a subconcept of the semantic domain of *Seqret*'s output: $NucleotideSequence \sqsubseteq Sequence$. Suppose that the domain conflict of *Seqret*'s output has been resolved by removing or marking as incorrect the asserted domain annotation of *Blastx*'s input, the annotation conflict of *GetFFEntry*'s output is also resolved as a result since its new derived annotation is null. Assume now that the domain conflict of *Seqret*'s output has been resolved by modifying the domain annotation of *Blastx*'s input. Since the semantic domain of *GetFFEntry*'s is a subconcept of the semantic domain of *Seqret*'s output, then it is also a subconcept of (and thus compatible with) the new domain annotation of *Blastx*'s input. In both cases, the resolution of the domain conflict of *Seqret*'s output implies the resolution of the domain conflict that *GetFFEntry*'s output suffers from.

Acting on Workflow Definitions. An annotation conflict can be due to an error in a workflow, that is, to a data link that connects $\langle op, p\rangle$ to an incompatible parameter $\langle op', p'\rangle$. The user may choose to mark such a data link as incorrect. When recomputing the derived annotations of both $\langle op, p\rangle$ and $\langle op', p'\rangle$, marked data links will not be considered. Alternatively, since the workflows that contain mismatched data links are likely to contain other data links that connect incompatible parameters, the annotator may choose to mark a whole workflow as mismatched. When recomputed, the derived annotations

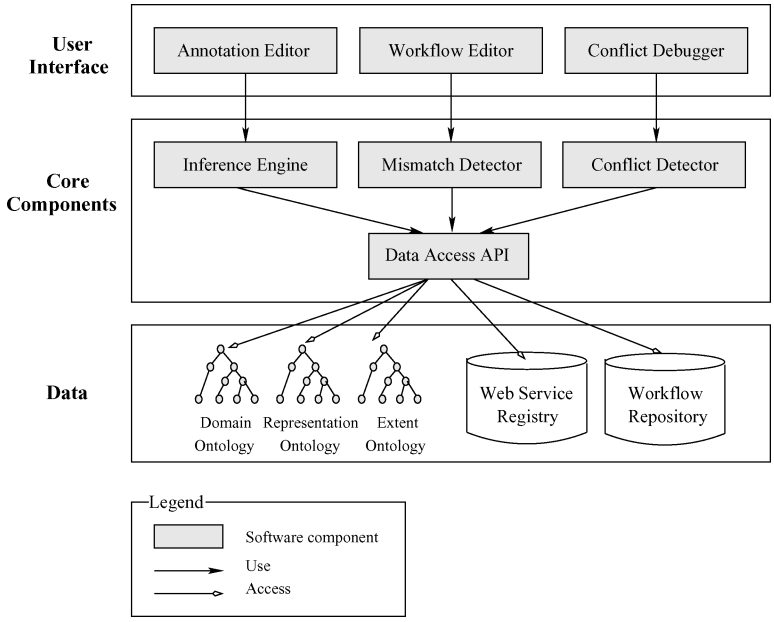


Fig. 7. Architecture of the annotation workbench.

of the operation parameters involved in such workflows may change as a result.

Acting on the Ontology Used for Annotation. The asserted and the derived annotations of $\langle op, p\rangle$ may in reality be compatible, but due to an error in the ontology they appear to be conflicting. Consider, for example, that $\langle op, p\rangle$ is an output that is connected to an input $\langle op', p'\rangle$, such that $domain(\langle op, p\rangle) = PolyPeptide$ and $domain(\langle op', p'\rangle) = Sequence$. The two parameters are in reality compatible, $PolyPeptide \sqsubseteq Sequence$. However according to the domain ontology they are conflicting. Such incompatibility can be resolved by specifying a subsumption relationship in the domain ontology that links *PolyPeptide* (or one of its super concepts) to *Sequence* (or one of its subconcepts).

6. IMPLEMENTATION

To assess the value of the annotation derivation mechanism presented here, we implemented an annotation workbench the overall architecture of which is illustrated in Figure 7. The workbench provides a GUI for annotating Web services, debugging conflicts between asserted and inferred annotations, and for analyzing workflows for mismatches. To do so, it relies on the functionalities of three core components: the *Inference Engine* implements the annotation derivation algorithm, the *Conflict Detector* is used for identifying annotation conflicts, and the *Mismatch Detector* identifies mismatches between connected parameters in workflows. These components access a repository of workflows, a Web service registry containing the semantic annotations, and the ontologies used for annotation utilizing the *Data Access API* component.

It is worth noting that while the example of ontologies and semantic annotations used throughout this paper belong to the domain of bioinformatics, the annotation derivation mechanism and the workbench implemented can be applied to any domain. For example, if a user wants to derive annotations for Web services that belong to a domain different from bioinformatics, then s/he can use the annotation workbench for that purpose by providing the necessary inputs, that is, the ontologies that model the domain of interest, workflows that connects some of the Web services in the domain together with some example of asserted annotations from which other annotations can be inferred.

The following presents in more detail the functionalities provided by the annotation workbench.

6.1 Supporting the Manual Annotation of a Web Service

A user annotates a Web service by manually relating the service elements (i.e., service operations, and their input and output parameters) to concepts from the ontologies used for annotation using the GUI illustrated in Figure 8. Once the annotator has chosen a service for annotation, the service details are displayed in the panel labeled A in Figure 8. To annotate a service element, for example, an operation parameter, the user browses the domain, representation and extent ontologies (labeled B in Figure 8), and selects a concept from each of these ontologies. At the end of the annotation task, the user submits the new annotation to the service registry for publication.

If the user starts to annotate an operation parameter that has a loose annotation derived for it, then he or she only has to choose from the (hopefully small) subset of the ontology indicated by the loose annotation, rather than from the full set of ontology concepts. For example, when specifying the semantic domain of the input parameter belonging to the *Blast* operation given in an earlier example (Figure 1), the user has only to choose from the collection of five concepts specified by the loose annotation (labeled D in Figure 8), rather than all the concepts in the ontology (labeled C). Where the ontology used for annotation is large and/or complex, this can result in a significant time saving for the human annotator and may reduce the chances of making manual annotation errors.

