SADI Semantic Web Services – 'cause you can't always GET what you want!

Mark D Wilkinson, Benjamin Vandervalk, Luke McCarthy
Department of Medical Genetics, Heart + Lung Institute at St. Paul's Hospital, University of British Columbia, Vancouver, BC, Canada.

Abstract

SADI – Semantic Automated Discovery and Integration – is a set of standards-compliant Semantic Web Service design patterns that exploit the relatively straightforward interfaces exposed by most Bioinformatics Services to simplify and partially automate Service design and deployment. The SADI design explicitly exposes an important Service feature – the semantic relationship between input and output data. SADI Services consume and produce instances of OWL Classes, where the service's function is to add properties onto the input Class until it fulfills the class-membership requirements of the output Class. Indexing Services based on the properties they add enables discovery of Services that generate the biological features of interest relative to a piece of in-hand data. We show that this design pattern can be used to create a client application with strikingly rich semantic behaviors, such as automated discovery of distributed data resources and the automated orchestration of chains of Web Services into complex workflows.

1. Introduction

Two Web technologies - Web Services and the Semantic Web – have the promise to achieve integration and interoperability among the currently disparate bioinformatics resources on the Web. Web Services attempt to tightly define Web interfaces through a machine-readable specification called Web Services Description Language (WSDL) and the use of XML Schema to describe the structure of the input and output messages. Provided that sufficient information is known about the intent of relevant XML elements in the Schema, Services can be chained together into workflows, and workflow design is most commonly done manually ahead of time in an environment such as Taverna [1]. This lack of shared "semantics" around XML elements means that the promise of *automated* Web Service interoperability and workflow construction is only truly successful within well-defined, often project-specific situations.

"Semantics" is also at the core of the emergent Semantic Web initiative [2]. The Semantic Web can be thought of as a graph in which the nodes are anything that can be named (a concept, a document, a person) and the labeled edges are meaningful properties that describe the relationships between the nodes. Resource Description Framework (RDF) [3] is a way of encoding these nodes and labeled edges such that they can be explored and traversed by machines, and most of the data on the Semantic Web is currently stored in RDF documents made available by HTTP GET, or in "triple-stores", which are the RDF equivalent of relational databases. All nodes and properties in RDF datasets are referenced by globally unique identifiers (URIs), and thus the encoding provided by RDF is precise, unambiguous, and ideally suited for automated processing by software. Moreover, it is possible to use logical reasoning to derive new facts which are not explicitly stated in the data. Description logics (DL) are typically employed for this purpose due to their improved computational characteristics in comparison to first order logic, and OWL [4] is the family of description logics (and associated XML syntax) that has been developed for use with the Semantic Web.

It remains problematic, however, that the majority of the data in bioinformatics is in the "Deep Web" - hidden behind query interfaces or existing only as the transient output from invocations of analytical tools. Thus, the majority of data the biologist requires cannot be published on the current "GET-able" Semantic Web. The limited interoperability between archetypal Web Services has spawned a number of "Semantic Web Service" (SWS) initiatives in the Life Sciences that utilize Semantic Web technology in an attempt to enhance Web Service interoperability: TAMBIS [5] was a mediator system in which wrappers around biological resources were mapped to an overarching ontology of bioinformatics; thus the semantics of a service were outside of the service itself. *my*Grid [6] used an extensive bioinformatics domain ontology to annotate traditional Web Services within a formal Web Service annotation/discovery model ("Feta" [7]), thus enhancing the discovery-accuracy by placing semantics in the service interface description. caBIO (part of caCORE [8]) designed a traditional Web Service API describing all "valid" operations for a given set of biological objects, and required providers to consume and produce annotations within their data using a (vast) common vocabulary, thus the semantics were inside the message, but not part of the message structure; Finally, both BioMoby [9] and SSWAP [10] push their semantics into the input and output messages themselves. BioMoby requires service providers utilize a common ontology of biological data-types, and to consume and produce "boutique" XML serializations of instances of that ontology. SSWAP utilizes a shared,

