Towards Self-Describing Web Services[⋆]

Phillipa Oaks

Centre for Information Technology Innovation - Faculty of Information Technology
Queensland University of Technology
GPO Box 2434, Brisbane, QLD 4001, Australia
{p.oaks}@qut.edu.au

Abstract. Self describing web services is a catchy phrase but it should mean more than having a description based on XML syntax. In this paper we take software engineering requirements for software interfaces and compare them to web service description specifications. The comparison shows that, at present, less information is associated with a web service than we expect for ordinary software.

1 Introduction

> Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web [1].
>
> ... agents provide service descriptions that tell how they can be used to accomplish other agents' *goals* [2].
>
> ... every information dependent resource, including enterprises, information services, application services, and devices, need to become augmented with machine processable descriptions to support the finding, reasoning about (e.g., which service is best), and using (e.g., executing or manipulating) the resource. The idea is that self-descriptions of data and other techniques would allow context-understanding programs to selectively find what users want, or for programs to work on behalf of humans and organizations to make them more scalable, efficient and productive [3].

Web services are a promising new technology for machine to machine interaction across application, enterprise and web boundaries. One of the reasons web services have gained so much interest and support over the last two years is that there are many existing branches of information technology that can be further developed within this new paradigm. Software engineering, components, component integration, distributed programs, grid computing, middleware, reusable

[⋆] This work was supported by the Australian Research Council SPIRT Grant "Self-describing transactions operating in a large, open, heterogeneous, distributed environment" involving QUT, UNSW and GBST Holdings Pty Ltd.

software, databases, knowledge representation, and other areas of computer science can contribute to web services.

The many sources of interest and contribution have led to a diversity of opinions and approaches on what web services can achieve, on what kinds of conceptual models can be used to describe web services, and also diversity in the approaches to implementation. Already there are competing standards and specifications being driven by various interest groups seeking to make a place for themselves in the web services arena. This leads to standards that either describe overlapping concerns, or are not integrated with one another, as was the case for software engineering in the mid 1990's [4].

There are many consequences of this diversity. There is no clear idea or agreement about what web services will be. Some see web services as the "dumb reactive" cousins of "intelligent proactive" agents. In this scenario, web services merely provide functionality to the agent, which uses and composes services to achieve its goals. Some describe "user facing" web services, these provide informational services via web pages (the stock ticker is a classic example). Others see web services as being able to invoke other (downstream) services themselves in order to provide composite and complex functionality. These interactive services, will have various levels of reactive and proactive functionality depending on the users' (human, agent, or other service) goals, and the interaction capabilities of both parties.

In addition, there is no clear idea or agreement about how web services will be used. At present there appear to be three ways to use web services. The first services made available[1] are rather simple, single function programs that can be invoked across the web. In the main, they are located by manually searching a UDDI[2] registry, and their invocation and use is pre-programmed by developers. The second type, are services that automate interactions between business partners who know each other and can agree on the syntax and semantics of the services up-front, similar to EDI. The third type, are "semantic web services" [5]. These are on the boundary between services, semantic web and agents. These web services of the future will provide complex functionality to previously unknown interaction partners. The exact nature of what they provide, and how to interact with the service will be contained within "self describing" machine readable documents [6,7].

The web services architecture is based on a "Publish, Find, Interact" model [8,9]. This model is insufficient for web services to progress towards automated service to service interaction. A more complex model that includes description, advertisement, discovery, evaluation and selection, initiation of dialogue, negotiation, configuration, interaction, and management is required. The first step is description. Web service descriptions must contain sufficient information to allow each step to proceed automatically.

As yet, there is no clear definition of what self description really means, but web services are still software. Therefore we can use the requirements for ordi-

[1] http://www.xmethods.com
[2] http://www.oasis-open.org/committees/uddi-spec/

nary software interface descriptions as the basis for requirements for web service interface descriptions. The reason for doing this is to ensure that the description of web service interfaces is at least as comprehensive as the description provided for other types of software. Web services operate in a complex environment and there should be more information about them provided to developers and users than for ordinary software.

In the next section we outline the requirements for software interfaces described in [10]. In section 3 we compare two prominent specifications for web service description (WSDL and DAML-S), against these requirements. We conclude with section 4.

2 Documenting Interfaces

In this section we give an overview of the interface documentation template presented in [10] and [11] and comment on how the sections apply in the web services context. These requirements have been refined over many years by the software engineering community. Each element is necessary to ensure that the developers implementing the interface, and third parties using the interface, can fully understand what the interface requires, what it provides and its constraints.

