Describing Web Applications

Marc **Hadley**

Abstract

An increasing number of Web based enterprises (Google, Yahoo, Amazon, Flickr to name but a few) are developing HTTP-based applications that provide access to their internal data using XML. Today, these applications are typically described using a combination of textual protocol descriptions combined with XML schema-based data format descriptions. Interestingly, the Web Service Description Language (WSDL) is rarely used to describe such applications and generally is only used where a parallel SOAP-based service is exposed. There are probably many reasons for this but chief amongst them must be the fact that WSDL is not a Web-oriented language and the WSDL HTTP binding appears to be something of an afterthought.

In reaction to this emerging trend, the W3C recently started a new mailing list (public-web-http-desc@w3.org) dedicated to “discussion of Web description languages based on URI/IRI and HTTP, and aligned with the Web and REST Architecture”. This paper frames the subject of Web description and describes one of the candidate XML languages, the Web Application Description Language (WADL), designed to provide a machine process-able protocol description format for use with HTTP-based Web applications, especially those using XML.

Table of Contents

1. Web Applications and Their Description

For the purposes of this paper, a Web application is defined as a dynamic HTTP-based application whose interactions are amenable to machine processing. While many existing Web sites are examples of dynamic HTTP-based applications, a large number of those require human cognitive function for successful non-brittle[1] use. Typically Web applications:

- Are based on existing Web architecture and infrastructure
- Are platform and programming language independent
- Promote re-use of application beyond the browser
- Enable composition with other Web or desktop applications
- Require semantic clarity in content exchanged during use

The latter requirement can be fulfilled by use of XML either by defining a complete custom schema for the application domain or embedding a custom micro-format in an existing schema using its extensibility points.

Given the above definition of a Web application one can see that the following aspects of an application could be usefully described in a machine processable format:

- List of resources

  Analogous to a site map showing the resources on offer.

- Relationships between resources.

  Describing the links between resources , both referential and causal.

- Methods that can be applied to each resource.

  The HTTP methods that can be applied to each resource, the expected inputs and outputs and their supported formats.

- Resource representation formats

  The supported MIME types and, where XML is used, the XML schemas in use.

2. Use Cases

The current state of the art in Web application description is textual documentation plus one or more XML schemata. Whilst entirely adequate for human consumption this level of description precludes many of the following use cases which require a more machine use-able description format:

- Application Modelling and Visualization

  Support for development of resource modelling tools for resource relationship and choreography analysis and manipulation.

- Code Generation

  Automated generation of stub and skeleton code and code for manipulation of resource representations.

[1] Brittle use, e.g. HTML page scraping, is generally always possible but less desirable in terms of maintenance.

- Configuration

  Configuration of client and server using a portable format.

It would also be good to have a common foundation for individual applications and protocols to re-use and perhaps extend rather than each inventing a new description format.

3. How To Describe Web Applications

3.1. WSDL

Versions 1.1[WSDL11] and 2.0[WSDL20] of the Web Services Description Language (WSDL) both contain bindings for HTTP. The WSDL 1.1 HTTP binding exhibits the following problems:

- Interface centricity

  WSDL is interface centric in its structure whereas the Web is URI centric. This structural mismatch makes a WSDL description of a Web application non-intuitive.

- Inflexibility

  - It only supports GET and POST
  - Ports are limited to a single HTTP method
  - URIs are limited to a single method
  - One can't mix input between generative URI path segments, query parameters and entity body

- Lack of implementations

  Possibly due to the other problems mentioned above.

WSDL 2.0 fixes a number of the problems that are present in the WSDL 1.1 HTTP binding but some remain and much of the new functionality is marked as “at risk” in the latest draft and may be removed before the document is completed.

3.2. Proposals

There have been a number of proposals for Web description languages, rather than list them all here, the reader is referred to an excellent summary of the space by Dion Hinchcliffe[SDLs].

3.3. Web Application Description Language

This section describes one of the proposals for a Web description language, the Web Application Description Language (WADL)[WADL]. This section provides a brief overview, for full details consult the WADL specification[WADL].

3.3.1. Guiding Principles

- HTTP only

  The language is designed to describe HTTP based applications only, additional levels of abstraction to support use with different protocols is not a goal.

- Resource centric

The language will treat resources as first class objects.

- Flexible

  The language will support the full set of HTTP capabilities.

- Schema language independent

  The language will not be tied to any particular XML schema language.