lightweight OWL model of a service interface, where RDF-XML instances of this model are used as both the interface definition and as the "container" of the input and output data during service invocation. Both BioMoby and SSWAP exhibit highly integrative behaviors by virtue of placing their semantics directly in their messages, and both have chosen to discard XML-Schema in favour of ontology-based message structures. Unfortunately, both BioMoby and SSWAP share the limitation of having invented a messaging "scaffold" for their service invocation and response messages, thus forcing clients to be framework-specific. In addition, neither project uses the full range of standard HTTP request/response elements to handle common issues such as asynchronous service invocations. Given these successes and limitations, we have attempted to construct an SWS framework that integrates itself more naturally into the Semantic Web, and into the Web itself, and yet retains the rich semantic Service discovery behaviors of BioMoby and SSWAP.

We, and others [11] have noted that Web Services in bioinformatics (and other scientific domains also) share distinct subset of behaviours compared to the full spectrum of behaviours allowed by Web Services in general. With few exceptions, bioinformatics Web Services are independent, idempotent, stateless, transformative, and atomic. This stands in stark contrast to Web Service solutions to, for example, the ticket-ordering use-case that is commonly discussed in this domain. Almost invariably, bioinformatics Web Services consume some specific input data type, and in a stateless and atomic operation, return related output data type(s) generated by whatever operation the Service executes. Given that these services are trasformative (i.e., that the output is generated by some transformation on the input, and is therefore related to that input) we also note that our end-user community does not usually have a "process model" or "business model" in-mind when searching for a Service. For example, biologists do not execute a BLAST analysis because they wish to run a sequence similarity matrix over their input data; they execute a BLAST analysis because they are interested in finding sequences that are related to their input sequence by homology. They are interested in the *relationships* between the input and output data. From these two observations, we contend that by acknowledging these characteristic features, and designing a standards-compliant interoperability framework that exploits them, we can improve on previous efforts in bioinformatics SWS interoperability, and more tightly and naturally integrate Web Services into the Semantic Web.

Here we present SADI - Semantic Automated Discovery and Integration - a novel SWS framework, and reference implementation, that utilizes Semantic Web standards at all levels of the Web Services "stack", including discovery, messaging, and service description. SADI does not propose any new technologies, standards, messaging formats or structures, metadata structures, result codes, or unusual Web behaviors. SADI simply comprises a set of standards-compliant conventions and suggested best-practices for data representation and exchange between Web Services that fully utilizes Semantic Web technologies. The key novel feature of SADI is that the input and output of a Web Service *must* share a common "base" identifier – effectively, all services are "annotator services", where the input becomes decorated by additional information before being returned to the client. The consequence of this simple constraint is that the semantics of the service operations become extremely transparent. We wish to show, through discussion of several demonstrative implementations of the SADI conventions, that: (a) these conventions enable novel data discovery, interoperability, and integrative behaviors that we believe closely mirror the needs and expectations of our end-user community; (b) that SADI Web Services are compatible with most or all existing Web Service registry systems; (c) that SADI Services are compatible with existing SWS annotation frameworks such as SAWSDL[12]; and (d) through demonstration of a prototype SADI client, that implementing these conventions and practices enables data dynamically derived from Web Services to be exposed in a standard, Semantic Web-compatible way.

2. Implementation

SADI consists of several standards-compliant recommendations for how services could be implemented and described in order to achieve a set of useful, interoperable behaviors. In addition, we have created supporting software and several reference implementations that demonstrate various aspects of the behaviors that are made possible by these recommendations.

The requirements are:

- SADI Web Services are stateless and atomic.
- SADI Services consume and produce data via simple HTTP POST and GET.
- SADI Services consume and produce data in RDF format. This allows SADI Services to exploit existing OWL reasoners and SPARQL query engines to enhance interoperability between Services and the interpretation of the data being passed between them.
- Service interfaces (i.e., inputs and outputs) are defined in terms of OWL-DL classes; the property restrictions on these OWL classes define what specific data elements are required by the Service and what data will be provided by the Service, respectively.

