Web Service Scope Annotations *

Jacek Kopecký, Thomas Strang

Digital Enterprise Research Institute, Innsbruck
{firstname.lastname}@deri.org

Abstract. Web services are usually described with Web Service Description Language, which generally assumes that services are always available, and it does not provide a built-in mechanism for describing that services may have a *scope*. In our paper, we describe the concept of web service scope and we propose a simple mechanism for annotating WSDL with scope information, so that clients can automatically evaluate whether a particular service is useful for them. We demonstrate how XPath can be used to express scoping conditions, however, our solution is extensible with respect to the scope expression language so semantic languages can also be incorporated for applications that require the full expressivity of logics.

1 Introduction

Current web service description languages (WSDL [15] versions 1.1 and 2.0) only specify how and where a web service is accessible, assuming the web service is generally always available (barring network and server outages). However, many services have limited *scope*, a set of conditions under which the service can function.

As examples of services with limited scope we can think of a shipping service that can only deliver packages up to 50kg apiece within Austria and Germany; a pizza delivery service that does not operate between midnight and 6am; an electronic shop that only ships purchased products to Canada and mainland USA; or a digital media hosting service that can only stream videos with bit rates up to 2Mbit/s.

These examples indicate that there are different parameters of scope, such as time (operating hours), location of some aspect of the service (shipping address), or other arbitrary variables like the package weight or size, or media file bit rate.

It is generally assumed that the scope of a service will be self-evident (we do not expect a small Austrian shipping service to deliver to Brazil) or documented somewhere in plain text. However, such descriptions of the scope are not amenable to automated processing. In this paper, we propose a mechanism of enhancing WSDL web service descriptions with scope information that will allow a client to evaluate the scope conditions automatically before invoking a service and thus avoid unnecessary interactions that would only result in *out-of-scope* faults.

Apart from preventing unnecessary faults, machine-processable scope information in web service description also allows the client software to select a more appropriate service when multiple alternatives are available, or to generate more meaningful warning or error messages for the user.

* This material is based upon works supported by the EU funding under the project DIP (FP6 – 507483).

We have selected web service scope as a well-defined and concrete part of general web service metadata. In effect, our approach is in between the current purely syntactic WSDL web service description and the fully-fledged Semantic Web Service (SWS) approaches such as the Web Service Modeling Ontology (WSMO [4]).

Semantic web service description is very related to process modeling (because a service can be viewed as a process), and process models often refer to preconditions or assumptions. In this paper we aim to narrow the concept of preconditions down into what we call service scope, and to define clearly and exactly where and how service scopes can be used. In comparison with generic precondition/assumption systems, service scope should be easier to adopt and implement in existing web service deployments.

In the remainder of this paper, we first detail in section 2 what we mean with web service scope and how it can be expressed. In section 3 we propose a mechanism for annotating WSDL files with scope information. Related work, especially with respect to Ubiquitous Computing, is covered in section 4, and in section 5 we present our conclusions.

2 Web Service Scope

Under the term *web service scope* we understand any practical limitations of a service that are not inherent in the function of the service.

For example, the core function of a delivery service would be to deliver packages between any two arbitrary locations. This description directly implies that this service will not brew coffee, so this limitation is not a part of the scope. However, limitations on the size or weight of the packages or on the delivery locations can be expressed as service scope.

Formally, web service scope is a set of conditions over *scope parameters* which are any of the following three classes of data:

1. the *input parameters* of service invocations (like shipment address for an electronic shop)
2. *global parameters* like current time (for example for operating hours)
3. *external parameters* relevant for the service invocations (the bit rate of the video to be hosted, even though the hosting service does not request the bit rate value explicitly as one of its input parameters)

When scope is explicitly defined in a service description, a potential client can decide whether it is useful to invoke this particular service, i.e. whether the service can be expected to give positive result; or a client can see which of the available services are indeed currently applicable.