1. **Interface identity**: The most common means is to give a name to the interface and version numbers if appropriate.
   A unique identity for an interface is particularly important when many implementations of the same interface are expected or if different interfaces to the same service are provided for different classes of users. The identity can be used by service advertisements in catalogues and registries to indicate that the service complies with a particular interface.
2. **Resources provided**: These are the operations or methods provided in the interface.
   The description of resources should be sufficient to aid the discovery, evaluation and selection of services that meet the users' needs.
   Each resource needs to describe its:
   a) **Syntax**: The signature including its name and the logical datatypes of arguments.
   b) **Semantics**: A description of what happens when the resource is used i.e. what is visible to the user, and what are the restrictions on use of the resource. For example, the semantics describe the assignment of values to data that the user can access; changes in state, either to this, or another, element; the events signalled and messages sent by the resource; details of how other resources will behave differently when this one is used; and the execution style, whether the operation is atomic, interruptible or suspendible.
      The semantics of the operations is perhaps the most important part of the description in terms of enabling automated interaction. The semantics will be necessary in most phases from evaluation and selection to interaction.

   c) **Usage restrictions**: Similar to pre-conditions, these state the assumptions about the environment that must be true, or describe the side effects of the operation. Exceptions should also be described here, detailing how errors are handled. For example, the number of retries or the meaning of returned status indicators.
3. **Locally defined datatypes**: This section describes how to declare variables, constants and literal values of the datatypes defined and used in the interface, and the operations, comparisons and conversions that can be performed on instances of those types.
   Datatypes in the web context can be viewed as both traditional programming language constructs and XML documents. In the case of XML documents, a reference to the document schema or a document template can be provided to allow creation of required documents or to enable understanding of the documents supplied by resources.
4. **Error handling**: The errors that can be raised by the resources on the interface and error handling behavior.
   This information will be necessary during the interaction phase to determine the cause of unexpected results.
5. **Variability provided by the interface**: Details of what configuration is possible, and range of allowable values for each configuration parameter should be provided. In addition, a description of how configuration affects the semantics of the interactions.
   Web service users will be of many different types, operating in different contexts, each with different capabilities. Consequently, web service descriptions must provide the facility to describe what can be configured and how that configuration can be achieved. In addition to operation attributes, there are other aspects of service delivery that could be configurable such as, the interaction mechanism or the security and transaction management protocols.
6. **Quality attributes of the interface**: Quality attributes include such things as the level of performance and reliability that the interface provides. Services may be selected based on their certified conformance to specific standards, or their ability to provide various quality of service attributes.
7. **What each interface element requires**: Either, specific named resources (described as above with syntax, semantics and restrictions), or other preconditions or assumptions about the environment.
   The usage restrictions (or non-functional requirements) of the operations will be used in the evaluation and interaction phases to determine if the constraints can be satisfied by the user.
8. **Rationale**: The motivation for the design, the constraints, compromises and alternatives considered.
   This is mainly for the benefit of developers implementing the interface.
9. **Usage guide**: the protocol of interaction or patterns of use for the entire interface.
   This information will be used during the interaction phase to ensure the correct order of interaction. This information could also be used to describe how to initiate a dialogue with the service, how to negotiate with or configure

the service, and how to manage the operation of the service. Alternatively, the information may be used to select services that provide compatible interaction mechanisms.

The interface requirements described here can be considered the minimum requirements for the interfaces of web services. In the next section we use the template to determine how much of this information can be expressed using the current web-service specifications WSDL and DAML-S.

3 Evaluation of Standards in Terms of Interface Requirements

In this section we evaluate two web service specifications in terms of the interface documentation template introduced in section 2. The specifications are, the Web Services Description Language (WSDL) Version 1.2, W3C Working Draft 9 July 2002[3] and DAML-S 0.7 Draft Release[4].

3.1 WSDL Evaluation

WSDL is the primary specification for web service descriptions and it has achieved a high degree of acceptance. It provides a machine processable language, with XML syntax, for the description of web services.

1. Identity: No specific identity attribute is provided by WSDL. When registered in a UDDI registry as part of a UDDI tModel, a unique identity is assigned by the registry. The URI of the interface specification can be used as a unique identifier, however identical copies of the same interface with different URI's have to be treated as different resources.
2. Resources provided: A WSDL portType (interface) has a collection of operations (methods). Each operation has a set of input and output messages. Each message has one or more parts (parameters). Messages are defined outside of the operation, so in theory they are reuseable. Each message part has a name and a type. Although any type system can be used, XML Schema[5] datatypes are preferred.
   a) Syntax: Operation signatures are provided in XML syntax according to the WSDL 1.1 XML Schema specification, for example:

```
<message name="aNameMessage">
  <part name="firstName" type="xsd:string">
</message>

<operation name="printName">
```

[3] http://www.w3.org/TR/2002/WD-wsdl12-20020709/
[4] http://www.daml.org/services/daml-s/0.7/
[5] http://www.w3.org/XML/Schema

```
      <input message="aNameMessage"/>
      <output message="returnName"/>
    </operation>
```

   b) Semantics: WSDL makes no provision for semantics [12].
   c) Usage restrictions: Not supported.
3. Local data types: Defined using XML Schema complexTypes. No support for operations on the defined types.
4. Error handling: Not supported.
5. Variability: Not supported.
6. Quality attributes: Not supported.
7. Required resources: Input messages (as described above) are how required resources are described in WSDL. There is no support for semantics, usage restrictions or pre-conditions.
8. Rationale: Not supported, although could be added as documentation.
9. Usage guide: Not supported, although other specifications such as BPEL4WS[6] provide this kind of information for WSDL services.