- Input RDF data - data that is compliant with (i.e. classifies into) the input OWL class - is "decorated" or "annotated" by the service provider to include new properties. These properties will (of course) be a function of the lookup/analytical operations performed by the Web Service. The output RDF data is an instance of the OWL class that defines the output of the service.

Importantly, discovery of SADI Services can include searches for the properties the user wants to add to their data. This contrasts with other Semantic Web Service standards which attempt only to define the computational process by which input data is analyzed, rather than the novel data properties that process generates. This is key to the semantic discovery behaviors of SADI.

While these proposals can be implemented in a wide variety of ways, the general way we anticipate this framework will be used for discovery and invocation of services is as follows:

Service Discovery

Discovery of services is possible through a wide variety of parameters. In our current implementation, SADI Services are represented as Feta-compliant OWL/RDF individuals of the class myGrid-Moby-Service [13], which details a variety of information including the input and output data-types, service provider, service operations, and a human-readable description of the service. For the purposes of our own exploration and experimentation with the system, we have constructed a prototype Web Service registry in which SADI services are indexed by their input and output data types, as well as the properties (predicates) that are added to the input node by the service; however, since the service interface descriptions are simply OWL/RDF documents published on the Web, and thus accessible to Web search engines, the requirement for a SADI-specific registry will diminish as search engines begin to index Semantic Web data. Moreover, we do not suggest that this prototype registry is comparable to, or better than, any existing SWS registry system.

In the most naive case, data/service matchmaking could be done by simple string searches; however, since SADI services are expected to natively consume and generate RDF-formatted data, service discovery is better-achieved through a novel reasoner-based approach. To discover a set of services capable of operating on the data the client wishes to analyze, one can retrieve the OWL class definitions of the service inputs, and use these to reason over the in-hand data. Thus, appropriate data becomes "typed" (in the formal-classification sense 'rdf:type') as being a valid input to a service, and in this way we achieve match-making between in-hand data, and services capable of operating on it. Importantly, the service interface definitions can, and hopefully will, define their interfaces using OWL property restrictions rather than simply asserting a specific OWL Class. Defining the *data properties* the service requires, rather than specifying a specific data type, allows more flexibility in the matchmaking between a client's triple-store and a potential service provider because it is not possible to anticipate (a) how the client or any other service classifies/types its data, or (b) the combinations of properties that might be attached to a particular piece of data after it has passed through several distinct services. An additional positive consequence of this approach is that SADI is not reliant on all participants agreeing on a particular data-typing system (i.e. ontology), which is a potential bottleneck of the Semantic Web initiative in general. Since the reasoner can dynamically determine if the selected data is of a "type" that the service provider can operate on, so long as there is general agreement in the community on the meaning and usage of predicates, data from any classification system can be "matched" to the class definition of any given service provider using reasoning. That SADI avoids having to construct a "grand unifying ontology", but rather expects service provider's ontologies to contain just one class (i.e., the input class of the service defined by its properties), provides extreme scalability and avoids the logical errors that are common in large-scale ontology building efforts.

Service Invocation

Once a service is discovered, its end-point is determined and RDF data is passed to that endpoint through a simple HTTP POST. Due to the limited ability of most programming languages to access DL reasoners, we suggest that the data passed to a service should be pre-classified according to the input class definition of that service provider - this allows service providers to easily identify which node(s) from the incoming document represents the intended input based on its rdf:type, without having to reason over that input and without requiring an explicit location in the message scaffold such as in SSWAP. Thus SADI services can be easily written in most popular programming languages, even those that don't yet have rich support for Semantic Web technologies such as Perl.

The response from a service invocation is an RDF document containing the output data, typed according to the output OWL class in the service interface definition. In most cases, the output will consist of new properties attached to the input nodes, where those properties are described in the service interface definition. In this way, the service can be thought of as "annotating" or "decorating" its input with new properties derived by its analysis of the input data. Bulk-invocations of services are achieved by sending more than one appropriately-typed input node in the POST to

the service, and the output will contain a corresponding set of appropriately-typed output nodes. The association between each input and its corresponding output is achieved by comparing URI's, where the URI of the input node is identical to the URI of the output node. Again, this removes the requirement for such information to be encoded by explicit message-structure elements such as is the case in BioMoby and SSWAP.