- Composable

  The language will support composition of descriptions from multiple components.

- Easy to parse

  The language should be easy to parse/read for both machines and humans.

- Don't try to describe HTTP

  Avoid duplicating information from the HTTP specification, only describe the application built on HTTP.

- Keep it simple

  Focus on basic description for now, ignore process, flow and choreography modelling

3.3.2. Structure

A WADL document follows the following structure:

```
<application>
  <grammars/>?
  <resources base='anyURI'>?
    <resource uri='anyURI'?>+
      <path_variable name='NMTOKEN' type='QName'?>?
      ( <method/> | <resource/> )+
    </resource>
  </resources>
  ( <method/> | <representation/> | <fault> )*
</application>
```

The resources element defines the base URI for its child resource elements. Note that resource elements can be arbitrarily nested and that each is relative to its parent. A resource element must either have a uri attribute or a child path_variable element, the values of these two constructs provide the relative URI for the resource.

The grammars element is a container for XML schemas, it can either contain embedded schema documents or references to external schemas as shown below:

```
<wadl:grammars>
  <include href="xs:anyURI"/>*
  <xs:any/>*
</wadl:grammars>
```

The WADL specification[WADL] defines bindings for W3C XML Schema[XSD] and RelaxNG[RELAXNG] but can support any schema language that defines element structures in terms of namespace qualified names.

Each resource element can have one or more child method elements that define the methods that can beapplied to that resource. The method element has the following structure:

```
<method name='NMTOKEN'? id='ID'? href='anyURI'?>
  <request>?
    <representation/>*
    <query_variable name='NMTOKEN' type='QName'?
      required='boolean'? repeating='boolean'? fixed='string'?>*
  </request>
  <response>?
    ( <representation/> | <fault/> )*
  </response>
</method>
```

Methods can either embed their information or reference globally defined elements using the href attribute. Method requests and responses can contain one or more representation child elements. Sibling representation elements define a set of possible alternative representations for a resource. A representation element has the following structure:

```
<representation id='ID'? mediaType='string'? element='QName'?
  href='anyURI'?>
  <representation_variable name='NMTOKEN' type='QName'?
    path='string'? required='boolean'? repeating='boolean'?
    fixed='string'?/>*
</representation>
```

Child representation_variable elements provide additional information about the content of a representation. They are particularly useful in combination with the application/x-www-form-urlencoded and multipart/form-data media types where they define the fields contained within the representation.

3.3.3. Example: Yahoo News Search

The following shows a WADL description of the Yahoo! news search service[YAHOO]:

```
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://research.sun.com/wadl wadl.xsd"
  xmlns:tns="urn:yahoo:yn"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:yn="urn:yahoo:yn"
  xmlns:ya="urn:yahoo:api"
  xmlns="http://research.sun.com/wadl">

  <grammars>
    <include href="NewsSearchResponse.xsd"/>
    <include href="http://api.search.yahoo.com/Api/V1/error.xsd"/>
  </grammars>

  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource uri="newsSearch">
      <method name="GET" id="NewsSearch">
        <request>
          <query_variable name="appid" type="xsd:string" required="true"/>
          <query_variable name="query" type="xsd:string" required="true"/>
```

```
          <query_variable name="type" type="xsd:string"/>
          <query_variable name="results" type="xsd:int"/>
          <query_variable name="start" type="xsd:int"/>
          <query_variable name="sort" type="xsd:string"/>
          <query_variable name="language" type="xsd:string"/>
        </request>
        <response>
          <representation mediaType="application/xml"
            element="yn:ResultSet"/>
          <fault id="SearchError" status="400" mediaType="application/xml"
            element="ya:Error"/>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

Note that two of the query_variable elements have the required attribute set to true, these query variables must be supplied each time the method is invoked, the other query variables are all optional.

3.3.4. Example: Amazon Item Search

The following shows a WADL description of the Amazon item search service[AMAZON]:

```
<application xmlns="http://research.sun.com/wadl"
  xmlns:aws="http://webservices.amazon.com/AWSECommerceService/2005-07-26"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <grammars>
    <include href="AWSECommerceService.xsd"/>
  </grammars>

  <resources base="http://webservices.amazon.com/onca/">
    <resource uri="xml">
      <method href="#ItemSearch"/>
    </resource>
  </resources>