6.2 Identifying and Resolving Annotation Conflicts

As well as supporting the annotator in the manual annotation task, the tool automatically identifies conflicts between asserted and derived annotations. When a conflict, for example, in a domain annotation, is detected, the button labeled E in Figure 8 is enabled. By clicking on this button, the panel illustrated in Figure 9 is displayed. This panel allows the annotator to perform the operations implementing the actions described earlier in Section 5.2 to resolve the errors responsible for the detected conflict. The annotator can remove, mark as incorrect or modify the asserted annotation of the parameter for which the conflict exists (labeled A in Figure 9) or the asserted annotations of its connected parameters (labeled B). The user can also mark as mismatched the workflows or data links used by the annotation inference algorithm (labeled C in Figure 9), or

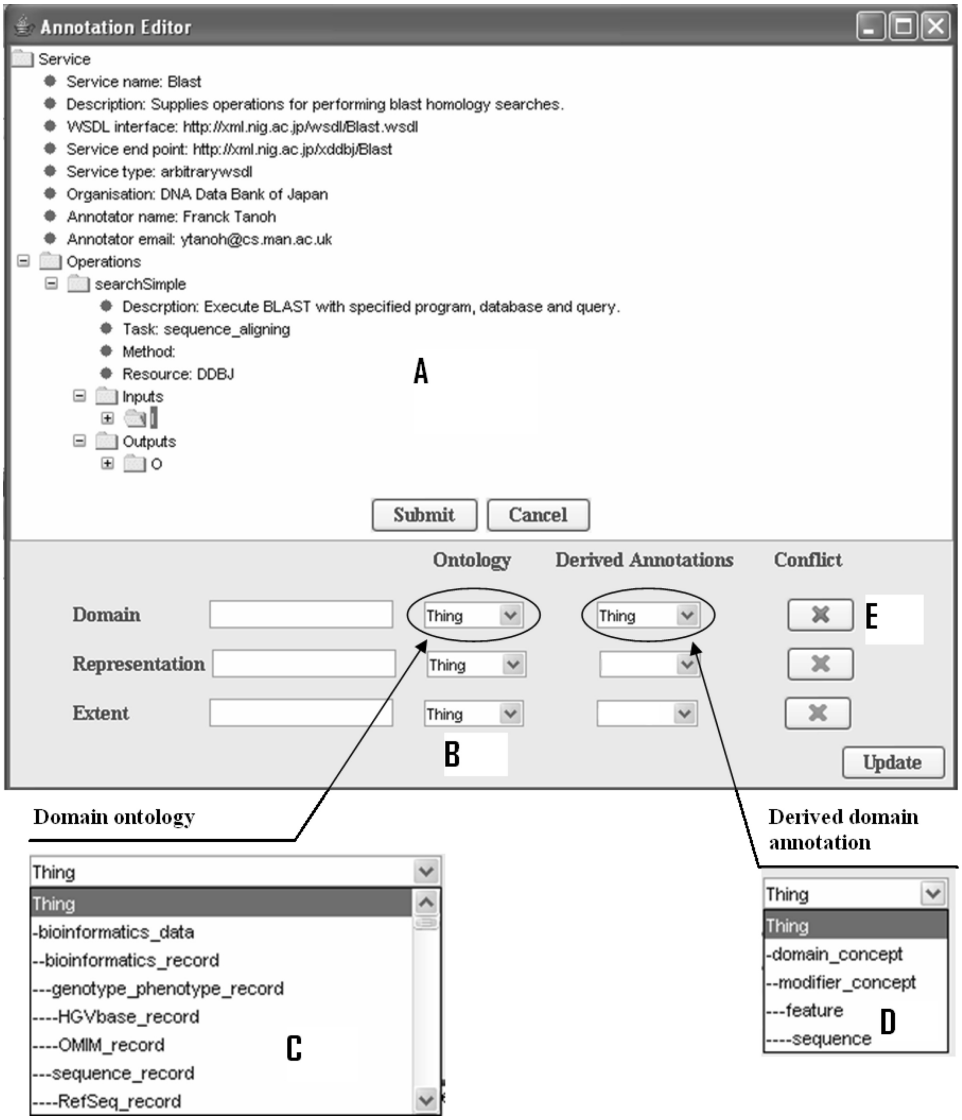


Fig. 8. The annotation editor GUI used for assigning ontological concepts to service elements.

add new relationships between the concepts of the ontology used for annotation (labeled D in Figure 9).

6.3 Inspecting Workflows Using Derived Annotations

As mentioned in the introduction, semantic (tight) annotations of Web services can be used for inspecting workflows for errors. This can be done by identifying the data links that violate the parameter compatibility rules presented in Section 3. The natural question that arises is, when derived loose annotations are available, can they be used for inspecting workflows for errors? And if so, what

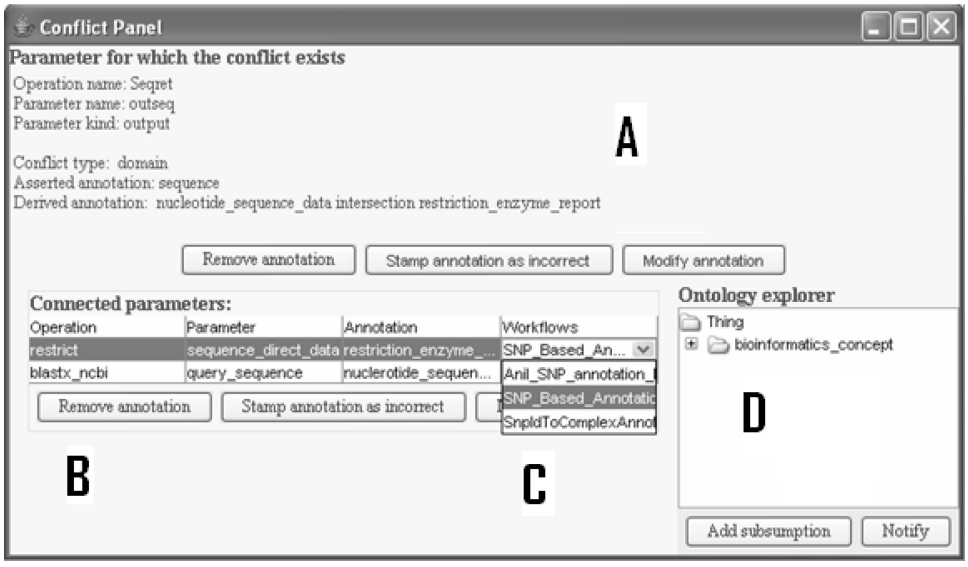


Fig. 9. A graphical interface that allows resolving annotation conflicts by acting on parameters' annotations, workflows and ontologies.

are the parameter compatibility rules to be used for identifying mismatched data links?