Both synchronous and asynchronous services are easily supported by this framework. In the case of a synchronous service, the entire RDF graph is returned in the response message. In the case of an asynchronous service, the typed input URIs are returned as with the synchronous service, however these are joined to a temporary service-specific URI by the 'rdfs:isDefinedBy' predicate, in compliance with the defined usage of this predicate [14]. These service specific URIs, when resolved by GET, either return the output graphs (if the service operations are complete) or return an HTTP 202 code ("Accepted but incomplete" [15]). In future implementations, a Web Services Resource Framework [16] reference will likely be passed in the HTTP headers of the response containing information about the state of the asynchronous service to assist clients in determining when an output graph will be available. Supplementary information showing example message structures is provided at [17]. In keeping with our longstanding recognition of the importance of asynchronous service invocation within the BioMoby SWS framework, support for asynchronous services has been a high priority in the design of SADI.

3. Results and discussion

To discuss the consequences of the SADI framework, it is perhaps most informative to observe the behaviors of two reference implementations. These provide two distinct "real world" examples of how such middleware might be used, and demonstrate some of the novel integrative behaviors that are enabled by the SADI approach.

The most straightforward example of usage comes from the SADI plug-in to the Sentient Knowledge Explorer™, a commercial data exploration and visualization system from IO Informatics [18]. When data is selected in the Knowledge Explorer, the SADI plug-in provides a list of data properties that can be generated by SADI services known to the registry. This list is obtained by querying the Knowledge Explorer for the data type of the selected data, and querying the prototype SADI registry for services that consume this data type as input. To invoke a service, the user simply selects the properties of interest to them from the list, and the RDF data is POSTed to the SADI Web Services capable of generating those properties from that input data. The response RDF is then added to the graph displayed in the Knowledge Explorer (see screen-shots at [17]). Thus, in a very intuitive way, the end-user is able to select and retrieve the data properties of interest to them via simple menu-selection backed by the SADI registry.

A slightly more complex example of usage is presented by our Semantic Health And Research Environment (SHARE) prototype query system. SHARE connects the SADI middleware to the Pellet SPARQL query engine and DL Reasoner. Predicates presented to Pellet in SPARQL queries are "intercepted" and passed to SADI to be used for Web Service discovery. Output data from the discovered services is added into Pellet's local triplestore and control is handed back to Pellet. In this way, a query-specific triplestore is dynamically generated as a query is being processed; effectively, the database required to answer the question is automatically generated as a result of the question being posed.

For example, consider the SPARQL query in the box below. SHARE intercepts the first query triple [?protein pred:has3DStructure pdb:1mwp] and attempts to find one or more Web Services capable of generating RDF triples with that pattern. SHARE creates SADI registry queries by considering the subject of the triple pattern as the service input and the predicate as the query for what predicate the Web Service adds to its input. However, in this case the subject of the first triple is a variable.

```
PREFIX pred: <http://sadiframework.org/ontologies/predicates.owl#>
PREFIX ont:<http://ontology.dumontierlab.com/>
PREFIX pdb: <http://lsrn.org/PDB:>

SELECT ?protein ?gene ?path
WHERE {
  ?protein pred:has3DStructure pdb:1mwp .
  ?protein pred:isEncodedBy ?gene .
  ?gene ont:isParticipantIn ?path
}
```

SHARE examines the ontology and detects that pred:has3DStructure has an inverse predicate (owl:inverseOf), so SHARE initiates a SADI discovery query for services that can attach the "pred:is3DStructureFor" predicate to Protein Data Bank (PDB) structure identifiers. In response, the SADI registry discovers a service – getSymbolInfo, provided by the cnio.es authority – that returns the relevant protein information about PDB records. This service is invoked and the resulting RDF is stored in the temporary triple-store. In addition to demonstrating dynamic discovery of relevant

Web Services triggered by query predicates, this also shows that we are capable of doing simple predicate reasoning in parallel with SPARQL query resolution.