3.2 DAML-S Evaluation

DAML-S provides several ontologies, based on the DAML ontology language, for describing the properties and capabilities of web services. DAML-S markup is intended to facilitate the discovery, execution and interoperation of web services [13]. DAML-S provides three sub-ontologies, each provides a different view of the service. The Profile describes three types of information including information about the organization providing the service, what function the service performs, and non-functional service characteristics. The Process Model describes how the service works in terms of its inputs, outputs, preconditions and effects. The Grounding describes how a service is used and provides a mapping from the the Profile and Process specifications to specific concrete protocols and message formats.

In this evaluation we concentrate on the Service Profile, but describe elements from the Process model and Grounding when they provide more information. Although the various models provide a good separation of concerns they can be confusing when they appear to overlap. For example, the Profile describes parameters that have a name and an unspecified range. These Profile parameters make references to input and output parameters separately described in the Process model.

1. Identity: A Profile has a unique serviceName attribute.
2. Resources provided:
   a) Syntax: XML syntax is used with the vocabulary from the DAML and DAML-S specifications, for example:

[6] http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

```
<profileHierarchy:BookSelling
  rdf:ID="Profile_Full_Congo_BookBuying_Service">

  <!-- reference to the service specification -->
  <service:presentedBy
    rdf:resource="&congoService;#FullCongoBuyService"/>
  <profile:serviceName>
    Congo_BookBuying_Agent
  </profile:serviceName>

  <profile:input rdf:resource="#BookTitle"/>
  <profile:output rdf:resource="#ShippingOrder"/>

  <!-- specification of quality rating for profile -->
  <profile:qualityRating rdf:resource="#Congo-Rating"/>

  <!-- Preconditions and effects -->
  <profile:precondition rdf:resource="#AcctExists"/>
  <profile:effect rdf:resource="#BuyEffectType"/>
</profileHierarchy:BookSelling>
```

b) Semantics: The Profile provides the means to describe pre-conditions and effects. Processes can define conditions and condition effects for parameters. Other elements, such as computedInput and computedOutput, in the Process model can be used to give explicit semantics for specific items.
Other aspects of semantics, such as events signalled and how other resources will behave differently are not supported.
c) Usage restrictions: Some support, but the property *domainResource*, that describes resources necessary for the task to be executed, has been deprecated.

3. Local data types: Most DAML-S objects are defined locally in DAML. Operations on data types are not supported.
4. Error handling: Not supported, although errors could be described in terms of *effects* as described in the Process specification.
5. Variability: Not supported.
6. Quality attributes: Various quality attributes can be defined for services, pre-defined attributes include *maxResponseTime* and *avgResponseTime*.
7. Required resources: Input parameters as described above.
8. Rationale: Not supported, although could be added as documentation.
9. Usage guide: The Process Control Model provides various control constructs including splits, joins, sequence etc. Composite processes can specify constraints on the ordering and conditional execution of sub-processes.

3.3 Summary

Table 1 gives a summary of the results of the evaluation. A "-" indicates no support, "+/-" indicates some support and "+" indicates reasonable support for the item.

Table 1. Summary of evaluation results

	WSDL	**DAML-S**
1 Identity	-	+
2 Resources provided		
2a Syntax	+	+
2b Semantics	-	+/-
2c Restrictions	-	+/-
3 Local Datatypes	+/-	+/-
4 Error handling	-	-
5 Variability	-	-
6 Quality	-	+
7 Resources required	+/-	+/-
8 Rationale	-	-
9 Usage guide	-	+

Both specifications rate well on providing the syntax of operation signatures. Unfortunately this is just about all that WSDL does provide. This means that all the processes of discovery, selection etc. must be based on operation signatures, which is clearly inadequate for all but the most simple services. To progress beyond manual discovery, selection and pre-programmed interaction, WSDL needs to be expanded to include the other aspects of interface description, or other specifications (such as BPEL4WS) covering those aspects must be developed.