Let us examine the case of semantic domain annotations. According to the compatibility rule given in Section 3, an output $\langle op1, o\rangle$ is compatible with an input $\langle op2, i\rangle$ in terms of domain if and only if the semantic domain of $\langle op1, o\rangle$ is a subconcept of the semantic domain of $\langle op2, i\rangle$:

$$domain(\langle op1, o\rangle) \sqsubseteq domain(\langle op2, i\rangle).$$

Now assume that neither $\langle op1, o\rangle$ nor $\langle op2, i\rangle$ have asserted domain annotations but both have loose domain annotations. As shown earlier, the semantic loose annotation specifies an upper bound on the semantic domain of the outputs, that is, $domain(\langle op1, o\rangle) \sqsubseteq getOutputDomain(\langle op1, o\rangle)$, and a lower bound on the semantic domain of the inputs, that is, $getInputDomain(\langle op2, i\rangle) \sqsubseteq domain(\langle op2, i\rangle)$. Therefore, in order for the two parameter $\langle op1, o\rangle$ and $\langle op2, i\rangle$ to be domain compatible, it is sufficient that:

$$getOutputDomain(\langle op1, o\rangle) \sqsubseteq getInputDomain(\langle op2, i\rangle).$$

Note that the previous condition may well pose a stronger condition for compatibility than is required, however, it is conservatively true given the information we have available in the loose annotations. In other words, when the earlier condition is met then the input and output parameters are definitely domain compatible; however, when such a condition is not satisfied then the two parameters are potentially (but not necessarily) domain incompatible. This is perhaps better explained using an example. Consider the workflow shown in Figure 10. It is used for performing value-added protein identification in which

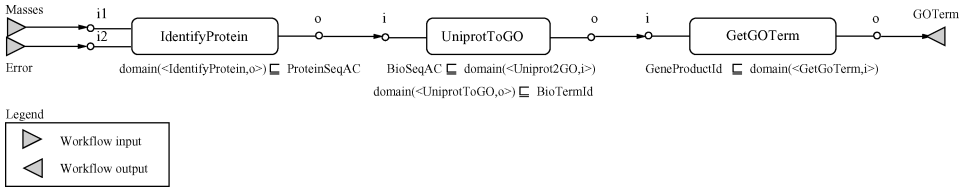


Fig. 10. Value-added protein identification.

protein identification results are augmented with additional information from the Gene Ontology[21] [Belhajjame et al. 2005].

The workflow consists of three operations. The *IdentifyProtein* operation takes as input peptide masses obtained from the digestion of a protein together with an identification error and outputs the Uniprot accession number of the "best" match. Given a Uniprot accession number, the operation *UniprotToGO* delivers its corresponding gene ontology identifiers. The *GetGOTerm* operation takes as input a gene ontology identifier and delivers the associated gene ontology term. The parameters of the three operations do not have any associated tight annotations. However, we have been able to derive loose domain annotations of some of the parameters (see Figure 10). Using these loose annotations, let us check the domain compatibility of the two data links connecting *IdentifyProtein*'s output to *Uniprot2GO*'s input, and *Uniprot2GO*'s output to *GetGOTerm*'s input. The semantic domain of *IdentifyProtein*'s output is a subconcept of protein sequence accession, *ProteinSeqAC*, and, the semantic domain of *Uniprot2GO*'s input is a super concept of bioinformatics, sequence accession, *BioSeqAC*. *ProteinSeqAC* is known to be a subconcept of *BioSeqAC*. Therefore, using the domain compatibility condition presented earlier, we can conclude that the two parameters are domain compatible. On the other hand, the derived loose annotations specify that the semantic domain of *Uniprot2GO*'s output is a subconcept of bioinformatics, term identifier, *BioTermId*, and that the semantic domain of *GetGOTerm*'s input is a superconcept of gene product identifier, *GeneProductId*. Given that *BioTermId* is not known to be a subconcept of *GeneProductId*, the two parameters are potentially incompatible.

Because derived loose annotations allow the detection of potential (not certain) mismatches, when both tight and loose annotations are available the use of tight annotations should be preferred for detecting mismatches. In the case where only one side of a data link has a tight annotation then we can check the compatibility of the parameters the data link connects using a rule that is less strict than the compatibility rule expressed only in terms of loose annotations. Consider, for example, the case where the output $\langle op1, o\rangle$ has a tight domain annotation whereas the input $\langle op2, i\rangle$ has only a derived domain annotation. For the two parameters to be domain compatible, it is sufficient that.

$$domain(\langle op1, o\rangle) \sqsubseteq getInputDomain(\langle op2, i\rangle).$$

The previous analysis of domain compatibility also applies to representation compatibility and extent compatibility. The following presents the conditions

[21] http://www.geneontology.org/

that can be used for verifying representation and extent compatibility using the loose annotations of operation parameters.