The next predicate, isEncodedBy, is then used in a SADI query to discover services capable of generating KEGG Genes based on protein identifiers. The convertIdentifier2KEGGId service, hosted at dev.biordf.net is discovered and invoked with the list of protein identifiers discovered from the first invocation. Finally the isParticipantIn predicate is used to discover the getKeggPathwaysByKeggID service, and this is executed using the list of KEGG gene identifiers from the previous service as its input. The resulting RDF graph is integrated into the temporary triple-store, and control is returned to Pellet to allow it to finally resolve the query. A screen-shot showing the result can be viewed at [17].

This second demonstration shows that the SADI system is capable of resolving predicates from *any* ontology – in this case, from our own ontology at *sadiframework.org*, and from the Dumontier Laboratory ontology at *ontology.dumontierlab.com*. All that is required is for a service to register itself as being capable of generating that predicate. Again, this is important for scalability, since it means that the SADI project does not need to define its own ontologies, rather it can take advantage of any well-formed OWL ontology published on the Semantic Web.

We believe that the computational behavior of SHARE, enabled by SADI, is quite striking and unique in three ways. First, the ability to automatically and dynamically discover, access, and integrate relevant data from distributed, non-uniform data-sources using disparate ontologies is one of the key promises of the Semantic Web that, to our knowledge, has never yet been achieved. Second, through SHARE, users are able to execute SPARQL queries over data that might not exist at the time the query is being composed. For example, a great deal of bioinformatics data comes from the output of analytical algorithms. Using SHARE, a query can be posed about results within that output data before the analysis has been run, since discovery and execution of the required SADI service happens as a result of the query being processed. Finally SHARE behaves very much like a workflow enactment system, where the "intent" of the workflow is represented in the SPARQL query, and then is instantiated through the dynamic discovery and orchestration of Web Services at run-time. Again, we believe this to be a behavior unique to this particular client application, and note that these behaviors are made possible simply by circumscribing a set of best-practices within the broader scope of the Semantic Web and Web Service standards.

4. Limitations

SADI suffers from the same limitations that pose barriers to other Web Service and Semantic Web projects [19]. In the Web Service domain, the utility of SADI is entirely dependent on the number of providers who adopt its conventions. Despite the highly desirable behaviors we have demonstrated with our two sample SADI implementations, the extensive tooling available for traditional Web Services and their relative simplicity (i.e. the simplicity of XML versus RDF/OWL) provides a much lower barrier-to-entry than what we propose. Moreover, there are extensive legacy Web Services that are not interoperable (neither with each other, nor with SADI services), and thus there is little benefit to becoming an early-adopter of SADI. To counter this, we have created software libraries that partially automate the process of service construction – in both Perl and Java – similar to the "Dashboard" application for BioMoby [20]; this is possible because the behaviors of SADI services are predictable, and thus the code for SADI services is similarly consistent and predictable. In addition, we believe that the SAWSDL specification [12], together with XML Transformations, will allow us to build semi-automated "wrappers" around traditional Web Services that will make them SADI-compliant (at the expense of a loss in semantic richness versus creating a native SADI service). This work is ongoing.

The Semantic Web aspects of SADI also expose limitations. In particular, success of the SADI architecture (like the success of the Semantic Web itself) will largely depend on widespread re-use of publicly-available and well-defined ontological predicates, and the definition of service inputs in terms of these properties. Unfortunately, the majority of focus in the Semantic Web efforts of the health-care and life science community thus far has been on defining classes, rather than predicates; asserting class-hierarchies without formally defining what properties a member of that class is expected to have, or what distinguishes members of one class from another. We hope, however, that the power we have demonstrated in these prototype implementations provides a sufficiently compelling argument to initiate the evolution of a slightly higher level of Semantic Web complexity in the health-care and life-sciences space.