In general, service description languages must include the definitions of the input parameters (listed above as the first class of data that can be subject to scope limits). WSDL describes input parameters chiefly using XML Schema element declarations, i.e., it declares the parameters by name and data type. We can easily envision a mechanism for expressing conditions over the values of these elements, and we describe such a mechanism in section 3.

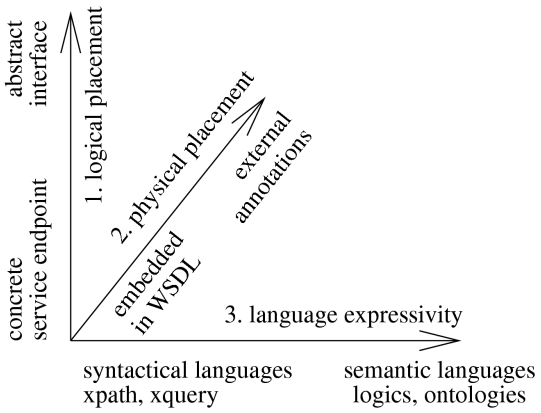

Fig. 1. Scope modeling choices

However, for the other two classes of scope-able data (global and external parameters), in order to express conditions we must also be able to declare them as *scope parameters*, additionally to the input parameters already present in the service description. Only then can we use a similar mechanism as mentioned above for limiting the values of these scope parameters.

There are three major axes of choices regarding how we model web service scope (shown in figure 1):

1. **Logical placement** — this axis represents the trade-off between reusability of the scope information and of the WSDL interfaces and bindings. In other words, should scope (along with scope parameter declarations) be attached to the abstract interface, concrete network binding or actual service endpoints?
2. **Physical placement** — here the trade-off is between the ease of use vs. manageability of multiple service and scope descriptions. Concretely, should the scope information be embedded in the service description or should it be expressed as external annotations in a different document?
3. **Scoping language expressivity** — we have to weigh the benefits and disadvantages of various data constraint languages. On the syntactic end of the expressivity spectrum we find XPath [17] expressions and on the semantic end there are logical languages with fully expressive ontologies. Should the scoping conditions be expressed in a syntactic or semantic way?

We describe our concrete approach in section 3. It covers all the options on the first axis (i.e., the scope annotations can be put on any level within the WSDL files); on the second axis we choose to embed scope information within WSDL documents; and along the third axis we show a concrete mechanism for using XPath expressions. In the subsections below, we analyze briefly the other options along axes 2 and 3.

2.1 Axis 2: External vs. embedded scope annotations

In the two options on the second axis, the scope information can be put inside a WSDL file (like we show in section 3), or it can reside in a separate document that points to the appropriate WSDL file(s).

Putting the annotations inside a WSDL file is realized by adding extension elements to the WSDL data. In this way, the scope information is closest to the description of the particular web service, so it is trivially accessible to any software that processes WSDL. Moreover, when both the service description and the scope annotations are in a single document, it is easier to manage the information and keep it up to date and consistent.

Making scope annotations external, on the other hand, allows having the scope information for multiple services in a single document, which may be useful for example when all the services of a particular company have the same operating hours. External annotations also make it easier to specify the scope of services whose WSDL description is generated dynamically by the infrastructure in which the services are deployed (e.g. an application server)[1].

Both approaches clearly have their advantages. In this work, we choose to embed semantic annotations inside WSDL documents due to the simplicity of this approach, however, in future work we plan to investigate putting scope annotations in web service policies (cf. WS-Policy Framework [3]). In fact, we expect that our WSDL extensions can be translated into policy assertions fairly easily.

2.2 Axis 3: Semantic scope annotations

Conceptually, WSDL describes web service input parameters with XML data types (by specifying the XML Schema). Global or external parameters (as described earlier in this section) can also be viewed as XML data types. Using XML languages like XPath for scoping expressions seems like a natural choice, especially for simple conditions like a time range or a maximum package weight.