—*Representation compatibility.* Two connected parameters that are domain compatible are compatible in terms of representation if the derived representations of the output are a subset of the derived representations of the input. Specifically, the output $\langle op1, o\rangle$ and the input $\langle op2, i\rangle$ are representation compatible if:

$$(getOutputDomain(\langle op1, o\rangle) \sqsubseteq getInputDomain(\langle op2, i\rangle)) \quad \text{and}$$
$$(getOutputRepresentations(\langle op1, o\rangle) \subseteq getInputRepresentations(\langle op2, i\rangle)).$$

—*Extent compatibility.* Two connected parameters that are representation compatible are compatible in terms of extent if the derived extent of the output is covered by the derived extent of the input. Specifically, the output $\langle op1, o\rangle$ and the input $\langle op2, i\rangle$ are compatible in terms of extent if:

$$(getOutputDomain(\langle op1, o\rangle) \sqsubseteq getInputDomain(\langle op2, i\rangle)) \quad \text{and}$$
$$(getOutputRepresentations(\langle op1, o\rangle) \subseteq getInputRepresentations(\langle op2, i\rangle))$$
$$\text{and}$$
$$coveredBy(getOutputExtent(\langle op1, o\rangle), getInputExtent(\langle op2, i\rangle)).$$

We have developed a tool that implements the above compatibility rules for identifying potential mismatches in workflows using derived loose annotations of operation parameters. It extends a tool that we have developed in previous work for detecting errors in workflows based on the semantic tight annotations of operation parameters [Belhajjame et al. 2006]. The tool examines the data links of a given workflow. If a potential mismatch is detected, then the workflow is modified to indicate the location of the mismatch. For example, in the protein identification workflow, the data link connecting *Uniprot2GO()* to *GetGOTerm()* is potentially mismatched. The mismatch is flagged by inserting a labeled red box between *Uniprot2GO()*'s output and *GetGOTerm()*'s input. Since the detected mismatch is not certain, the tool allows the user to confirm that the data link is not mismatched based on his/her better knowledge of the real semantics of the operation parameters.

7. EVALUATION

To further assess the value of the annotation derivation method presented in this paper, we applied the annotation algorithm to a repository of real workflows and a small set of real (manually asserted) annotations taken from the domain of bioinformatics. The objective of this experiment was to see whether the annotation algorithm is able to derive new annotations from a small set of existing manual annotations. The annotations derived by the algorithm are loose and, therefore, contain less information than conventional tight annotations. To show that despite their loose nature the derived annotations are still of value and worth the effort made to collect them, we conducted an experiment to assess their utility in practice. We issued a set of service discovery queries with and without considering the derived annotations. We then examined the

results of the queries to see whether the use of derived annotations improves service discovery in terms of recall and precision.

7.1 Assessing the Ability of the Annotation Algorithm to Infer New Annotations

A large number of public Web services are available in bioinformatics, for example, the myGrid toolkit provides access to over 3000 third party bioinformatics Web services. These services have been used by bioinformaticians and biologists for composing their workflows; the Taverna repository contains 131 bioinformatics, workflows[22]. Some of the services used in these workflows were annotated by domain experts. At the time of writing, the myGrid Web service registry, Feta [Lord et al. 2005], provides parameter annotations for 33 services[23].

We used these as inputs to our algorithm, and were able to derive 35 new domain annotations for operation parameters. Upon analysis with the help of a domain expert, 11 of the derived annotations were found to be incorrect. The errors in derived annotations are not due to problems with the annotation derivation algorithm, but rather to the following:

—Errors in the original annotations: of the 11 erroneously derived annotations, four were found to have been derived from parameters that were incorrectly annotated by a human annotator.

—Incompatibilities between connected parameters in the workflows: seven incorrect annotations were derived using data links that connect incompatible operation parameters. The existence of mismatched workflows in the repository may be explained by the fact that mismatched workflows may, in certain cases, be executed successfully and deliver the expected results. For example, we found in one of the workflows a data link connecting the *Seqret* operation that delivers *Sequences* to the *GetGenePredict* operation that requires *DNA Sequences*. This data link is mismatched: *Sequence* is not a subconcept of *DNA Sequence*. However, *DNA Sequence* is known to be a subconcept of *Sequence*. This means that *GetGenePredict* will accept as input those outputs of *Seqret* that are *DNA sequences*. The workflow containing this data link may have passed the tests successfully, and have been, as a result, added to the workflow repository.

Of the 11 incorrect annotations, five were identified by diagnosing the conflicts automatically detected by the annotation tool between asserted and derived annotations. For example, a conflict was detected between the annotation manually asserted for the input parameter *query_sequence* of the *blastx_ncbi*, *NucleotideSequence*, and its derived annotation that states that it must be a superconcept of *Sequence*. According to the myGrid ontology, as well as common sense, *NucleotideSequence* is not a super-concept of *Sequence*. After manual diagnosis of the annotation conflict, the derived annotation was found to be

[22]The workflow specifications are accessible at http://myexperiment.org/.

[23]The reader will note how the number of annotations lags far behind the number of available services and even behind the number of workflows.

incorrect due to a data link that connected the parameter *query_sequence* to an incompatible parameter.

The remaining six incorrect derived annotations were discovered when we manually investigated the derived annotations for correctness.

This experiment showed that it is possible to derive a significant number of new annotations from a small annotation repository. Unfortunately, some of the derived annotations were found to be incorrect due to errors in workflows and existing manual annotations, highlighting the importance of inspecting the correctness of derived annotations. In this respect, the experiment showed that the diagnosis of the conflicts automatically detected by the tool between asserted and derived annotations can help the annotator uncover errors in workflows and manual annotations, and thus identify those annotations that were incorrectly inferred because of these errors.

7.2 Using Derived Annotations for Service Discovery

The annotations we derived for the bioinformatics, Web services are less informative than conventional tight annotations as they do not provide the "exact" semantics of operation parameters. We have shown in Section 6 that, despite their loose nature, derived annotations have utility in supporting the manual annotation of Web services and in inspecting workflows for mismatches. To further assess the usefulness of derived annotations and provide experimental evidence that demonstrates their utility in practice, we conducted an experiment with the objective of assessing the degree to which they may improve service discovery. To this end, we issued a set of service discovery queries and compared the results obtained with and without derived annotations. We specifically considered the following two kinds of query:

—Queries that retrieve service operations requiring an input matching a given semantic domain, c. That is, the operations having an input, the semantic domain of which is a subconcept of c. When derived annotations are considered, the query also returns those operations having an input whose derived domain annotation is a subconcept of c.

