Composing Workflows of Semantic Web Services

Mikko Laukkanen and Heikki Helin
TeliaSonera Finland
P.O.Box 970 (Teollisuuskatu 13)
FIN-00051 SONERA, Helsinki, Finland
{mikko.laukkanen,heikki.j.helin}@teliasonera.com

ABSTRACT

Web services are software components that can be accessed over the Internet by other software components. Web service workflows are a set of web services that are executed in a structured way. This paper introduces a workflow composer agent, which is able to compose web service workflows, and more importantly, uses semantic descriptions of web services in finding and matching web services for a workflow. We will show how a workflow can be composed by utilizing semantic web service ontologies. With this technology we will present a model for composing web service workflows, and provide a example scenario, which shows how the composition of a web service workflow can be done in practise.

1. INTRODUCTION

The web, once solely a repository of static data, such as of text and images, is evolving into a provider of services [6]. In the future the web will be increasingly dynamic in its nature, that is, services may be available infrequently, for instance based on time and location. Furthermore, the services will be composed of one or more web services, and it is normal to expect web services to be integrated as part of workflow processes [3].

Web services are standard-based software components that can be accessed over the Internet by other software components [13]. Web services can vary in functionality from simple operations, such as a retrieval of a stock quote, to complex business systems, such as online travel scheduling, which access and combine information from multiple sources. Once deployed, other applications and web services can discover and invoke the web service.

When speaking about a composition of web service workflows, one may refer to at least two cases. Firstly, one can have a system, where the workflow is already defined, but one or more web services of it is unavailable. Secondly, a brand new workflow can be generated, and the web services for it are searched. In both cases, it may happen that one or more web services in the workflow are not available at the time of invocation, thus, it is required to search and find a replacing web service(s) to complete the workflow. Should this be the case, the replacing web service(s) must implement similar enough functionality than the one that is to be replaced. Because web services are deployed by various organizations world-wide, in reality it is very unlikely to find a perfect match for a web service. The reason can be for instance the different number of input or output parameters, but maybe above all, the semantic meaning of the input and output parameters. There has to be a way to find semantically similar kinds of services, which have the same kind of pre- and post-conditions. In other words, semantically similar services have a same kind of effect on the "state of the world".

The rest of this paper is structured as follows. Section 2 will give background information about ontologies and Business Process Execution Language for Web Services (BPEL4-WS), which are used later in the paper. In Section 3 we will describe how the web service workflow can be composed using the semantic descriptions of the web services. Section 4 will give an example, which illustrates how the model can be applied in practise. Finally, in Section 5 we will discuss some open issues and future work, and conclude this paper.

2. BACKGROUND

2.1 Ontologies

There is a need for an ontology when applying search and semantic matching for web services. To illustrate this, let us consider the following scenario. A web service that provides the location of a user's mobile phone needs to be found. In the search arguments we specify that the inputs for the web service should be the "phoneNumber", and the outputs are the "longitude" and "latitude" of the phone. Unfortunately, when searching for replacing web service, a exact match is not found. However, a service, where the input is called "MSISDN", and the output is called "Location", is available. Without knowing anything about the semantics, we would discard this service in the first place. However, if we would have a ontology defining that "MSISDN *is-kind-of* phoneNumber" and "Location *is-composed-of* longitude and latitude", we could see that in fact the found web service would serve our needs.

The DARPA Agent Markup Language (DAML) [5] is a language for expressing sophisticated class definitions and properties. The DAML-S [2] is a DAML-based web service ontology, which specifies how a web service can be supplemented with a semantic description. The DAML-S is

.

divided into three parts: a service profile, a process model, and a grounding. The service profile is used for advertising and discovering services, that is, it describes the service in terms of its inputs, outputs, preconditions and the effects it has once it is executed. The process model gives a detailed description of a service's operation, that is, how the service should be used. While the service profile and the process model are abstract descriptions of the service, the grounding on the other hand provides concrete details on how to interoperate with a service. For instance, in DAML-S 0.7 and later the grounding can be binded to WSDL, which in turn allows it to be interoperable within the web service world.

