Toward Semantics Empowered Biomedical Web Services

Jia Zhang*, Ravi Madduri**, Wei Tan***, Kevin Deichl*, John Alexander*, Ian Foster**
*Department of Computer Science, Northern Illinois University, USA
**University of Chicago and Argonne National Lab, USA
***IBM T.J. Watson Research Center, Hawthorne, NY 10532, USA
*jiazhang@cs.niu.edu, **madduri@mcs.anl.gov, ***wtan@us.ibm.com, **foster@mcs.anl.gov

***Abstract*—caGrid has accumulated a repository of biomedical services; however, how a cancer researcher can find proper services in the caGrid when needed remains a big challenge. This research aims to enhance the cyberinfrastructure of caGrid, by developing a mechanism that turns caGrid services into semantic-aware interoperable services. We proposed a service semantics model, and developed a technique that automatically extracts semantic metadata from static WSDL service descriptions. Such semantic information is stored as loosely coupled annotations that can be queried using semantic Web techniques, to enhance services discovery and composition. We also proposed a two-phase discovery technique that helps users quickly identify interested service operations. This paper also reports our examinations over available techniques and recommends a feasible infrastructure for biomedical service reuse. A prototyping system is developed as a proof of concept.**

I. Introduction

Services computing technique has enabled scientists to expose data and computational resources as Web services. As shown in Fig. 1, scientists create services (data, code, instructions) and publish them on the Internet using the machine understandable Web Service Description Language (WSDL). Other scientists may discover the services and decide whether or which one to use. They may also compose multiple services to create a new experimental process (scientific workflow [1]). If successful, the scientific workflow will be published as a new service for other scientists to reuse. Such a virtual cycle can be envisioned as a new paradigm in science: service-oriented science, or e-Science [2].

Life science is one of the disciplines that pioneer in the trend of e-Science. When the National Cancer Institute (NCI) launched the initiative of cancer Biomedical Informatics Grid (caBIG) project, one key strategy was to leverage the services computing technology to connect the entire cancer community to accelerate cancer research.

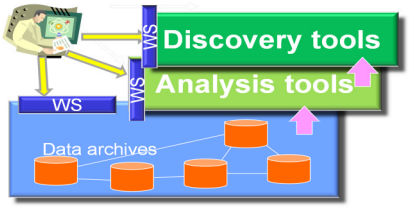

Fig. 1 Service-oriented Science.

To help life scientists publish and discover scientific services, centralized repositories and registries have been established. One known repository is the BioCatalogue [3], which has accumulated more than 1,700 biomedical services to date. The caBIG project has also created caGrid as a service repository. However, it has become a big challenge for life scientists to understand the thousands of available services and select appropriate ones to facilitate their own workflow design. Our recent analytical study revealed that only a small number of utility services at BioCatalogue (e.g., Blast, http://xml.nig.ac.jp/wsdl/Blast.wsdl, an application of comparing genome sequences) are frequently used by life scientists to build scientific workflows [4].

Our survey from various caBIG projects exposed three reasons that may lead to this phenomenon. First, life scientists are not aware of the available services. Second, life scientists struggle with how to operate a service (e.g., to conform to its data and operation formats). Third, life scientists are unable to gather all relevant knowledge to best understand the services.

NCI thus formed a project "Semantic Workflows," aiming to explore a way to facilitate life scientists in building workflows from reusable services. The Center for Biomedical Informatics and Information Technology (CBIIT) at NCI decided to adopt the Healthcare Level 7 (HL7) Services Aware Interoperability Framework (SAIF) [5] to enable the development of domain software components as Working Interoperability (WI).

Toward this ultimate goal, our first step is to investigate to apply Semantic Web technology [6] to enhance artifact discovery and composition. Here the term artifact refers to either a service or a workflow. Our rationale is as follows. 1) The existing artifact publication techniques (e.g., WSDL files) only provide syntactical invocation interfaces for the artifacts. 2) Semantic information, referring to the meaning of data and functions, should help scientists better understand available artifacts (e.g., what are the best ways to use them; are there any constraints, etc.), thus help scientists better leverage existing artifacts. 3) The Semantic Web community has created a wealth of methods, standards, and technologies to create, store, analyze, and query machine-understandable meta-data (semantics) over Web information [6]. 4) Therefore, we shall investigate how to leverage the Semantic Web technologies to improve artifact discovery and composition, in the context of caBIG and life science research. It should be noted that our research approaches may be easily expanded to other areas not

978-0-7695-4463-2/11 $26.00 © 2011 IEEE
DOI 10.1109/ICWS.2011.123

limited to caGrid and life science.

This paper reports our on-going efforts of automatically extracting semantic information from available biomedical services toward making them "computable semantic interoperability" (CSI). Our contributions are four-fold. 1) We proposed a service semantics model and built techniques to automatically extract semantic information from published service documents and annotate services. 2) We established a two-phase search technique that leverages structural and semantic information to quickly identify related operations. 3) We suggest a feasible service semantics extraction and generation infrastructure compatible with existing standards and techniques. 4) We built a prototyping service search engine as a plugin to a known scientific workflow system.

The remainder of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we introduce our service semantics model and an automatic semantics extraction technique. In Section 4, we present our two-phase services discovery technique. In Section 5, we present our proposed system infrastructure and implementation details. In Section 6, we present experiments and discussions. In Section 7, we make conclusions.