—Queries that retrieve service operations delivering an output matching a given semantic domain, c. That is, operations having an output, the semantic domain of which is a subconcept of c. When derived annotations are considered, the query also returns those operations having an output whose derived domain annotation is a subconcept of c.

Figure 11 illustrates the number of service operations retrieved by each of the discovery queries considered. For example, the two columns labeled *Sequence* in the left hand chart illustrate the number of service operations retrieved that require *Sequence* as input with and without considering derived annotations. The charts show that the use of derived annotations increases the number of service operations retrieved. For example, the number of operations found that require a *Sequence* as input has been quadrupled with respect to the number of operations retrieved using only existing asserted annotations, and the number of service operations that produce gene ontology identifiers (*GOId*) has been doubled.

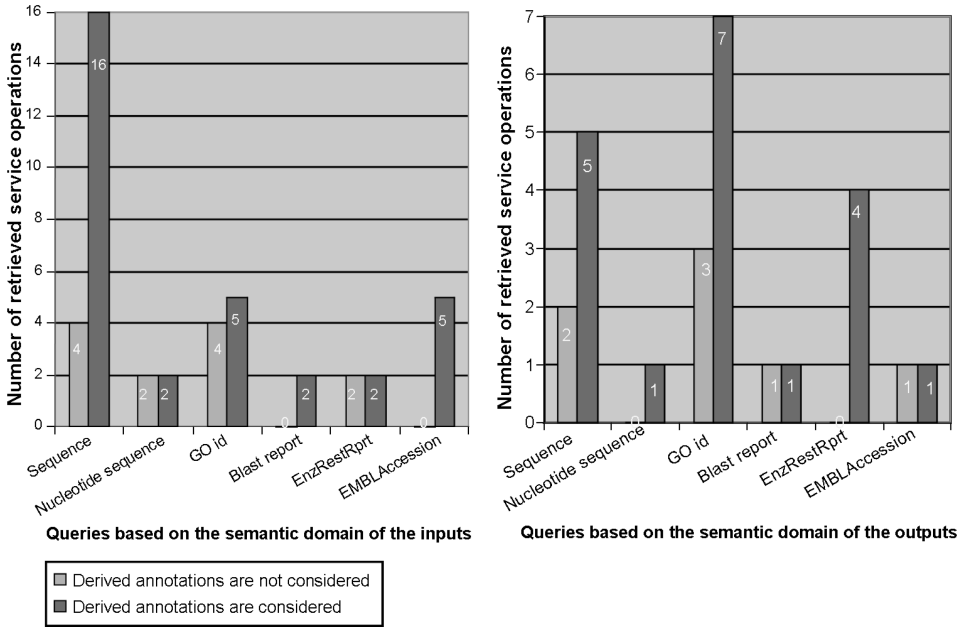


Fig. 11. Number of retrieved service operations with and without using derived annotations.

To assess the effects of possible errors in derived annotations as well as their loose nature on the query results we measured the precision of the results for each of the discovery queries, and to see whether the use of derived annotations allows discovering more relevant service operations we measured the queries' recall (Table I and Table II). The precision is defined as the ratio of the number of relevant operations retrieved to the number of operations retrieved, and the recall as the ratio of the number of relevant operations retrieved to the number of relevant operations appearing within the workflows in the repository. For each query, the set of relevant operations in the workflows were manually identified with the help of a domain expert.

Table I shows the precision and the recall of the discovery queries that locate service operations based on the semantics of their inputs. Using derived annotations, the precision remains unchanged for the first three queries and changed from *undefined* to 100% for the fourth query: all the operations retrieved that require *Sequence*, *NucleotideSequence*, *GOId* and *BlastReport*, respectively, were found to be relevant.

The precision of the query locating the operations that require an *EMBLAccession* has changed from *undefined* to 20%: of the five operations retrieved, only the operation *queryHgvbaseByEmblAccNumber* was found to be relevant. The analysis of the remaining four operations revealed that they are not relevant because the lower bound specified by the derived domain annotation of their input does not match their actual domain annotation. Take, for example, the retrieved operation *queryXByRef*. The derived domain annotation of its input parameter specifies that it is a super concept of *EMBLAccession* whereas its actual domain annotation is *SequenceAccession*. Although compatible, the

Table I. Precision and Recall of the Discovery Queries Based on the Semantic Domain of the Inputs

	Precision[a] (%)	Precision[b] (%)	Recall[a] (%)	Recall[b] (%)
Sequence	100	100	8	32
NucleotideSequence	100	100	5	5
GOId	100	100	57	71
BlastReport	undefined[c]	100	0	25
EnzRestReport	0	0	0	0
EmblAccession	undefined[c]	20	0	100
Average	undefined	70	12	26

[a]Derived annotations are not considered.
[b]Derived annotations are considered.
[c]Undefined precision because the set of retrieved service operations is empty.

Table II. Precision and Recall of the Discovery Queries Based on the Semantic Domain of the Outputs

	Precision[a] (%)	Precision[b] (%)	Recall[a] (%)	Recall[b] (%)
Sequence	100	80	11	22
NucleotideSequence	undefined	100	0	20
GOId	100	86	43	86
BlastReport	100	100	14	14
EnzRestReport	undefined	0	0	0
EmblAccession	100	100	100	100
Average	undefined	61	28	40

derived and the asserted annotations are not equivalent: *SequenceAccession* is a strict super concept of *EMBLAccession*. Because of this, *queryXByRef* is irrelevant for the discovery query that locates the operations that require an *EMBLAccession*, that is, the operations having an input that is equivalent to or subconcept of *EMBLAccession*.

It is worth noting that while the four retrieved operations are not relevant for the issued discovery query since they do not *require EMBLAccession*, they are relevant for the query that fetches the operations that *accept EMBLAccession*. For example, the operation *queryXByRef* accepts as input *EMBLAccession* since *EMBLAccession* is a subconcept of the semantic domain of *queryXByRef* 's input: *SequenceAccession*. This kind of query is particularly useful when composing workflows for locating the service operations able to consume the data produced by a constituent operation of the workflow being designed [Belhajjame et al. 2006].