However, often the values of scope parameters are not in a simple interval, rather they come from complex structured knowledge bases, such as address and location databases. It would be impractical, for example, to formulate a condition like "address is in mainland USA" with XPath over the string representation of an address. In contrast, it would be easy to express the condition using knowledge representation and reasoning techniques (ontologies, knowledge bases, logical inference and query engines).

To illustrate the difference, let us assume that we have a service with delivery limited to mainland USA, and that the address data structure contains a field `state` with the code of the US state. Lexically (in XPath), we could enumerate the 48 contiguous states (USA states except Alaska and Hawaii):

```
<scope language="http://www.w3.org/TR/xpath">
  state="AL" or state="AR" or state="AZ" or state="CA" or ...
</scope>
```

[1] Some web service providers are concerned about changing their WSDL documents, as they re-generate them every time the implementation is updated.

Using a semantic approach, the ontology can define a term MainlandUSA and say that the 48 states are in it, and then the condition would be simply

```
<scope language="http://wsmo.org/TR/wsmo">
  is_in(?state, MainlandUSA)
</scope>
```

While both approaches do have to make 48 statements (one for each state), with the semantic approach we can "store" all those statements in the ontology as the definition of the term MainlandUSA (which is similar to "stored queries" in databases), whereas in XPath we would have to repeat the whole list every time we want to express this condition[2]. Storing the expression in the ontology also helps with maintenance, e.g. if California decides to split into North California and South California, updating all the occurrences of the full XPath condition would be an expensive and error-prone task, as opposed to that of updating the (presumably single) ontology.

On top of "stored queries" with their various benefits, ontologies provide many useful mechanisms like subsumption hierarchies (BMW325Ci is_a Car, Car is_a Product), transitivity (SanDiego is_in California, California is_in MainlandUSA, therefore automatically SanDiego is_in MainlandUSA) and others.

We demonstrated that semantic scoping expressions are desirable, but we have so far neglected the point that semantic expressions manipulate semantic data, whereas WSDL treats all data as XML. Due to the nature of XML, there cannot be a single automatically generated mapping between XML and semantic data[3]. In general, the mapping must be tailored for each specific XML language (XML Schema) and each specific ontology. We use the term *data grounding* for mapping between XML and semantic data. There is a number of approaches for data grounding (see [7]), but they have one thing in common: there is generally an explicit mapping specified for any given pair of XML schema and ontology. If we want to add semantic scope annotations to WSDL, we also need to specify the mapping so that, at run time, the client with concrete data can evaluate it against the semantic expressions for scope.

This paper does not propose any concrete data grounding mechanism — that is an ongoing research in a different context[4]. Only for illustration, the simplest (and most limited) way to do data grounding would be a straightforward annotation of the XML Schema with matching ontology class or property identifiers, using an attribute like the `sem:class` in figure 2.

3 Scope Annotations in WSDL

In this section, we present our initial approach for annotating WSDL web service descriptions with scope information. We introduce two annotation elements, `scope` and

[2] This is because syntactical languages like XPath do not allow external (imported) expressions, and all-purpose inclusion tools like XInclude [16] are not generally supported yet.

[3] Semantic data is also called knowledge bases, ontology instances etc.

[4] See for example WSMO Grounding [6], or the Semantic Annotations for WSDL Working Group at W3C: http://www.w3.org/2002/ws/sawsdl/

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:sem="http://semantic.example.org/grounding">
  <xs:complexType name="address" sem:class="AddressOntology#Address">
    <xs:sequence>
      <xs:element name="street" type="xs:string"
                  sem:class="AddressOntology#Street"/>
      <xs:element name="state" type="xs:string"
                  sem:class="USAOntology#State"/>
      <!-- other subelements omitted -->
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Fig. 2. Illustration of a straightforward but limited approach to data grounding

`additionalScopeParameter`, but first we discuss how and where such annotations can be placed in WSDL documents so that a client can process them as scope information.

3.1 Placement of scope annotations in WSDL

WSDL describes web services in three layers: an abstract interface, a network binding and a concrete service.