II. Related Work

Segev and Zheng [7] propose an ontology bootstrapping method that automatically generates concepts and their relations in a domain from WSDL files. In contrast to their work, we focus on clustering services and service operations to facilitate services discovery.

Semantic Automated Discovery and Integration (SADI) [8] framework is able to recommend SADI services based on input or output data types. However, a SADI service has to be created manually, built and deployed as a servlet, and then registered to the SADI registry, before it can be searched by the SADI search engine. In contrast, our search engine can cover normal WSDL services. Furthermore, while SADI search only compares the input/output OWL class URLs in registered SADI services, our work considers more semantic conditions (e.g., functional profile, pre- and post-conditions, and constraints).

Zhang and Li introduce the concept of service cluster [9] to represent a collection of available services provided by multiple service providers to perform a specific common function. Here we use the concept to represent a collection of functionally relevant services based on domain-specific ontology. We use clustering algorithms to automatically identify service clusters from their published documents.

The WINGS [10] project adopts AI planning and semantic reasoners to verify whether a workflow complies with the requirements of the comprising components and datasets. The Kepler [11] workflow management system provides ontology-driven search capability for data and actors that have been annotated with formal ontologies. In contrast, we focus on using automatically extracted semantic metadata to enhance services discovery.

There have been a lot of efforts on semantic services discovery, most of which performing profile-based service signature (I/O) matching [12]. OWLS-MX [12] and WSMO-MX [13] propose to combine logic-based reasoning and syntactic concept similarity computations in OWL-S. Sbodio et al. [14] propose to use SPARQL as a formal language to describe the pre- and post-conditions of services. Junghans et al. [15] propose a practical formalism to describe functionalities and service requests. In contrast, we leverage service functional profiles and service operation structures to enhance services discovery.

III. Semantics Model and Automatic Extraction

As shown in Fig. 2, we propose a service semantics model comprising both static semantics and behavioral semantics. Static semantics describe the functionalities that a service promises to provide, as well as the goals of the service. Behavioral semantics describe the required circumstances when a service can behave, including input and output parameters, pre- and post-conditions, constraints, and historical usage patterns.

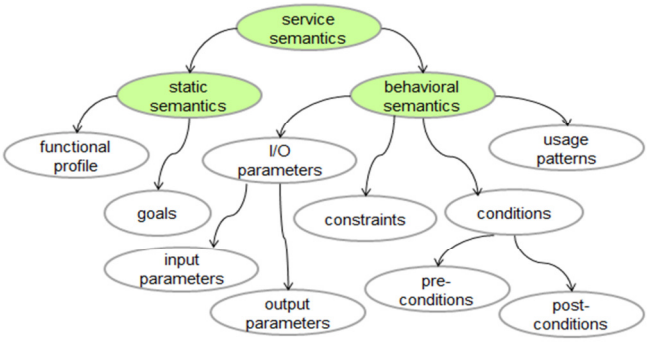

Fig. 2 Service semantics model.

The model defines an ontology that describes the semantics of a service. From service provider perspective, it guides how to depict a service; from consumer perspective, it facilitates effective services discovery. We have developed a technique that can automatically extract various aforementioned semantic metadata from a published WSDL service.

A. *Static semantics*

The goals of a service are usually implied by embedded comments; therefore, we focus on how to extract the functional profile of a service from its WSDL file. Our hypothesis is that, user-defined names used in a WSDL document may depict a functional profile of the corresponding service. The rationale is that, service developers tend to follow naming conventions and use meaningful words to name operations and services. Furthermore, when an IDE automatically generates a WSDL document from source code where naming convention is typically strictly enforced, the actual method names will be used for generating corresponding WSDL interfaces. As shown by the following example, a method named "add" in a

Java class Calculator will cause all related WSDL segments to be named after it (WSDL 1.1 generated by Java Eclipse IDE), including portType name, operation name, input/output message names, and part names inside of input/output messages.

```
<wsdl:message name="addResponse">
  <wsdl:part element="impl:addResponse" name="parameters"/>
</wsdl:message>

<wsdl:message name="addRequest">
  <wsdl:part element="impl:add" name="parameters"/>
</wsdl:message>

<wsdl:portType name="Calculator">
  <wsdl:operation name="add">
    <wsdl:input message="impl:addRequest" name="addRequest"/>
    <wsdl:output message="impl:addResponse" name="addResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

Therefore, we define the functional profile of a service *s* as follows. It is a combination of service name, portType name and all comprising operations' information, each in turn including the name of the operation, names of the input/output messages and names of all comprising parts of the messages.