Regarding the impact of incorrect derived annotations on the query results, we observed that they did not negatively affect the precision of any of the input-based discovery queries. The reason is that most of the inputs for which incorrect domain annotations have been inferred, belong to semantic domains that are subconcepts of their derived domains. For example, for some of the operation inputs that belong to *ProteinSequence*, *NucleotideSequence* or *DNASequence*, the inferred domain stated that they are *Sequences*. As such, these operations were retrieved by the discovery query that locates the operations that require a Sequence. Although the derived and the asserted annotations of the inputs

of these operations are conflicting, given that *ProteinSequence*, *NucleotideSequence* and *DNASequence* are subconcepts of *Sequence*, such operations were found to be relevant for the issued discovery query. This, surprisingly, shows that even incorrectly derived annotations can in certain cases be of value and help locating relevant service operations.

Regarding the recall of the technique overall, Table I shows that the use of derived annotations has considerably improved the recall of four queries out of six. For example, the number of retrieved service operations that require *Sequence* has increased from four to 16, thereby covering 32% of the available relevant operations.

Table II shows the precision and the recall obtained from the discovery queries that search for service operations based on the semantics of their outputs. The precision remains unchanged for the queries locating the operations that deliver *BlastReport* and *EMBLAccession*, respectively, and changed from *undefined* to 100% for the query retrieving the operations that produce a *NucleotideSequence*. On the other hand, it decreased for the queries retrieving the operations that produce a *Sequence* and a *GOId*. Of the five retrieved service operations that output *Sequence*, the operation *getEmblAccession* was found to be irrelevant. This operation was retrieved because its derived domain annotation is incorrect: its output is connected to an incorrectly annotated input of the operation *seqret*. Of the seven retrieved service operations that output *GOIds*, *queryByxRef* was found to be irrelevant. This operation was retrieved because its derived annotation is incorrect due to a mismatched data link connecting its output to the input of the operation *addTerm*. Notice also that the precision of the query retrieving the operations that output an enzyme restriction report is zero: the outputs of the four retrieved operations have incorrect derived annotations that were inferred using erroneous input annotations.

Regarding recall, Table II shows that it improved for the first three queries: the number of service operations that produce a *Sequence* and a *GOId*, has doubled in each case, and the number of operations found that output *NucleotideSequence* has increased to cover 20% of the available relevant service operations.

This experiment showed that:

—the use of derived annotations significantly increases the number of service operations located by discovery queries: the recall average increased from 12% to 26% for input-based service discovery queries, and from 28% to 40% for output-based service discovery queries.

—erroneous derived annotations have some impact on the precision of discovery queries. This impact is, however, relatively small compared with the benefits gained in terms of recall. Only two output-based discovery queries have seen their precision drop due to errors in derived annotations. Regarding input-based discovery queries, the errors in derived annotations did not have a negative impact on the precision of service discovery. In contrast, as shown earlier, they allowed locating relevant service operations. This is because the erroneous annotations were within the same concept hierarchy as the correct

annotations; specifically, the inferred annotations were super concepts of the correct annotations rather than being unrelated.

—due to the loose nature of derived annotations, input-based service discovery queries may return irrelevant operations (e.g., as we have seen earlier, the discovery query that locates the operations that require *EMBLAccession* returns irrelevant operations because of this). In the conducted experiment this occurs relatively infrequently; of the 6 queries only 1 suffered from this problem. Nevertheless, it suggests that the operations returned by input-based discovery queries should be checked for relevance when derived annotations are used. Note that, on the other hand, output-based discovery queries do not suffer from this problem. The derived annotation of output parameter $\langle op, o\rangle$ specifies an upper bound domain: $domain(\langle op, o\rangle) \sqsubseteq c$. Therefore, the operation $\langle op, o\rangle$ is definitely relevant for the discovery query that retrieves the operations whose outputs are equivalent to or subconcepts of the semantic domain c.

8. RELATED WORK

As well as describing Web service parameters, as we have seen throughout this paper, semantic annotations can be used for describing other aspects of Web services, for example, the tasks performed by service operations within a domain of interest [Lord et al. 2005] and the relationship between the inputs and outputs of Web services. For example, Hull et al. [2006] have proposed a framework for matching stateless Web services in which the inputs and outputs of a given service operation are associated using description logic assertions that relate the semantic concepts used for their description.

Semantic annotations are key components of several semantic Web service applications. They can be used, as seen in the experimental evaluation, for discovering Web services based on the semantics of their inputs/outputs or the task they implement [Ludwig and Reyhani 2006; Sycara et al. 2003]. They can also be used for guiding the composition of workflows by automatically suggesting the Web services that can safely extend an incomplete workflow [Berardi et al. 2005; Traverso and Pistore 2004], and in detecting mismatches between connected parameters in pre-designed workflows [Bussler et al. 2002; Nezhad et al. 2006].

Unfortunately, the scarcity of service annotations remains a critical bottleneck in the delivery of the above functionalities. This has been recognised by a number of researchers who have proposed mechanisms by which annotations can be learned or inferred by using existing classic schema matching and machine learning techniques [Mitra et al. 2000; Rahm and Bernstein 2001; Mitchell 1997].

Patil et al. [2004], taking inspiration from the schema matching problem, have developed a tool using the WSDL elements which are automatically matched to ontology concepts based on their linguistic and structural similarity. The framework was then adapted to make use of machine learning classification techniques in order to select an appropriate domain ontology to be used for annotation [Oldham et al. 2004].

Heß et al. [2004] have designed a tool called ASSAM, which uses text classification techniques to learn new semantic annotations for individual Web services from existing annotations of other Web services [Heß and Kushmerick 2003]. Specifically, the input and output parameters of a Web service are represented using vectors of terms that are constructed from the names of the parameters. A Bayesian classifier is then used to match constructed vectors to ontology concepts.

Lerman et al. [2006] proposed a classification method for assigning semantic concepts to the inputs and outputs of Web services. They adopted a method similar to that of Assam for annotating input parameters. However, they developed a different method for annotating output parameters, which uses as input the instances delivered by the Web service. Specifically, they elaborated an algorithm that learns the pattern that characterizes a given semantic domain using sample instances. Given the instances delivered by a nonannotated output, a content based classifier is used to assign the output to the pattern that characterizes it and, thus, to a semantic concept that can be used for its annotation.