Later in this paper we will show how a semantically annotated service profile plays an essential role when finding semantically similar web services The service profile could be realized for instance using DAML-S.

2.2 Business Process Execution Language for Web Services

Business Process Execution Language for Web Services (BPEL4WS) [1] allows modeling business processes (workflows) for web services. BPEL4WS depends on WSDL [14], XML Schema [12], and XPath [11]. Of these, the WSDL has the most influence on BPEL4WS; in fact, BPEL4WS builds on top of WSDL. BPEL4WS represents the combination of two previously competing standards: XML business process language (XLANG) [10] from Microsoft, and web services Flow Language (WSFL) [4] from IBM.

The BPEL4WS workflow can be defined using the following general structure:

```
<process name="TheProcess">
  <partners>
      <!-- lists the external web services
      invoked from within the workflow -->
  </partners>

  <variables>
      <!-- specifies the data elements that
      flow within the workflow -->
  </variables>

  <correlationSets>
      <!-- specifies bindings for a set of
      operations to a service instance -->
  </correlationSets>

  <faultHandlers>
      <!-- lists the elements to catch
      faults -->
  </faultHandlers>

  <compensationHandler>
      <!-- specifies the elements that
      implement compensating actions in
      the case of transaction rollback -->
  </compensationHandler>

  <eventHandlers>
      <!-- for receiving external events to
      the workflow -->
  </eventHandlers>

  <sequence>
      <!-- the workflow execution logic -->
  </sequence>
</process>
```

Every process begins with a "header" XML fragment, which specifies the process name and namespaces being referred to. Partner section specifies the external web services invoked from within the workflow. Partner definition has also reference to the WSDL documents of the web service. When composing workflows, the partner section is created based on the web services that have been found during semantic matching. Variables contain the data that flows within the workflow. Based on the web services' inputs and outputs (as specified in the WSDL document), the variables section can be formed. Correlation sets are used to bind a set of operations to a service instance. Fault handlers are used in catching failures and the sequence comprises the actual workflow logic. Compensation handlers are used to implement compensating actions in the case of transaction rollback. Event handlers are used to receive external events to the workflow. Finally, the web services can be invoked as specified by the sequence section, which may include basic control flow structures, such as `sequence`, `switch` (for conditional routing), `while` (for looping), `flow` (for parallel execution), and `pick` (for race conditions based on external triggers).

3. COMPOSING WORKFLOWS

3.1 Model for Composing Workflows

Figure 1 depicts the general entities of the web service composition model and the information flow between them. When a new web service instance is created, it is advertised by registering the WSDL and DAML-S description to the directory, which can be for instance UDDI [7] `(1)`. The workflows are stored in the workflow repository, from where they are fed into the workflow composer agent `(2)`. Each workflow is composed of one or more web services, which can be situated anywhere on the web. The DAML-S and WSDL descriptions of the web services are queried from the directory `(3)`, and the semantic matching is applied. The workflow composer agent composes the executable workflow, and feeds it into the workflow execution engine `(4)`. Finally, the execution engine executes the workflow using the web service instances.

There are two cases to consider when composing web services: to replace a web service in a existing workflow with a similar functionality or to define a whole new workflow using the available web services on the web. In the following, both of these options are discussed in detail, and a four-step algorithm is presented for composing a workflow.

3.2 Creating or Updating a Web Service Workflow Dynamically

The web is evolving from a collection of information to a distributed computing environment of web services, where the web services may be available more or less infrequently, depending on for instance the time and location. When a workflow is composed of such web services, it may happen that at the time of initiating the workflow, one or more of the web services are not available. Should this happen, a similar functionality, implemented by one or more web services, should be found. In addition to replacing unavailable web services in a workflow, there might be a need to compose a whole new workflow, or extend an existing workflow to cover new web services.