```
spf(s) ={service_name U
            portType_name U
            [ operation_name U
            input_message_name, [part_name]* U
            ouput_message_name, [part_name]* ]*}
```

We thus obtain a service functional profile as a document containing all the names extracted from its WSDL file. The resulting profile may comprise duplicated words.

B. Behavioral semantics

Our hypothesis is that, a service, especially a scientific service, may require a suitable execution context for the best result. For example, our past analysis found that the *performKNN*[1] service (a machine learning method *K-Nearest Neighbor*) usually performs well when integrated with pre-processing services provided by the same research group[2]. Therefore, we highlight the importance of extracting behavioral semantics of a published service. Such metadata shall benefit the precision level of services discovery, because they can be used to compare with service requestors' residing contexts [16]. In other words, by automatically extracting behavioral semantics of published services and advertising their expected behaviors, we escalate the interoperability of services.

As shown in Fig. 2, we identify four categories of behavioral semantics. Our categorization is compatible with the HL7 Behavioral Framework Metamodel [5] that defines the metadata needed for workflows in NCI CBIIT.

I/O parameters refer to input and output data types. At the current time, we consider exact matching of data types defined in WSDL files or referred XML Schema files. According to W3C WSDL specifications, each input or output parameter is an XML-compatible data type, either an XML built-in type or a user-defined compound type based on built-in types.

Our earlier study on automatic service test case generation [17] pointed out that constraining facets can be used for service providers to specify data constraints, each restricting aspects of the value space of a data type. The following code segment describes a user-defined data type containing two integer-type elements. The first element leverages constraining facets to delimit the value range of 0 to 100, inclusive. The second element requires non-negative integer numbers.

```
<element name="add">
 <complexType>
  <sequence>
   <element minInclusive="0" maxInclusive="100" name="i"
type="xsd:int"/>
    <element minInclusive="0" name="j" type="xsd:int"/>
  </sequence>
 </complexType>
```

According to XML Schema, all 19 built-in primitive types and 27 built-in derived types can each be associated with a set of constraining facets. Our study yielded a matrix of allowable constraining facets for all XML built-in types [17]. In this project, we leverage our earlier work and search for constraints defined by service providers in the format of constraining facets in WSDL files.

Our another earlier study revealed that historical usage data may be leveraged to increase the effectiveness and efficiency of services discovery [18]. For example, assume that historical data shows two services (s_1 and s_2) are always used together in common workflows. This knowledge indicates that, if a scientist selects one such service (say s_1), the other service should be recommended to the scientist. The usage pattern metadata intends to record such best practice from past experiences. Our technique of automatically mining services-oriented past usage patterns is reported in another paper [4].

Note that behavioral semantics may help to enable the computationally assisted assembly of services using business-level guidance in addition to technical-level guidance (e.g., syntactical service match making). As the first step, here we focus on developing a notion of "just enough semantics."

C. Semantic annotations

Based on our proposed service semantics model, all metadata will be automatically generated from service sources (WSDLs). We record such semantic metadata in the form of annotations to further facilitate services discovery.

We face two options to store annotations for a service: a tightly coupled way or a loosely coupled one. The first option is to insert annotations into the original WSDL file. The second option is to store annotations in a separate document, while inserting a link into the original WSDL file to refer to the annotation document. Each option has a W3C

[1] KNN service: http://node255.broadinstitute.org:6060/wsrf/services/cagrid/KNN?wsdl
[2] broadinstitute, www.broadinstitute.org

project supporting it: SPARQL Annotations in WSDL (SPDL) project [19] for the former and Web Service Modeling Ontology (WSMO) [20] for the latter.

SPDL chooses to insert SPARQL Annotations (SPAT) into WSDL files, as illustrated by the example below.

```
<xs:sequence>
    <xs:element name="Keywords" type="xs:string" minOccurs="0"/>
    <xs:element name="SearchIndex" type="xs:string" minOccurs="0"/>
     <!-- xs:annotation><xs:appinfo><spat:SPAT>
       PATHPATTERN { ?req tns:keywords xpath("tns:Keywords") ;
                     tns:index    xpath("tns:SearchIndex") }
     </spat:SPAT></xs:appinfo></xs:annotation -->
</xs:sequence>
```

This example includes two triples patterns, associating the objects of the triples with some information accessible by the XPaths /tns:Keywords and /tns:SearchIndex. RDF applications can then conduct SPARQL queries over annotated Web services.

In contrast, WSMO introduces Semantic Annotations for WSDL (SAWSDL) for inserting annotation reference links. The following example shows that an annotation file "elementRef" can be accessed by the URI, using Web Service Modeling Language (WSML), a SAWSDL extension:

```
    <xs:schema elementFormDefault="qualified"
targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#">
       <xs:element name="add"
sawsdl:modelReference="http://www.example-one.org#elementRef">
```

We consider a loose coupling solution is important to offer higher reusability and flexibility in this project. Especially, Web services are hosted and maintained by corresponding service providers. As shown in Fig. 3, the WSMO approach allows us to keep the registry of our search engine clean, since we do not have to change the WSML-annotated WSDL files when annotations get changed, e.g., new usage patterns are identified and recorded into the annotation file of a service.

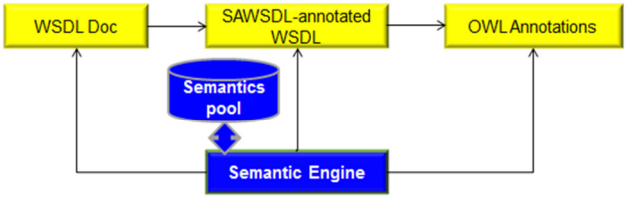

Fig. 3 Loosely coupled semantic annotation.

IV. Two-Phase Services Discovery

We propose a two-phase technique for services discovery at the search engine. A published service usually offers a set of operations, providing similar functionalities but serving slightly different scenarios. For instance, two operations in a service may offer exactly the same functionality but require different numbers of input arguments or different input data types. This observation is extremely important for biomedical services discovery. Consider a life scientist who holds some KEGG data at hand, and intends to find an available service that can take the KEGG data type as input to conduct data analysis. In such a circumstance, we aim to identify proper operations, instead of stopping at the service level.

Meanwhile, performance of services discovery remains a significant issue because of the size of the search space. For example, the BioCatalogue repository has accumulated over 1,700 biomedical services.

We thus developed a two-phase approach. In phase one, we aim at quickly locating a group of related services (a service cluster comprising relevant services from the original large-scale service set) to largely narrow down the search space. In phase two, we aim at finding proper operations inside of the service cluster based on semantic context.

A. *Service clustering*

To find a proper service cluster in a service repository, we apply the information retrieval technology to divide services into clusters based on their functional similarities. In contrast to the existing service categorization approaches that depend on the keywords provided by service providers, our approach aims to automatically divide services into clusters based on their static semantics (i.e., functional profiles) that can be automatically extracted from their published files. The following pseudo code shows how we cluster services in a registry:

Algorithm: Cluster services
Input: a collection of services registered SS.
Output: a collection of service clusters CS.
1: build_corpus(collection of services)
2: $CS \leftarrow \emptyset$
3: **For** each service $s_i \in SS$
4: $normalize(sfp(s_i))$
4.1: $decompose(sfp(s_i))$
4.2: $poster\_stemming(sfp(s_i))$
5: $weighting(sfp(s_i))$
5.1: $sfp\_new \leftarrow \emptyset$
5.2: **For** each $w_{j,i} \in sfp(s_i)$
5.3: **if** $w_{j,i} \notin sfp\_new$ **then**
5.4: $tf\_idf(w_{j,i}) = tf_{j,i} \times idf_j$
5.5: add$(w_{j,i}, tf_idf(w_{j,i})) \rightarrow sfp_new$
5.6: **endif**
5.7: **EndFor**
5.8: $sfp\_new \rightarrow sfp(s_i)$
6: **EndFor**
7: **For** each pair of services $s_1, s_2 \in SS$
8: $sim(s_1, s_2) \leftarrow sim(sfp(s_1), sfp(s_2))$
9: **EndFor**

In step 1, we gather all terms (normalized names) from all services and establish the corpus $(\bigcup_{j=1}^{|S|} \forall t_i \in s_j)$. The original service cluster set is set to be empty in step 2.

We use their functional profiles to evaluate the similarities among services. As the process shown in Section III.A, the functional profile of a service is constructed as a document containing a list of names (terms).

Each name obtained has to go through a normalization and a weighting process before it can be used for further analysis, as shown in steps 3-6. A name is an identifier that is either a single English term or a composite one. For

example, the name of an operation may be preProcessData or get_Array_Data. Such a composite term can be divided into single terms: by identifying big-case characters and removing separators for the above two situations, respectively. Another issue is the synonym issue. For the same meaning, one may choose to use syntactical variants, such as plurals, past tense suffixes, and gerund forms. We chose to partially solve this problem by substituting names with their respective stems, the portion of a word left after the removal of its affixes. We adopted the Porter stemming algorithm [21] for prefix and affix removal and Wordnet (http://www.wordnet.princeton.edu) for solving the synonym issue. Going through all the names in a functional profile of the service, we get a normalized document.

In step 5, by applying the term frequency–inverse document frequency (TF-IDF) algorithm [22], we calculate the weight ($w_{j,i}$) of every term ($t_{j,i}$) inside of the functional profile of every service (s_i). It indicates the importance of the term representing the semantics of the service.

$$tf\_rdf(w_{j,i}) = tf_{j,i} \times idf_j = \frac{C_{j,i}}{\sum_k c_{k,i}} \times lg \frac{|S|}{1 + |\{s: t_{j,i} \in sfp(s)\}|}$$