At first, WSDL defines an abstract *interface* consisting of operations, where each operation is a simple message exchange, and each message is described as an XML element. Interfaces are meant to be reusable, i.e. services with the same functionality should optimally be described by a single interface, so that the clients need not change if they decide to switch between the available services.

The second layer is a so-called *binding*. With the same structure as an interface, a WSDL binding specifies what the messages from interface operations will look like on the wire, i.e. how they are serialized into actual bits and bytes to be transmitted between the client and the service — for example, the typical binding would specify that the messages are carried as payload of SOAP Envelopes [11] over HTTP [5]. In addition to networking details, a binding can specify the use of extensions, for example for security, reliable or asynchronous delivery. Binding can also be reused among different services, but some web service tools[5] allow the client to adapt to an actual binding at invocation time, so binding reusability carries lesser practical value.

The third layer of WSDL description is an actual concrete *service*. Every web service specifies what interface it implements, and it provides a number of *endpoints* — network addresses where the service can be accessed. Each endpoint can have a different binding, which allows a service to be accessible over different transports, or with differing transport-level features.

WSDL is a very extensible XML language, i.e., elements and attributes from foreign namespace can be placed almost anywhere in a WSDL file, and unless an extension is marked as mandatory, a WSDL processor is free to ignore extensions that it does not recognize. This level or extensibility allows us to consider placing scope annotations

[5] For instance the Web Service Invocation Framework (WSIF), http://ws.apache.org/wsif/

in all the layers of a WSDL description (this is captured in axis 1 in section 2). In particular, we can limit the scope on an interface for all services with this interface; on a binding that can also be shared by multiple services; or on the concrete service to specify the particular scope of this particular service. The placement of scoping information effectively depends on how many services will share any particular annotations. As scope is often one of the main parameters that differentiates services with otherwise the same functionality, we expect that there will be few use cases for scope annotations on the WSDL interface layer.

Until now we have talked primarily of *service scope*, taking service as something atomic. However, as WSDL defines a web service interface as a group of operations, we must examine whether different operations may have different scopes. Indeed, for example an electronic shop service that only ships products within Canada and mainland USA can still have a catalog inquiry operation that does not share that scope limitation because it only produces information that can be electronically delivered to any client, independent of where they are located. Therefore, our annotation mechanism should be able to specify different scope information for different service operations.

Combining the conclusions of the previous two paragraphs, we see that we should particularly support scope annotations on WSDL binding operations. However, we do not intend to introduce artificial limitations on where scope information can be placed.

There are two kinds of scope annotations — defining the actual scope conditions, and declaring additional (external) parameters that can be subject to scoping. These two kinds are described in the following subsections.

3.2 `sc:scope` annotation element

To express service scoping conditions, like a weight limit for a package delivery service, we introduce the XML element `sc:scope` with the following structure:

```
<sc:scope language="lang-uri">
  scope expression
</sc:scope>
```

The value of the `language` attribute is used to indicate (by means of a URI) the language in which the scope expression is written, such as XPath [17] or XQuery [18], or an ontological query language like WSML [2] axioms.

To illustrate scope annotations, in figure 3 on lines 63-66 we use XPath[6] to specify that a train reservation service can only reserve trips that begin or end in Austria.

The `sc:scope` element can be included within the WSDL elements `interface`, `operation`, `binding`, `service` and `endpoint`. When specified on an operation (either in an interface or in a binding), scope is evaluated against concrete input message data (and any additional scope parameters, as discussed below in section 3.3). The client forms the message which it would send to the service, and evaluates the scope condition against this message. In this way the client can determine that a particular operation on a particular service is not suitable for the particular inputs, and the client can choose an alternative service or an alternative operation to invoke instead.