Dong et al. [2004] developed a search engine for web service discovery called Woogle, which is used for locating service operations based on the name of their inputs or outputs. Different from the previous approaches, which assign Web service parameters to concepts from ontologies, in Woogle the inputs and outputs of Web services are clustered into groups using unsupervised learning techniques [Mitchell 1997]. Parameters that belong to the same group are assumed to have the same semantics.

The proposals just discussed require as input information that is readily available and which can be extracted from the WSDL documents that describe the Web services. Therefore, they are able to infer semantic annotations for (almost) any Web service parameter. However, they are based on assumptions that often do not hold in practice and, therefore, may well generate inaccurate annotations. Moreover, they do not provide a means by which the correctness of inferred annotations can be verified.

For example, most of machine learning inspired proposals assume that parameters with the same name have the same semantics. This is not always true. For example, both of the operations *GetDADEntry* and *GetUniprotEntry* provided by the DNA Data Bank of Japan have an output named *Result*. Yet, the semantic domain of the output of *GetDADEntry* is a DNA sequence whereas that of the output of *GetUniprotEntry* is a Protein Sequence. This observation holds for a large number of currently available Web services.

Schema mapping-based proposals, for example, Meteor-S [Patil et al. 2004], annotate a parameter using the semantic concept with the closest structure to that of the data type of the parameter. In so doing, they assume that the parameters are well typed, that is, the data type provides detailed information about the parameter internal structuring. Unfortunately, as mentioned earlier, the parameters of currently available Web services are often weakly typed. For example, the parameters of most of the Web services that we found and used in our experiment are typed either as a string or a collection of strings, regardless of the complexity of the content of their instances.

Different from the previous proposals, in our solution we can infer annotation of a parameter only if it is connected to another input within a tried and tested workflow. Nevertheless, in our case, the annotations are not inferred based on heuristics but using compatibility conditions between connected parameters in workflows that if tested and tried are likely to lead to accurate annotations. Furthermore, we are able to assess, to a certain extent, the correctness of inferred annotations by automatically detecting annotation conflicts.

More recently, Bowers et al. have proposed a technique by which the semantics of the output of a service operation is computed from information describing the semantics of the operation's inputs, and vice-versa [Bowers and Ludäscher 2005, 2006]. Specifically, they assume that the relationships between the semantics of the Web service parameters is available and encoded in the form of a query expression using which the semantics of the outputs can be computed when the annotations of the inputs are available. This approach is similar to ours in that annotations are inferred based on associations that relate Web service parameters. However, it relies on information that is scarce: we are not aware of any accessible source that provides queries specifying the relationships between the inputs and outputs of service operations. This can be partly explained, as pointed out by the authors themselves, by the fact that specifying such queries is not a straightforward task.

9. CONCLUSIONS

This article shows that valuable information about service annotations can be automatically inferred based on the workflows in which the Web services are involved. The proposed method improves on existing work in this area in that, in addition to supporting the manual annotation task, it can be used for inspecting the compatibility of parameters in workflows and detecting errors in manual annotations and the ontology used for annotation.

The annotation derivation mechanism was implemented and experimentally evaluated. The results provided evidence in support of our annotation mechanism and showed its effectiveness and ability to discover new annotations from a small set of existing (manual) annotations and to help detecting mistakes in existing annotations. The experiments also demonstrated the value of the inferred annotations by showing that their use considerably increases the number of services located by discovery queries even with a small starting set of manual annotations.

ACKNOWLEDGMENTS

We are grateful to Antoon Goderis and Peter Li who provided us with access to the myGrid workflow repository that was used in the experimental evaluation, and to Duncan Hull, Franck Tanoh, Katy Wolstencroft and Jun Zhao who helped in the analysis and validation of the experimental results.

REFERENCES

Belhajjame, K., Embury, S. M., Fan, H., Goble, C. A., Hermjakob, H., Hubbard, S. J., Jones, D., Jones, P., Martin, N., Oliver, S., Orengo, C., Paton, N. W., Poulovassilis, A., Siepen, J., Stevens, R., Taylor,

C., VINOD, N., ZAMBOULIS, L., AND ZHU, W. 2005. Proteome data integration: Characteristics and challenges. In *Proceedings of the UK All Hands Meeting*. National e-Science Centre, Nottingham, UK.

BELHAJJAME, K., EMBURY, S. M., AND PATON, N. W. 2006. On characterising and identifying mismatches in scientific workflows. In *Proceedings of the 3rd International Workshop on Data Integration in the Life Sciences (DILS 06)*. Springer, 240–247.

BELHAJJAME, K., EMBURY, S. M., PATON, N. W., STEVENS, R., AND GOBLE, C. A. 2006. Automatic annotation of Web services based on workflow definitions. In *Proceedings of the 5th International Semantic Web Conference*. Springer, 116–129.

BENATALLAH, B., HACID, M.-S., LÉGER, A., REY, C., AND TOUMANI, F. 2005. On automating Web services discovery. *VLDB J. 14,* 1, 84–96.

BERARDI, D., CALVANESE, D., GIACOMO, G. D., HULL, R., AND MECELLA, M. 2005. Automatic composition of transition-based semantic Web services with messaging. In *Proceedings of the 31st International Conference on Very Large Data Bases*, Trondheim, Norway. 613–624.

BOWERS, S. AND LUDÄSCHER, B. 2005. Towards automatic generation of semantic types in scientific workflows. In *WISE 2005 International Workshops*. Springer, 207–216.

BOWERS, S. AND LUDÄSCHER, B. 2006. A calculus for propagating semantic annotations through scientific workflow queries. In *Query Languages and Query Processing Workshop (QLQP'06) in the 10th International Conference on Extending Database Technology*. Springer, 712–723.