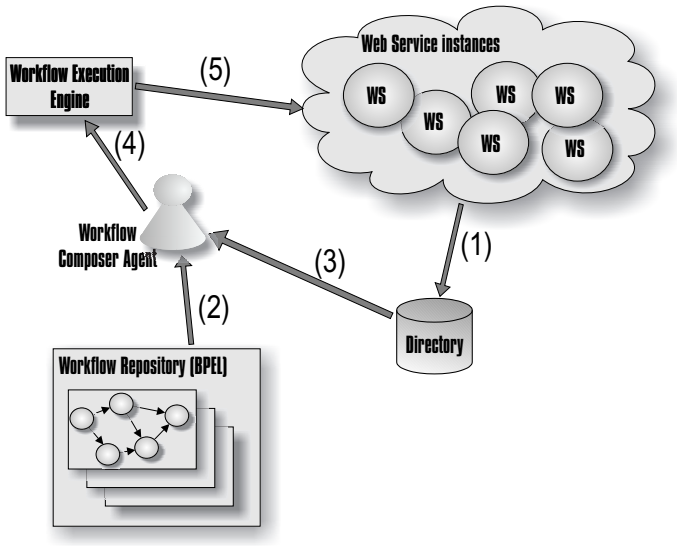

Figure 1: Enabling architecture for dynamic workflow composition. For explanations for the numbers in parentheses, see the body text.

The process of finding a web service to replace an unavailable one in some workflow or creating a whole new workflow can be divided into the following general steps:

1. Identifying the required functionality
2. Semantic matching of web services
3. Creating or updating the workflow
4. Executing and monitoring the workflow

In the following, each of the steps are described in more detail.

3.3 Identifying the Required Functionality

The first task is to identify the general functionality that the workflow should accomplish. When replacing some unavailable web service in a workflow, the identification for the functionality is easy; a similar web service or web services implementing the similar functionality than the unavailable one has to be found. However, when creating a new workflow, the identification for the functionality usually originates from some problem that needs to be solved. For instance, there could be a need to arrange a business trip to a foreign country. This kind of workflow consists of several tasks, such as booking airline tickets, booking a hotel, and renting a car. Furthermore, these tasks may have some dependencies on each other. For instance, if the the arriving airport is far a way from the hotel, renting a car may be the best choice, whereas if the airport is near to the hotel, the use of public transportation could be better justified. The output of the first step is a list of tasks (later mapped to web services), which are described in terms of pre- and post-conditions as well as input and output arguments.

3.4 Semantic Matching of Web Services

After the general tasks have been identified, the web services implementing the workflow can be searched and matched. Semantic matching can be divided into two parts: finding the web service(s) that fulfills the pre- and post-conditions of the needed functionality, and finding the web service(s) that accept the required input and output arguments. To make this possible, the available web services have to be accompanied with a semantic description, which is able to express this kind of information. Furthermore, the pre/post-conditions and input/output arguments should refer to an ontology.

Pre-conditions specify the state of the "world" before, and post-conditions after the web service is executed. When an unavailable web service is to be replaced, the replacing functionality, let it be a single web service, or a set of web services executed as a composed service, must have the same pre- and post-conditions.

While the similar pre- and post-conditions generally is enough, in practise the input and output arguments need also be matched. This step follows the ideas of Paolucci *et al* [8], where the semantic matching algorithms for input and output arguments are presented. When searching and matching web services, a similar kind of DAML-S description that the advertisement, so called search template, is created.

The input and output parameters in both the advertisement and the search template refer to an ontology, and the similarity is evaluated by the relationships between the referred classes in the ontology. This results in that there can be four different kinds of matches [8]:

- Exact match—the searched service and the advertisement refer to the same class in the ontology.
- Plug-in—the advertisement is more general than the searched service.
- Subsumes—the searched service is more general than the advertisement; in this case the advertisement cannot completely fulfill the searched service.
- Fail—No subsumption relation between the search template and the advertisement can be found.

In the case of exact match the advertised web service can be used as such. However, in the case of plug-in and subsumption, the advertised web service can be used, but some additional processing may be needed. The plug-in service produces more results than needed, thus, some kind of filtering is needed. The subsumed result is not complete, so the requester may need other web services to complete the result set.

3.5 Defining the Workflow

Assuming all the web services implementing the required functionality are found, the next step is to define the workflow or update the existing one. This includes the (re)binding of the web services to the workflow, and definition of the possible dependencies.