[6] XPath is identified with the URI of the W3C recommendation, http://www.w3.org/TR/xpath

```
01    <description
02        xmlns="http://www.w3.org/2006/01/wsdl"
03        targetNamespace="http://trains.example.com/"
04        xmlns:tns="http://trains.example.com/"
05        xmlns:wsoap="http://www.w3.org/2006/01/wsdl/soap"
06        xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
07        xmlns:sc="http://context-aware.org/wsdl-annotations">
08
09      <types>
10        <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
11            targetNamespace="http://trains.example.com/"
12            xmlns="http://trains.example.com/">
13
14          <xs:complexType name="station-type">
15            <xs:sequence>
16              <xs:element name="station-name"  type="xs:string"/>
17              <xs:element name="country"  type="xs:string"/>
18            </xs:sequence>
19          </xs:complexType>
20
21          <xs:element name="ticket-reservation-request">
22            <xs:complexType>
23              <xs:sequence>
24                <xs:element name="from"  type="station-type"/>
25                <xs:element name="to"  type="station-type"/>
26                <xs:choice>
27                  <xs:element name="departure"  type="xs:dateTime"/>
28                  <xs:element name="arrival"  type="xs:dateTime"/>
29                </xs:choice>
30              </xs:sequence>
31            </xs:complexType>
32          </xs:element>
33
34          <xs:element name="ticket-reservation">
35            <xs:complexType>
36              <xs:sequence>
37                <xs:element name="from"  type="station-type"/>       <!-- these elements contain -->
38                <xs:element name="to"  type="station-type"/>         <!-- the concrete details   -->
39                <xs:element name="departure"  type="xs:dateTime"/>   <!-- of the reserved trip   -->
40                <xs:element name="arrival"  type="xs:dateTime"/>
41                <xs:element name="change-in"  type="station-type"  minOccurs="0"  maxOccurs="unbounded"/>
42                <xs:element name="reservation-code"  type="xs:string"/>
43                <xs:element name="price"  type="xs:decimal"/>
44              </xs:sequence>
45            </xs:complexType>
46          </xs:element>
47
48        </xs:schema>
49      </types>
50
51      <interface  name="train-reservation">
52        <operation name="reserve-ticket"  pattern="http://www.w3.org/2004/03/wsdl/in-out">
53            <input  messageLabel="In"   element="tns:ticket-reservation-request"/>
54            <output messageLabel="Out"  element="tns:ticket-reservation"/>
55        </operation>
56      </interface>
57
58      <binding name="oebb-SOAP-binding"
59               type="http://www.w3.org/2006/01/wsdl/soap"
60               interface="tns:train-reservation"
61               wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
62        <operation ref="tns:reserve-ticket">
63          <sc:scope  language="http://www.w3.org/TR/xpath"   xmlns="http://trains.example.com/">
64            (/ticket-reservation-request/from/country='Austria') or
65            (/ticket-reservation-request/to/country='Austria')
66          </sc:scope>
67        </operation>
68      </binding>
69
70      <service name="OEBB"  interface="tns:train-reservation">
71         <endpoint name="reservation-endpoint"
72                   binding="tns:oebb-SOAP-binding"
73                   address="http://oebb.example.com/reservation">
74         </endpoint>
75      </service>
76    </description>
```

Fig. 3. The WSDL description of an example train reservation service with highlighted scope annotation on lines 63-66

Any scope information specified on an interface can be viewed as additional scope conditions on all the operations within that interface, and similarly the scope of a binding applies to all the operations in that binding. Analogously, a scope on a service applies to all the endpoints of that service, and scope of an endpoint applies to any invocations that use this particular endpoint.

While ultimately scope conditions give indication of whether any given invocation would fail, scope can be evaluated even before the client is committed to do an invocation, for example to select appropriate services as part of the general discovery process.

3.3 `sc:additionalScopeParameter` annotation element

As described in section 2, scope parameters can be the input data (specified within the WSDL operation construct), or additional global or external parameters. In this section we show how a WSDL service description can declare these additional global and external parameters. The client uses such additional scope parameter declarations to know what data it needs to have available to evaluate the scope conditions properly.