BOWERS, S., MCPHILLIPS, T. M., LUDÄSCHER, B., COHEN, S., AND DAVIDSON, S. B. 2006. A model for user-oriented data provenance in pipelined scientific workflows. In *Proceedings of the International Provenance and Annotation Workshop (IPAW)*, L. Moreau and I. T. Foster, Eds. Lecture Notes in Computer Science, vol. 4145. Springer, 133–147.

BUSSLER, C., FENSEL, D., AND MAEDCHE, A. 2002. A conceptual architecture for semantic Web-enabled Web services. *SIGMOD Record 31,* 4, 24–29.

CARDOSO, J. AND SHETH, A. P. 2003. Semantic e-workflow composition. *J. Intell. Inform. Syst. 21,* 3.

DONG, X., HALEVY, A. Y., MADHAVAN, J., NEMES, E., AND ZHANG, J. 2004. Simlarity search for Web services. In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada. 372–383.

GOBLE, C. A., WOLSTENCROFT, K., GODERIS, A., HULL, D., ZHAO, J., ALPER, P., LORD, P., WROE, C., BELHAJJAME, K., TURI, D., STEVENS, R., AND ROURE, D. D. 2006. *Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences*. Springer Verlag, To appear.

HEß, A., JOHNSTON, E., AND KUSHMERICK, N. 2004. Assam: A tool for semi-automatically annotating semantic Web services. In *Proceedings of the 3rd International Semantic Web Conference*. Springer, 320–334.

HEß, A. AND KUSHMERICK, N. 2003. Learning to attach semantic metadata to Web services. In *Proceedings of the 2nd International Semantic Web Conference*. Springer, 258–273.

HULL, D., ZOLIN, E., BOVYKIN, A., HORROCKS, I., SATTLER, U., AND STEVENS, R. 2006. Deciding semantic matching of stateless services. In *Proceedings of the 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference*, MA. AAAI Press.

LERMAN, K., PLANGPRASOPCHOK, A., AND KNOBLOCK, C. A. 2006. Automatically labeling the inputs and outputs of Web services. In *Proceedings of the 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference*. MA. AAAI Press.

LORD, P. W., ALPER, P., WROE, C., AND GOBLE, C. A. 2005. Feta: A lightweight architecture for user oriented semantic service discovery. In *Proceedings of the 2nd European Semantic Web Conference (ESWC'5)*. Springer, 17–31.

LORD, P. W., BECHHOFER, S., WILKINSON, M. D., SCHILTZ, G., GESSLER, D., HULL, D., GOBLE, C. A., AND STEIN, L. 2004. Applying semantic Web services to bioinformatics: Experiences gained, lessons learned. In *Proceedings of the 3rd International Semantic Web Conference*. Springer, 350–364.

LUDWIG, S. A. AND REYHANI, S. M. S. 2006. Semantic approach to service discovery in a grid environment. *J. Web Sem. 4,* 1, 1–13.

MAXIMILIEN, E. M. AND SINGH, M. P. 2004. A framework and ontology for dynamic Web services selection. *IEEE Internet Comput. 8,* 5.

McGuinness, D. L. and v. Harmelen, F. 2004. Owl Web ontology language overview. In *W3C Recommendation*.

McIlraith, S., Son, T., and Zeng, H. 2001. Semantic Web services. *IEEE Intell. Syst.* Special Issue on the Semantic Web *16*, 2, 46–53.

Medjahed, B., Bouguettaya, A., and Elmagarmid, A. K. 2003. Composing Web services on the semantic Web. *VLDB J. 12,* 4, 333–351.

Mitchell, T. M. 1997. *Machine Learning*. Mc Graw Hill.

Mitra, P., Wiederhold, G., and Kersten, M. L. 2000. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings of the 7th International Conference on Extending Database Technology (EDBT'00)*. Springer, 86–100.

Nezhad, H. R. M., Benatallah, B., Casati, F., and Toumani, F. 2006. Web services interoperability specifications. *IEEE Computer 39,* 5, 24–32.

Oldham, N., Thomas, C., Sheth, A. P., and Verma, K. 2004. METEOR-S Web service annotation framework with machine learning classification. In *1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC'04)*. Springer, 137–146.

Patil, A. A., Oundhakar, S. A., Sheth, A. P., and Verma, K. 2004. METEOR-S Web service annotation framework. In *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*. ACM, New York, NY, 553–562.

Rahm, E. and Bernstein, P. A. 2001. A survey of approaches to automatic schema matching. *VLDB J. 10,* 4, 334–350.

Senger, M., Rice, P., and Oinn, T. 2003. Soaplab: A unified sesame door to analysis tools. In *UK e-Science All Hands Meeting*. National e-Science Centre. 509–513.

Sirin, E., Parsia, B., Wu, D., Hendler, J. A., and Nau, D. S. 2004. Htn planning for Web service composition using shop2. *J. Web Sem. 1,* 4, 377–396.

Sycara, K. P., Paolucci, M., Ankolekar, A., and Srinivasan, N. 2003. Automated discovery, interaction and composition of semantic Web services. *J. Web Sem. 1,* 1, 27–46.

Traverso, P. and Pistore, M. 2004. Automated composition of semantic Web services into executable processes. In *3rd International Semantic Web Conference*. Springer, 380–394.

Wilkinson, M. 2006. Gbrowse moby: A Web-based browser for biomoby services. *Source Code for Biology and Medicine 1,* 4, 1–8.

Wroe, C., Goble, C. A., Greenwood, R. M., Lord, P. W., Miles, S., Papay, J., Payne, T. R., and Moreau, L. 2004. Automating experiments using semantic data on a bioinformatics grid. *IEEE Intell. Syst. 19,* 1, 48–55.

Wroe, C., Stevens, R., Goble, C. A., Roberts, A., and Greenwood, R. M. 2003. A suite of daml+oil ontologies to describe bioinformatics Web services and data. *Int. J. Cooper. Inform. Syst. 12,* 2, 197–224.

Zhao, J., Wroe, C., Goble, C. A., Stevens, R., Quan, D., and Greenwood, R. M. 2004. Using semantic Web technologies for representing e-science provenance. In *3rd International Semantic Web Conference*. Springer, 92–106.

Received June 2007; revised December 2007; accepted January 2008