If the composition of the workflow is about replacing unavailable web services, this phase is trivial; the only thing to do is to update the partner section in the workflow to refer to the new web services (i.e., to their WSDL documents). However, if a new workflow is created, this phase is far more complex, and most likely requires human interaction at least in defining the actual workflow logic. In this paper we are not discussing this side of the coin, but leave it as the future work.

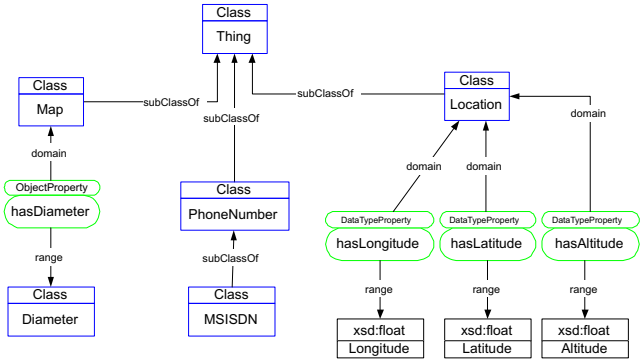

Figure 2: The example ontology used in the scenario

3.6 Executing and Monitoring the Workflow

Once the workflow is (re)defined, it can be executed by a workflow execution engine, which both executes the workflow as defined, and makes the appropriate external web service invocations. The workflow execution engines usually provide tools for monitoring the execution of the workflows, and for verifying the output of the workflow. To our knowledge, there are at least two implementations for BPEL4WS engine: BPWS4J [1] (IBM), and BPEL Orchestration Server [2] (Collaxa). The former is available as open-source distribution, whereas the latter is a commercial product.

4. EXAMPLE SCENARIO

Let us illustrate the workflow composition with the following scenario, where a traveler is on a vacation, and asks for restaurants near her current location. Furthermore, the restaurants should be pointed on a map.

During the execution of the scenario, the workflow composer agent refers to an ontology, which is depicted in Figure 2.

There is a workflow specification for the required functionality, and it is composed of two web services: `LocateMap` and `GetRestaurants`. The `LocateMap` web service takes the user's phone number (MSISDN) and a diameter for a map as an argument, and provides a map together with latitude and longitude of the center point as output arguments. The `GetRestaurants` web service uses the latitude, longitude and map as the input arguments to add the nearby restaurants to the map. The workflow is depicted in Figure 3.

In this scenario we will concentrate on the `LocateMap` web service, which has the following pre- and post-conditions:

[LocateMap]

Pre-conditions:	$\neg known(latitude) \wedge$
	$\neg known(longitude) \wedge$
	$\neg known(map)$
Post-conditions:	$known(latitude) \wedge$
	$known(longitude) \wedge$
	$known(map)$

Furthermore, as shown in Figure 3, the `LocateMap` accepts MSISDN and diameter as the input arguments, and latitude, longitude and map as the output arguments.

[1]See: http://www.alphaworks.ibm.com/tech/bpws4j
[2]See: http://www.collaxa.com/home.index.jsp

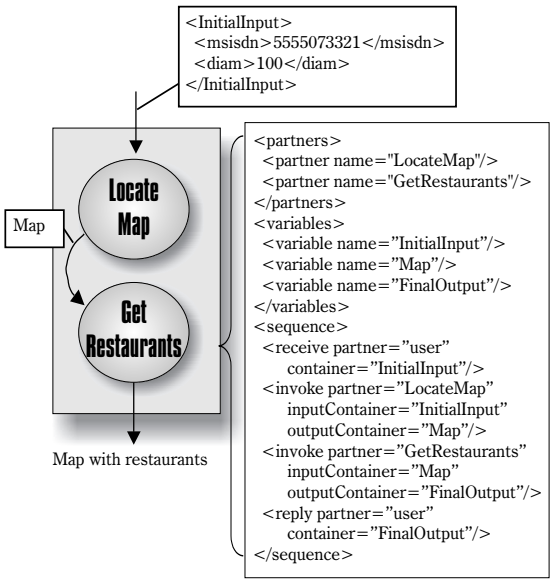

Figure 3: The existing workflow for locating nearby restaurants. Please note that the BPEL4WS definition only includes the relevant information and therefore is not complete.