5. Conclusions

The SADI framework proposes a set of conventions and best-practices, within the scope of accepted standards for Web Services and the Semantic Web, that enable the creation of bioinformatics software with novel interoperable and integrative behaviors. These were derived by closely observing the "nature" of Web Services in the bioinformatics domain, and the use of these services by biologists and informaticians. This subset of standards, we believe, accurately models both the services and the end-user requirements for dynamic and automated discovery of relevant services, automated pipelining of these services, and integration of the resulting data.

6. Availability and Requirements

SADI is an open-source project and its supporting codebase is hosted at Google Code (http://sadi.googlecode.com). The SHARE demonstration is available for public access (http://biordf.net/cardioSHARE/). The SADI Plug-in to the Sentient Knowledge Explorer is not publicly available at this time.

7. Acknowledgements

The SADI and SHARE projects were founded through a special initiatives award from the Heart and Stroke Foundation of British Columbia and Yukon. Additional funding was obtained from Microsoft Research and an operating grant from the Canadian Institutes for Health Research (CIHR). Core laboratory funding is derived through an award from the Natural Sciences and Engineering Research Council of Canada. Funding for the BioMoby project, from which SADI was derived, is through an award from Genome Alberta, in part through Genome Canada. We would like to thank Dr. Stephen Withers for donation of laboratory space in the Center for High Throughput Biology (CHiBi) at the University of British Columbia.

10. References

[1] Oinn T, Addis M, Ferris J, Marvin D, Senger M, Greenwood M, Carver T, Glover K, Pocock MR, Wipat A, Li P: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 2004 20(17):3045-54.

[2] W3C Semantic Web Activity http://www.w3.org/2001/sw/

[3] Resource Description Frameowork (RDF) http://www.w3.org/RDF/

[4] OWLWeb Ontology Language Guide. http://www.w3.org/TR/owl-guide/

[5] Stevens R, Baker P, Bechhofer S, Ng G, Jacoby A, Paton NW, Goble CA, Brass A: TAMBIS:transparent access to multiple bioinformatics information sources. Bioinformatics 2000, 16:184–5.

[6] Stevens RD, Robinson AJ, Goble CA: myGrid: personalised bioinformatics on the information grid. Bioinformatics 2003, 19 Suppl 1:302–4.

[7] Lord P, Alper P, Wroe C, Goble C: Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery . ESWC 2005, LNCS 3532, pp. 17–31, 2005.

[8] Covitz PA, Hartel F, Schaefer C, De Coronado S, Fragoso G, Sahni H, Gustafson S, Buetow KH: caCORE: a common infrastructure for cancer informatics. Bioinformatics 2003, 19:2404–12.

[9] Wilkinson MD, Links M: BioMOBY: an open source biological web services proposal. Briefings in Bioinformatics 2002,3:331–41.

[10] Gessler DDG, Schiltz GS, May GD, Avraham S, Town CD, Grant D, Nelson RT: SSWAP: A Simple Semantic Web Architecture and Protocol for semantic web services. BMC Bioinformatics 2009, 10:309.

[11] Potter S, Aitken S: A Semantic Service Environment: A Case Study in Bioinformatics . ESWC 2005, LNCS 3532, pp. 694-709, 2005.

[12] Semantic Annotations for WSDL http://www.w3.org/2002/ws/sawsdl/

[13] The myGrid Moby Service Ontology http://www.mygrid.org.uk/mygrid-moby-service

[14] RDF Schema http://www.w3.org/TR/rdf-schema/#ch_isdefinedby

[15] HTTP 1.1/Status Code Definitions http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[16] The Web Services Resource Framework specification http://www.oasis-open.org/committees/wsrf/

[17] Supplementary Information for this manuscript: http://sadiframework.org/documentation/

[18] IO-Informatics :: Accelerating Discovery in Life Sciences http://io-informatics.com/

[19] Martin D, Domingue J, Sheth A, Battle S, Sycara K, Fensel D: Semantic Web Services, Part 2. IEEE Intelligent Systems 2007, 22(6):8–15.

[20] The BioMoby Dashboard http://biomoby.open-bio.org/CVS_CONTENT/moby-live/Java/docs/Dashboard.html