For declaring additional scope parameters, either global or external, we introduce the XML element `sc:additionalScopeParameter` with the following structure:

```
<sc:additionalScopeParameter element="element QName"/>
```

Similarly to `sc:scope`, also additional scope parameter declarations can be placed in the WSDL `interface`, `operation`, `binding`, `service` and `endpoint` elements. Additional scope parameters declared on an interface can be used in scope conditions on this interface, on all its operations, and also in scope conditions of bindings or services that refer to this interface. Endpoint scope conditions can further use any additional scope parameters declared on the binding used by this endpoint, and the parameters declared on the service to which the endpoint belongs.

The example in figure 4 shows a WSDL pizza delivery operation that declares current time (for simplicity, only the hour part of it) and defines that it does not operate between midnight and 6 o'clock in the morning.

```
<operation ref="tns:deliverPizza">
  <sc:additionalScopeParameter element="tns:currentTimeHour"/>
  <sc:scope language="http://www.w3.org/TR/xpath">
    (/tns:currentTimeHour >= 6) and
    (/tns:currentTimeHour < 24)
    <!-- the greater-than and less-then signs must be escaped in real XML -->
  </sc:scope>
</operation>
```

Fig. 4. Example of an additional scope parameter

4 Related Work

We do not claim to have invented the general concept of scopes. The IETF *Service Location Protocol* (SLP) [10] for instance already contains a scope mechanism to narrow down the set of possible services a SLP User Agent reports as available. The SLP scope mechanism is pretty simple: if a SLP User Agent's scope name does not match a scope name listed in a SLP service description, the service is not further considered. More complex scopes, even still simple number-based scopes, can not be expressed in SLP.

The concept of scopes applied to web services was first introduced in [13]. There it was used as a means to decide whether there are options to perform a hand-off[7] between web service providers while maintaining the same or at least very similar service functionality within a service session. This explains why web service scopes are often considered to be non-functional properties in semantic web service (SWS) descriptions. However, as we discuss in section 2, scope parameters can be both functional and non-functional; e.g. the *input parameters* are obviously functional parameters.

SWS descriptions generally include the preconditions of web services (cf. [4, 14]), and these conditions can also be used to express scope limitations. Our approach differs from SWS preconditions in two aspects: firstly, we pick out scope conditions, be they functional or non-functional, whereas SWS preconditions can include conditions that we would not classify as scope. The knowledge that a precondition is a scope limitation implies that there may be a possibility of service hand-off if the client falls out of the scope[8], whereas failing a general precondition may also mean that the client simply has inappropriate data, and it does not help the client handle the situation. Secondly, our approach allows us to avoid the complexity of use and reasoner implementation that is inherent in the logics-based semantic languages; we allow the use of simpler technologies like XPath where they are sufficient.

The most prominent scope of a service is the location scope — usually the geographical region where a service should be considered to be available. For most kinds of Internet information services, the location scope is usually quite irrelevant as the scope of such services is "global", meaning that the service (e.g. a currency converter service) is available everywhere. Services which depend on the location of the user, the location of the service provider or the location of one or more other entities (so called *Location Based Services* (LBS) [8]) usually do have some constraints regarding the location scope. A weather forecast service, for example, is constrained to the regions where the responsible provider maintains a network of weather stations (or contracts them from other parties).

But as shown earlier, services may be scoped to more than locations; in fact, any contextual aspect (dimension of a situation-space) can be used to scope a service. Examples are time (e.g. opening hours of a restaurant, see [12]), current weather [1], light conditions [9] etc. All these are prominent examples of aspects sensed in various ways in Ubiquitous Computing environments. Services that automatically adapt to changes in these environments (so called *Context-Aware Services*) are in the focus of ongo-

[7] Sometimes also called service handover or service compensation.

[8] When the client detects it is out of scope of a service, it may initiate hand-off or it may prepare for service breaks which can be detected by monitoring the scope.

ing research. Scoping context-aware web services is just the natural follow-up of that research, enhancing web services with context awareness in a well defined way as a milestone towards *Ubiquitous Services*.