When the traveler makes the request for locating the nearby restaurants, the workflow composer agent finds out that the `LocateMap` web service is not available. It then begins to search for replacing web services with one or more similar matching pre- or post-conditions that the `LocateMap` has. In this scenario there are three such web services available: `LocatePhone`, `GetLatLon`, and `GetMap`. The pre- and post-conditions as well as input and output arguments are as follows:

[LocatePhone]

Pre-conditions:	$\neg known(location)$
Post-conditions:	$known(location)$
Input:	$MSISDN$
Output:	$location$

[GetLanLon]

Pre-conditions:	$known(location) \wedge$
	$\neg known(latitude) \wedge$
	$\neg known(longitude)$
Post-conditions:	$known(latitude) \wedge$
	$known(longitude) \wedge$
	$known(map)$
Input:	$location$
Output:	$latitude \wedge longitude$

[GetMap]

Pre-conditions:	$known(latitude) \wedge$
	$known(longitude) \wedge$
	$\neg known(map)$
Post-conditions:	$known(map)$
Input:	$latitude \wedge$
	$longitude \wedge$
	$diameter$
Output:	map

From the available web services and their pre- and post-conditions the workflow composer agent can infer the following:

1. `GetLatLon` fulfills the post-conditions, *known*(*latitude*) and *known*(*longitude*), of `LocateMap`.
2. `GetLatLon` has a pre-condition *known*(*location*) than in turn is the post-condition of the `LocatePhone` web service. Thus, `LocatePhone` should be called in order to fulfill the post-conditions for the `GetLatLon`.
3. `GetMap` web service has pre-conditions *known*(*latitude*) and *known*(*longitude*), which are post-conditions of `GetLatLon`. Clearly, `GetLatLon` should be invoked in order to fulfill the preconditions of the `GetMap`.
4. `GetMap` has the same post-condition than the `LocateMap` has. Therefore, together with `GetLatLon`, invoking the `GetMap` fulfills all the post-conditions than the unavailable `LocateMap`.

What about the input and output arguments then? We can see that the concept of `Location` is not present in the unavailable `LocateMap` web service. However, by referring to the ontology, the workflow composer agent, during the semantic matching of the web services, is able to infer that the `Location` in fact has the longitude and latitude as properties. Therefore, the when knowing the `Location`, the latitude and longitude are also known. In the matter of fact, if the ontology would also be used in the analysis of the pre- and post-conditions, the workflow composer agent could simplify the analysis by leaving the `GetLatLon` out; the ontology states that a condition *known*(*location*) also means that the conditions *known*(*latitude*) and *known*(*longitude*) are also true.

Once the workflow composer agent has analysed the available web services, and found out that by combining them in as a sequence the unavailable `LocateMap` web service can be replaced, the workflow can be re-defined as depicted in Figure 4.

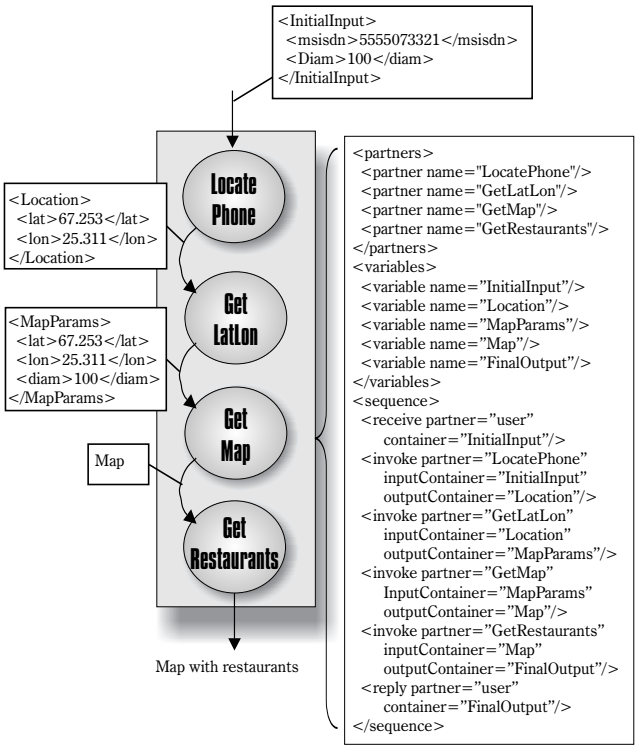

Figure 4: The new workflow for locating nearby restaurants. The BPEL4WS definition only includes the relevant information and therefore is not complete.