DAML-S provides a better interface description for semantics and the usage guide. The main problem areas are error handling and configuration. Errors are inevitable, and interfaces must declare what errors are possible and how they are handled, neither specification provides primitives for this kind of information. The lack of error handling, which is essential in production systems, may be a reflection on the immaturity of the specifications or a deliberate attempt to reduce their complexity. In either case, this issue must be dealt with before widespread commercial use of web services is possible.

Although there is no support in DAML-S for the definition of operations on locally defined data types, this is typical of ontology definitions. Ontologies focus on describing elements in terms of their relationships, rather than on how instances of the elements can be used.

4 Conclusion

The evaluation of the specifications reveals that web service interfaces created with WSDL and DAML-S provide much less information than is usually expected for software interfaces. It is not reasonable to expect services to operate in a complex environment like the web, and at the same time provide less information about them than is necessary for ordinary software.

The evaluation also shows that WSDL and DAML-S overlap in some areas, for example they both describe the syntax of operations very well. Although some work is being done in DAML-S to align these two specifications [13], further effort should be directed at making web service interfaces more comprehensive.

In the future when web services engage in automated ad-hoc interaction, they will also need to describe the domain they operate in and the products they deal with. They will need to detail their security and authentication policies, transaction management procedures, and the interaction mechanisms they support. Web services will be self describing when they can do all this as well as provide the basic information discussed in this paper.

References

1. Tidwell, D.: Web services : Education : Tutorials web services – the web's next revolution (2000) Available from `http://www-105.ibm.com/developerworks/`.
2. McDermott, D., Burstein, M.: Overcoming Ontology Mismatches in Transactions with Self-Describing Service Agents. In: Proceedings of SWWS' 01 The First Semantic Web Working Symposium, Stanford University, California, USA (2001) 285–302 Available from: `http://www.daml.org/services/daml-s/2001/05/`, (20 September 2001).
3. Sheth, A., Meersman, R.: Amicalola Report: Database and Information Systems Research Challenges and Opportunities in Semantic Web and Enterprises (2002) Database and Information Systems Research for Semantic Web and Enterprises. Invitational Workshop Sponsored by NSF CISE-IIS-IDM, Co-Sponsored by EU Thematic Network OntoWeb. In cooperation with VP-Research and LSDIS Lab, University of Georgia. April 3 - 5, Amicalola Falls and State Park, Georgia. Organizers: Amit Sheth and Robert Meersman. Available from:`http://lsdis.cs.uga.edu/SemNSF`, (14 December 2002).
4. Moore, J.W.: Fundamental Principles of Software Reuse (1997) Eighth Annual Workshop on Institutionalizing Software Reuse (WISR), held at the Ohio State University. Available from: `http://www.umcs.maine.edu/%7Eftp/wisr/wisr8/wisr8.html`, (11 January 2003).
5. Bussler, C., Fensel, D., Payne, T., Sycara, K.: Tutorial (T3): Semantic Web Services (2002) More information available at: `http://www.daml.ri.cmu.edu/tutorial/iswc-t3.html`, (15 October 2002).
6. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF (2002) to appear in Electronic Commerce Research and Applications. Available from: `http://www.cs.vu.nl/~dieter/wese/wsmf.paper.pdf`, (30 September 2002).
7. Sollazzo, T., Handschuh, S., Staab, S., Frank, M.: Semantic Web Service Architecture - Evolving Web Service Standards toward the Semantic Web. In: Proc. of the 15th International FLAIRS Conference, Pensacola, Florida, AAAI Press (2002) Available from: `http://www.citeseer.nj.nec.com/sollazzo02semantic.html`, (24 September 2002).

8. Hugo Haas (Activity Lead): Web Services Activity Statement (2002) `http://www.w.org/2002/ws`, (12 March 2002).
9. Bussler, C., Fensel, D., Maedche, A.: A Conceptual Architecture for Semantic Web Enabled Web Services. SIGMOD Record, Special Section on Semantic Web and Data Management **31** (2002)
10. Clements, P., Bachmann, F., Ross, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond. Addison-Wesley, Boston, USA (2003) ISBN 0-201-70372-6.
11. Bachmann, F., Bass, L., Clements, P., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architecture: Documenting Interfaces. Technical Note CMU/SEI-2002-TN-015, Carnegie Mellon Software Engineering Institute, Pittsburgh, PA (2002)
12. Booth, D.: Semantics and WSDL (2002) Available from: `http://www.w3.org/2002/09/wsdl-semantics-dbooth/semantics_clean.htm`, (12 December 2002).
13. Martin, D., Burstein, M., Lassila, O., Paolucci, M., McIlraith, S.: Describing Web Services using DAML-S and WSDL (2002) DAML-S Coalition working document; August 2002. Available from: `http://www.daml.org/services/daml-s/0.7/daml-s-wsdl.html`, (19 January 2003).