Where $tf_{j,i}$ is the term frequency (obtained by the number of occurrences of the term $t_{j,i}$ in the functional profile of service s_i, divided by the size of the functional profile of the service to ensure that $tf_{j,i} \in [0,1]$); idf_j *is the* general importance of the term (obtained by dividing the total number of services by the number of services containing the term in their functional profiles, and taking the logarithm of the resulting quotient). Meanwhile, duplicated terms will be removed. In other words, the result will be a map, each key being a term and corresponding value being its importance that represents the functionality of the service.

Although the concept relatedness between terms is broader than that of similarity [23], we only consider the latter for simplicity. We then leveraged the shortest path-based LCH algorithm [24] to measure the semantic similarity between two terms based on the lexical database WordNet.

$$sim(t_i: ST, t_j: ST) = lch(t_i, t_j)$$

Now we are ready to calculate the similarity between two lists of elements (terms) in step 8. Two lists can be abstracted as two disjoint sets of elements X and Y: $X = \{x_1, x_2, \dots x_m\}, Y = \{y_1, y_2, \dots y_n\}, m \geq 1, n \geq 1.$ $\forall x_i \in X, y_j \in Y,$ an edge always exists $(x_i, y_j) \in E$, with a weight of $sim(x_i, y_j)$. Thus, we obtain a weighted complete bipartite graph $G = (V = (X, Y), E)$. Calculating the similarity between the two lists is therefore turned into finding a perfect match (maximum cardinality matching), where the sum of the weights of the edges in the matching reaches a maximal value:

$$sim(sfp(s_1), sfp(s_2)) = \max\left(\sum_{k=1}^{\min(|X|,|Y|)} sim(x_{M1(k)}, y_{M2(k)}) \cdot tf\_idf(x_{M1(k)}) \cdot tf\_idf(y_{M2(k)}) \right)$$

where M1 and M2 are two mapping functions, each selecting the number of min(|X|,|Y|) elements from the sets X and Y, respectively:

$$M1: k \rightarrow x_{M1(k)}, \forall k1 \neq k2, M1(k1) \neq M1(k2);$$
$$M2: k \rightarrow y_{M2(k)}, \forall k1 \neq k2, M2(k1) \neq M2(k2);$$

We applied the Hungarian algorithm [25] to find the optimal match, with a cost of $O(V^2E)$.

We calculate the similarity factor over functional profiles between each pair of services, and obtain a similarity matrix for all services residing in the registry. If the similarity between a peer satisfies $sim(s_1, s_2 \geq \gamma)$, where γ is a preset value, they are put into the same service cluster. Without losing generality, we set γ as 0.75 and apply to the registry to identify service clusters.

B. Service operation clustering

We leveraged the behavioral semantics to cluster service operations in a service cluster. Right now we used the internal structure of service operations to cluster them based on their similarity. Fig. 4 illustrates the building blocks of a service operation as well as the relationships between them, using a UML class diagram. An operation comprises one input message and one output message. Each message may contain multiple parts, each comprising an attribute representing its data type that can be either an XML built-in type or a user defined type (a complex type that is defined recursively). As shown in Fig. 4, each comprising element contains an attribute declaring the name of the element.

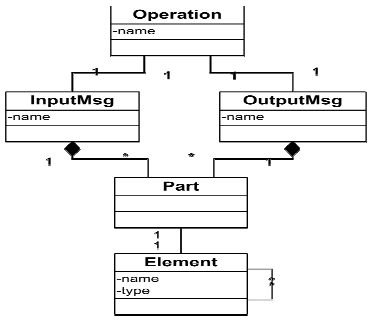

Fig. 4 Structure of a service operation.

Based on the structure of a service operation, we have developed a similarity computation algorithm for comparing two operations. Note that we only need to consider two operations belonging to two services residing in the same service cluster. Given two operations $\{ o_i, o_j \in sc_k \}$, their similarity can be calculated using the following formula.

$$sim(o_i: OP, o_j: OP) = w_1 \cdot sim(o_i.name: ST[], o_j.name: ST[]) + w_2 \cdot sim(o_i.in: MSG, o_j.in: MSG) + w_3 \cdot sim(o_i.out: MSG, o_j.out: MSG)$$

where $w_1+w_2+w_3=1$.

The coefficients indicate that we may assign different weights for operation names and messages, respectively. Each operation name is normalized into a list of terms using our method discussed in Section III.A.

In turn, the following formula compares the similarity between two messages (either input or output messages). The coefficients indicate that we may assign different weights for

message names and comprising parts, respectively.

$$sim(m_i{:}MSG, m_j{:}MSG) = w_1 \cdot sim(m_i.name{:}ST[], m_j.name{:}ST[]) + w_2 \cdot sim(m_i.parts{:}PT[], m_j.parts{:}PT[])$$

where $w_1+w_2=1$.

Each message name is normalized into a list of terms using our method discussed in Section III.A. Each message may contain a list of parts.

Calculating similarity between two lists of terms (names) or two lists of parts is more challenging because the two lists may contain different numbers of elements. We first calculate two parts only.

The following formula compares the similarity between two parts, belonging to two different messages. The coefficients indicate that we may assign different weights for part names and data type, respectively.

$$sim(p_i{:}PT, p_j{:}PT) = w_1 \cdot sim(p_i.name{:}ST[], p_j.name{:}ST[]) + w_2 \cdot sim(p_i.type{:}ST, p_j.type{:}ST)$$

where $w_1+w_2=1$.

For simplicity, here we only compare the names (strings) between data types. Considering containment relationships between data types will be our future work.