5 Conclusions

Web services are usually described with Web Service Description Language (WSDL), which specifies the data and operation signatures. WSDL generally assumes that services are always available, and it does not provide a built-in mechanism for describing that services may have a *scope*.

In this paper, we describe the concept of web service scope and we propose two XML elements for annotating WSDL with service scope information. Our solution is extensible with respect to the language for expressing the scope conditions. This allows adoption of simple languages such as XPath, which is nevertheless quite powerful, widely known and implemented. On the other hand, our mechanism also allows the use of semantic languages in applications that require the full expressivity of logics. However, the use of semantic languages requires mapping between the XML data described in WSDL and some semantic model, which we call *data grounding*, but this part is outside the focus of our paper.

Automatically processable scope information allows the clients to evaluate whether they fall out of scope of any given service, in order to avoid unnecessary invocations that would only result in faults. This enables a set of concrete use cases like service hand-off or filtering during discovery. Annotating WSDL with scope information is, therefore, a step towards both ubiquitous computing and web service automation.

References

[1] Thomas Buchholz. *Scalable Context-Aware Services*. PhD thesis, LMU München, 2005.

[2] Jos de Bruijn et al. The Web Service Modeling Language WSML. Available at http://www.wsmo.org/TR/d16/d16.1/v0.3/20051005/, October 2005.

[3] Jeffrey Schlimmer (editor) et al. Web Services Policy 1.2 – Framework (WS-Policy). Technical note, April 2006. Available at http://www.w3.org/Submission/WS-Policy/.

[4] D. Roman et al. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[5] Hypertext Transfer Protocol – HTTP/1.1. Draft Internet Standard, IETF, June 1999. Available at http://rfc.net/rfc2616.html.

[6] Jacek Kopecký, Matthew Moran, Dumitru Roman, and Adrian Mocan. WSMO Grounding. Available at http://www.wsmo.org/TR/d24/d24.2/v0.1/20050916/, September 2005.

[7] Jacek Kopecký, Dumitru Roman, Matthew Moran, and Dieter Fensel. Semantic Web Services Grounding. In *Proc. of the Int'l Conference on Internet and Web Applications and Services (ICIW'06), Guadeloupe*, February 2006.

[8] Axel Küpper. *Location-based Services: Fundamentals and Operation*. John Wiley & Sons Ltd., 2005.

[9] Julian Randall, Oliver Amft, and Gerhard Tröster. Towards LuxTrace: Using Solar Cells to Measure Distance Indoors. In Thomas Strang and Claudia Linnhoff-Popien, editors, *LoCA*, volume 3479 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 2005.

[10] Service Location Protocol, Version 2. Draft Internet Standard, IETF, June 1999. Available at http://rfc.net/rfc2608.html.

[11] SOAP Version 1.2 Part 1: Messaging Framework. Recommendation, W3C, June 2003. Available at http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.

[12] Thomas Strang. *Service Interoperability in Ubiquitous Computing Environments*. PhD thesis, LMU München, 2004.

[13] Thomas Strang, Claudia Linnhoff-Popien, and Matthias Roeckl. Highlevel Service Handover through a Contextual Framework. In *MoMuC 2003, 8th International Workshop on Mobile Multimedia Communications, Munich/Germany*, October 2003.

[14] The OWL Services Coalition. OWL-S 1.1 Release. Available at http://www.daml.org/services/owl-s/1.1/, November 2004.

[15] Web Services Description Language (WSDL) Version 2.0. Candidate Recommendation, W3C, January 2006. Available at http://www.w3.org/TR/2006/CR-wsdl20-20060106/.

[16] XML Inclusions (XInclude) Version 1.0. Recommendation, W3C, December 2004. Available at http://www.w3.org/TR/xinclude/.

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

[18] XQuery 1.0: An XML Query Language. Recommendation, W3C, June 2006. Available at http://www.w3.org/TR/xquery.