5. DISCUSSION AND CONCLUSION

In this paper we have discussed the composition of web service workflows, where one of the objectives is to automatise the process of finding and matching the web services accessed by the workflow. The model we have presented still requires human interaction in the initial phase of the workflow composition (see Section 3.3). To move the burden of defining the requirements and creating the initial workflow description from a human user to a computer program is out of this paper's scope. This kind of automatisation would require a complex planning and reasoning functionality, which are studied for instance by the software agent technology research community.

The semantic matching phase (see Section 3.4) plays a central role in the success of the workflow composition. In this paper we referred to the work of Paolucci *et al* in [8] in applying the actual semantic matching based on DAML-S descriptions. Because web services are deployed by various people world-wide, in reality it is very unlikely to find a perfect match for a web service. The reason can be for instance the different number of input or output parameters, but maybe above all, the semantic meaning of the input and output parameters. The ontology and its ability to cover the concepts used in the DAML-S descriptions is maybe the most important issue in the semantic matching. In the worst case the concept presented in the DAML-S description is not found from the ontology, which means that the corresponding Web Service cannot be used by the semantic matching phase at all.

The work presented in this paper did not depend on any particular directory service. Because the UDDI is specified as the repository for web services advertisements, we are not ruling out the possibility of using UDDI in storing the DAML-S descriptions together with the other information of web services. To do this, the UDDI needs to be supplemented with the functionality to be able to hold DAML-S descriptions. In fact, [9] describes how this can be done.

In this paper we have not discussed the relationship between the semantic description stored in the directory service and the concepts represented by the ontology: who updates it and keeps it consistent? For instance, when a new web service is deployed, is it the web service author's task to look up the ontology and "classify" the web service when creating the DAML-S description? Or could it even be that the web service author needs not to know about the existence of the ontology at all? Instead, the administrator of the directory service could be responsible for creating and maintaining the DAML-S descriptions once a new web service is advertised. These issues are out of this paper's scope and remain as future work.

6. REFERENCES

[1] T. Andrews, F. Curbera, H. D. Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1, Mar. 2003.

[2] A. Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In *Proceedings of The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), June 2002.

[3] J. Cardoso and A. Sheth. Semantic e-Workflow Composition. Technical report, LSDIS Lab, Computer Science, University of Georgia, July 2002.

[4] F. Leymann. Web Service Flow Language (WSFL 1.0), May 2001. Available at: http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[5] J. Hendler and D. L. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):67–73, 2000.

[6] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[7] OASIS Consortium. UDDI Version 3.0 Specification. July 2002. Working Draft, available at: http://www.oasis-open.org/committees/uddi-spec/.

[8] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *The Proceedings of The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), June 2002.

[9] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Importing the Semantic Web in UDDI. In *Web Services, E-Business and Semantic Web Workshop (WES)*, pages 225–236, May 2002.

[10] S. Thatte. XLANG: Web Services for Business Process Design. Microsoft Corporation, 2001. Available at: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

[11] W3C. XML Path Language (XPath), Version 1.0. Nov. 1999. W3C Recommendation, available at: http://www.w3.org/TR/xpath/.

[12] W3C. XML Schema Part 0: Primer. May 2001. W3C Recommendation, available at: http://www.w3.org/TR/xmlschema-0/.

[13] W3C. Web Services Architecture. Nov. 2002. W3C Working Draft 14, available at: http://www.w3.org/TR/2002/WD-ws-arch-20021114/.

[14] W3C. Web Services Description Language (WSDL) Version 1.2. Jan. 2003. W3C Working Draft 24, work in progress, available at: http://www.w3.org/TR/2003/WD-wsdl12-20030124/.