We have discussed the algorithm that compares the similarity between two terms in the last section $sim(t_i{:}ST, t_j{:}ST)$. We also have discussed how to calculate the similarity between two lists of names. Calculating two lists of parts is similar, as we formulate the problem as finding a perfect match in a weighted complete bipartite graph $G = (V = (X, Y), E)$, where the sum of the weights of the edges in the matching reaches a maximal value:

$$\max\left(\sum_{k=1}^{\min(|X|,|Y|)} sim(x_{M1(k)}, y_{M2(k)})\right)$$

The only difference is that the importance of each element (i.e., part) here is equal. We use the approach to calculate the similarity between names (operation, message, and part element) and parts. As a result, we can calculate the similarity between any two operations in one service cluster.

Then we apply a hierarchical clustering algorithm [26] to organize all operations in the same service cluster into a multi-level cluster. Each cluster represents an abstract operation providing similar semantic services with a similar structure. Searching in such a hierarchical structure, we can help scientists quickly identify proper service operations.

V. System Infrastructure and Implementation

Fig. 5 illustrates the infrastructure of our semantics-empowered services search engine, and the control flow among its comprising components. A number of open-source libraries are leveraged in our implementation. To make it easier for audience to read, we summarize the libraries in Table 1, with their abbreviations, published groups, and either their full names or short descriptions. We integrate our approach into Taverna workbench [27] (a known workflow management system in life science), and since Taverna is developed in Java, all of our selected open-source libraries are Java-based. Our search engine is implemented as a plug-in to the Taverna workbench where users can conduct SPARQL queries. Upon receiving a query, the search engine will rank available services, through evaluating their WSDL files as well as associated annotation files.

As shown in Fig. 5, WSDL files and their associated annotations (in the format of OWL/RDF) are stored separately for higher flexibility, linked through their intermediate SAWSDL-annotated WSDL files. As a matter of fact, all three parties are stored separately, while the original WSDL files are maintained by the corresponding service providers. The mappings among the three categories of files are maintained by our search engine, and remain to be kept up-to-date by monitoring whether changes are performed on the original WSDL files.

In case that a new WSDL file is published or an existing WSDL file is updated, our system will regenerate its annotations. As shown in Fig. 5, based on whether the service description file is compatible with WSDL 1.1 or WSDL 2.0, one of the two paths will be selected. Although WSDL 2.0 has already been recommended by W3C since June 2007, we found that many available biomedical services remain to be WSDL 1.1 compatible. Furthermore, many existing Java development environments (e.g., Eclipse and NetBeans) only support WSDL 1.1. As a result, we have to

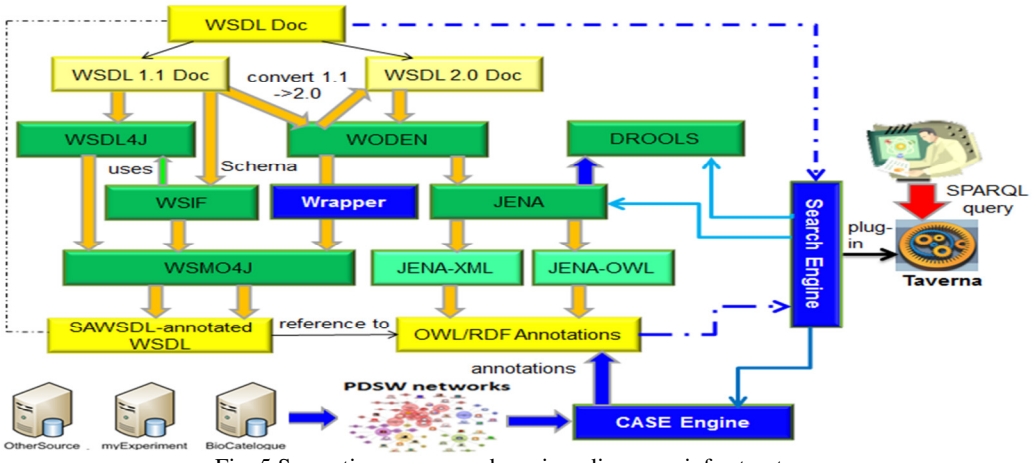

Fig. 5 Semantics-empowered services discovery infrastructure.

Table 1. Summary of open-source libraries used.

Abbr.	Group	Full Names/Descriptions
DROOLS	JBoss	JBoss-embedded rule engine
JENA	Talis, Epimorphics	A framework of building semantic Web applications
JESS	E.Friedman-Hill	rule engine
LUCENE	Apache	text search engine
SAWSDL	W3C	Semantic Annotations for WSDL
WODEN	Apache	reading, manipulating, creating and writing WSDL documents
WSDL4J	IBM	Web Services Description Language for Java
WSIF	Apache	Web Services Invocation Framework
WSMO4J	ESSI	Java API for Web Services Modeling Ontology

support both WSDL 1.1 and 2.0 handling.

Such a situation has an impact on our library selection. Woden was our original choice for parsing WSDL files, which is an Apache project aiming to develop a Java class library for reading, manipulating, creating and writing WSDL documents [28]. However, Woden only supports WSDL 2.0 to date, while Eclipse and other such programs only generate WSDL 1.1 documents. WSDL4J [29] has the ability to parse WSDL 1.1 files. By using WSDL4J, we are able to parse the documents to find available operations and what data types they require for their input and output messages. However, in the case of complex data types, we were unable to determine the requirements as WSDL4J does not support the parsing of the schema, where the complexType definitions are defined.

This challenge led us to adopt Apache's WSIF [30]. The only key component in which we are interested at this point is its schema parser in the package: org.apache.wsif.schema. Unfortunately, the latest version of WSIF is not compatible with the latest version of WSDL4J. In addition, not all the required jar files for WSIF were located in the download and had to be tracked down manually. After finding the required jars, we made the changes to make them coexist (in Wrapper shown in Fig. 5). As a result, as shown in Fig. 5, Woden and WSDL4J/WSIF are leveraged to analyze WSDL 2.0 and 1.1 files, respectively.

The Java objects obtained in memory after parsing a WSDL file are sent to the WSMO4J [31] library to produce SAWSDL[32]-annotated WSDL file. We insert SAWSDL tags into the WSDL file to point to actual annotation files containing semantic information extracted from the WSDL file. We adopt the JENA framework to create semantic annotation files from the Java objects in memory. To date, we have created annotations in two formats: RDF and OWL, leveraging Jena-XML and Jena-OWL, respectively.

Drools [33] from JBoss provides a unified and integrated platform for rules. Comparing to other known rule engines such as JESS [34], users of Drools can describe rules in a more object-oriented manner. We thus adopt Drools to record and validate rules in our service discovery engine.

As shown in Fig. 5, upon receiving a user query, our search engine will consult with JENA to enable SPARQL queries, DROOLS to enable rule evaluations, and our CASE engine [18] to enable run-time historical information checking. CASE engine periodically monitors newly published or updated biomedical workflows and services, updates the PDSW networks, and generates appropriate usage pattern-related semantic information and inserts it into the corresponding annotation files.

We adopted Apache Lucene (http://lucene.apache.org/), an open-source search library to index the information collection and associated annotations. SPARQL queries are hidden from users who may be domain scientist not familiar with the Semantic Web technologies.

VI. Experiments and Discssions

We have applied our approach to the BioCatalogue repository. Over the 1,740 services registered gathered on February 28, 2011, services are clustered as shown below, using our algorithm discussed in Section III.A.

#cluster	1	1	1	1	3	1	2	3	8	15	1140
#service	129	63	12	8	7	6	5	4	3	2	1

One large cluster was identified with 129 services; and one cluster with 63 services. Then the number of services that can be grouped into the same cluster drops significantly. Our investigation found several significant factors. First, service developers do not always follow naming conventions. For example, method name “polljob” is used instead of “pollJobs.” Second, the word base WordNet that we used does not support many domain-specific words. For example, acronyms such as “RNA” or “PNG” are not supported by WordNet. Meanwhile, some application-specific words, such as “Api” and “libs,” are not supported as well. Third, domain-specific synonyms may not be supported by WordNet, for example, “get” and “fetch.” Thus, in our future work, we will explore other dictionary to replace WordNet for a better clustering result.

Here we also report some of our experiences of building the infrastructure illustrated in Fig. 5, regarding Maven, eclipse plugins, svn, and project building. To create a Taverna plugin project in Eclipse, the Apache Maven tool is required as a software project management and comprehension tool (http://maven.apache.org/). To use Maven in eclipse, a user must first install an Eclipse plugin (m2eclipse). The most up-to-date version of m2eclipse is 0.12; however, it cannot create a Taverna plugin with an error message of “remote catalog is empty.” Through trial and error, we found that its earlier version 0.10 works correctly with the Tavern maven repository.

We used one known open-source version control system (Apache Subversion http://subversion.apache.org/) to manage the revisions of the system. We found that checking in a project is easy; however, checking a project out is a different story. One trick is that users must create a new Maven project before checking out a Maven project from Subversion. Meanwhile, the SVN checkout plugin for m2eclipse is needed to check out a Maven project.

As shown in Fig. 5, we used a number of open-source libraries in our implementation. We strived to seamlessly integrate them while eliminating overheads for a higher

performance. Therefore, we intentionally simplified and revised some implementations from the libraries. For example, WSIF's schema parser, located at the package org.apache.wsif.schema.Parser, requires a WSDLLocator (an interface) to read in a WSDL URI. The implementation that is defined in WSIF uses classes that are not necessary for our project (e.g., ClassLoaders); therefore, we implemented our own WSDLLocator.

VIV. Conclusions and Future Work

In this paper, we reported our on-going efforts of building a semantic-aware biomedical services discovery engine. Through automatic sematic metadata extraction from WSDL files and annotations, our approach helps to quickly identify related service operations. Our work also suggests a feasible infrastructure that is compatible with existing standards and techniques. Our prototyping system has demonstrated that our semantic infrastructure is able to fulfill the goal of next-generation caGrid 2.0: make easy things easy, lower the barrier to entry, and support existing users of the present infrastructure.

We plan to continue our research in the following directions. We will explore a notation that formally represents an abstract operation in the operation-level ontology and develop a technique that automatically generates abstract operations in the structure. We will also conduct case studies over real biomedical use cases and evaluate the effectiveness and efficiency of our approach.

X. Acknowledgement

We thank the caBIG Semantic Workflows team for constructive discussions. The work described in this paper is partially supported by the National Cancer Institute, the National Institutes of Health under contract N01-CO-12400, and the National Science Foundation under grant IIS-0959215.

XI. References

[1] B. Ludäscher, "Scientific Workflows: Cyberinfrastructure for e-Science", in Proceedings of *Pacific Neighborhood Consortium (PNC)*, 2007, Berkeley, CA, USA, Oct. 19.
[2] I. Foster, "Service-Oriented Science", *Science*, 2005, 308: pp. 814-817.
[3] J. Bhagat, F. Tanoh, E. Nzuobontane, T. Laurent, J. Orlowski, M. Roos, K. Wolstencroft, S. Aleksejevs, R. Stevens, S. Pettifer, R. Lopez, and C.A. Goble, "BioCatalogue: A Universal Catalogue of Web Services for the Life Sciences", *Nucleic Acids Research*, May 19, 2010, 38: pp. W689-W694.
[4] W. Tan, J. Zhang, and I. Foster, "Network Analysis of Scientific Workflows: a Gateway to Reuse", *IEEE Computer*, Sep., 2010, 43: pp. 54-61.
[5] CBIIT (2010) *HL7 Services Aware Interoperability Framework (SAIF) Implementation Guide*.
[6] B. Medjahed and A. Bouguettaya, "A Multilevel Composability Model for Semantic Web Services", *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Jul., 2005, 17(7): pp. 954-968.
[7] A. Segev and Q.Z. Sheng, "Bootstrapping Ontologies for Web Services", *IEEE Transactions on Services Computing (TSC)*, Oct.-Dec. 2010.
[8] M.D. Wilkinson, L. McCarthy, B. Vandervalk, D. Withers, E. Kawas, and S. Samadian, "SADI, SHARE, and the in silico Scientific Method". 2010, BMC Bioinformatics 11(suppl 12):S7.
[9] L.-J. Zhang and B. Li, "Requirements Driven Dynamic Business Process Composition for Web Services Solutions", *Journal of Grid Computing*, 2004, 2: pp. 121-140.
[10] Y. Gil, V. Ratnakar, J. Kim, P. González-Calero, P. Groth, J. Moody, and E. Deelman, "WINGS: Intelligent Workflow-Based Design of Computational Experiments", *IEEE Intelligent Systems*, Jan./Feb., 2010, 26(1): pp. 62-72.
[11] C.B. Shawn, S. Bowers, M.B. Jones, B. Ludäscher, M. Schildhauer, and J. Tao, "Incorporating Semantics in Scientific Workflow Authoring", in Proceedings of *the 17th International Conference on Scientific and Statistical Database Management (SSDBM)*, 2005, Santa Barbara, CA, USA, Jun. 27-29, pp. 75-78.
[12] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX", in Proceedings of *ACM International Conference on Autonomous Agents*, 2006, Hakodate, Japan, May 8-12, pp. 915-922.
[13] M. Klusch and F. Kaufer, "WSMO-MX: A Hybrid Semantic Web Service Matchmaker", *Web Intelligence and Agent Systems*, Jan., 2009, 7(1): pp. 23-42.
[14] M.L. Sbodio, D. Martin, and C. Moulin, "Discovering Semantic Web Services using SPARQL and Intelligent Agents", *Web Semantics: Science, Services and Agents on the World Wide Web*, Nov., 2010, 8(4): pp. 310-328.
[15] M. Junghans, S. Agarwal, and R. Studer, "Towards Practical Semantic Web Service Discovery", *Lecture Notes in Computer Science (The Semantic Web: Research and Applications)*, 2010, 6089/2010: pp. 15-29.
[16] S.J.H. Yang, J. Zhang, and I.Y.L. Chen, "Ubiquitous Provision of Context-Aware Web Services", *International Journal of Web Services Research (JWSR)*, Oct.-Dec., 2007, 4(4): pp. 83-103.
[17] J. Zhang, "A Mobile Agent-Based Tool Supporting Web Services Testing", *Wireless Personal Communications (WPC), Springer*, 56(1), Jan., 2010, pp. 147-172.
[18] J. Zhang, W. Tan, and J. Alexander, "CASE: A Taverna-Based Recommendation Engine Supporting Services-Oriented Workflow Reuse", Technical Report of Google Summer of Code, 2010.
[19] W3C, "SPARQL Annotations in WSDL (SPDL) Project", Available from: http://www.w3.org/2005/11/SPDL/.
[20] ESSI, "Web Services Modeling Ontology (WSMO)", Available from: http://www.wsmo.org/.
[21] M. Porter, "An Algorithm for Suffix Stripping Program", *Automated Library and Information Systems*, 1980, 14(3): pp. 130-137.
[22] K.S. Jones, "A Statistical Interpretation of Term Specificity and Its Application in Retrieval", *Journal of Documentation*, 1972, 28(1): pp. 11-21.
[23] A. Budanitsky and G. Hirst, "Evaluating WordNet-Based Measures of Lexical Semantic Relatedness", *Computational Linguistics*, Mar., 2006, 32(1).
[24] C. Leacock and M. Chodorow, "Combining Local Context and WordNet Similarity for Word Sense Identification", in *WordNet: An Electronic Lexical Database*, C. Fellbaum, Editor, 1998, MIT Press, pp. 265–283.
[25] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows*, 1993: Prentice Hall.
[26] R. D'andrade, "U-Statistic Hierarchical Clustering", *Psychometrika*, 1978, 4: pp. 58-67.
[27] T. Oinn, M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M.R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe, "Taverna: Lessons in Creating a Workflow Environment for the Life Sciences", *Concurrency and Computation: Practice & Experience*, 2006, 18(10): pp. 1067–1100.
[28] Apache, "Woden", Available from: http://ws.apache.org/woden/.
[29] "WSDL4J", 2005, Available from: http://wsdl4j.sourceforge.net/.
[30] Apache, "Web Services Invocation Framework", Available from: http://ws.apache.org/wsif/.
[31] ESSI, "WSMO4J", Available from: http://wsmo4j.sourceforge.net/.
[32] W3C, "Semantic Annotations for WSDL (SAWSDL)", Available from: http://www.w3.org/2002/ws/sawsdl/.
[33] JBoss, "Drools", Available from: http://www.jboss.org/drools.
[34] E. Friedman-Hill, "JESS", Available from: http://www.jessrules.com/.