  <method name="GET" id="ItemSearch">
    <request>
      <query_variable name="Service" fixed="AWSECommerceService"/>
      <query_variable name="Version" fixed="2005-07-26"/>
      <query_variable name="Operation" fixed="ItemSearch"/>
      <query_variable name="SubscriptionId" type="xsd:string"
        required="true"/>
      <query_variable name="SearchIndex" type="aws:SearchIndexType"
        required="true"/>
      <query_variable name="Keywords" type="aws:KeywordList"
        required="true"/>
      <query_variable name="ResponseGroup" type="aws:ResponseGroupType"/>
    </request>
    <response>
      <representation mediaType="text/xml"
```

```
        element="aws:ItemSearchResponse"/>
    </response>
  </method>

</application>
```

Note the following:

- The method is attached to the resource as a reference to a globally defined method rather than being embedded directly as in the previous example. In this instance there is no need to do this beyond illustrating the capability but this is useful where one method can be applied to multiple resources.

- A number of the query variables are marked as fixed value. The Amazon API use query parameters to identify services and operations within those services, use of the fixed attribute can be used to allow description of multiple logical methods on the same resource. Without the ability to fix values in this way, the Amazon API in would look like one single method with many parameters.

3.3.5. Example: Atom Publishing Protocol

The Atom publishing protocol[ATOM] defines a set of methods to introspect, view and update entries in an Atom feed. The publishing protocol is bootstrapped using introspection on a known URI for a particular set of feeds, this process is described in the following WADL document:

```
<application xmlns="http://research.sun.com/wadl"
  xmlns:app="http://purl.org/atom/app#"
  xmlns:atom="http://www.w3.org/2005/Atom">
  <grammars>
    <include href="http://purl.org/atom/app.xsd"/>
  </grammars>

  <representation id="service" mediaType="application/atomserv+xml"
    element="app:service"/>

  <method name="GET" id="getService">
    <response>
      <representation href="#service"/>
    </response>
  </method>
</application>
```

Note the absence of a resources element, this is omitted since the document above defines a method that is detached from any specific Web site. WADL descriptions of a particular Web site can re-use the method definition for the resources it offers. The “getService” method response consists of an application application/atomserv+xml document that describes the available feeds. An example of such is shown below:

```
<service xmlns="http://purl.org/atom/app#">
  <workspace title="Main Site" >
    <collection
      title="My Blog Entries"
      href="http://example.org/reilly/main" >
      <member-type>entry</member-type>
      <list-template>http://example.org/{index}</list-template>
    </collection>
```

```
    <collection
      title="Pictures"
      href="http://example.org/reilly/pic" >
      <member-type>media</member-type>
      <list-template>http://example.org/p/{index}</list-template>
    </collection>
  </workspace>
</service>
```

Note the similarity between the Atom service document and WADL, both describe a set of resources and methods that may be applied to them. In the case of an Atom service document the applicable methods are implicit based on the member-type of a collection. An Atom service document also defines some additional metadata (the feed title) specific to the protocol domain. One could replicate the information in an Atom service document using WADL as follows.

The first step is to create a WADL document that contains all of the Atom protocol methods and associated representations, this only needs to be done once since the contents of this document can then be re-used by WADL documents specific to each site.

```
<application xmlns="http://research.sun.com/wadl"
  xmlns:app="http://purl.org/atom/app#"
  xmlns:atom="http://www.w3.org/2005/Atom">

  <grammars>
    <include href="http://purl.org/atom/app.xsd"/>
  </grammars>

  <representation id="entry" mediaType="application/atom+xml"
    element="atom:entry"/>

  <representation id="feed" mediaType="application/atom+xml"
    element="atom:feed"/>

  <method name="GET" id="getFeed">
    <response>
      <representation href="#feed"/>
    </response>
  </method>

  <method name="POST" id="addEntryCollectionMember">
    <request>
      <representation href="#entry"/>
    </request>
  </method>

  <method name="POST" id="addGenericCollectionMember">
    <request>
      <representation href="#entry"/>
      <representation />
    </request>
  </method>

  <method name="DELETE" id="deleteCollectionMember"/>
```

```
  <method name="GET" id="readEntryCollectionMember">
    <response>
      <representation href="#entry"/>
    </response>
  </method>

  <method name="GET" id="readGenericCollectionMember">
    <response>
      <representation href="#entry"/>
      <representation />
    </response>
  </method>

  <method name="PUT" id="updateEntryCollectionMember">
    <request>
      <representation href="#entry"/>
    </request>
    <response>
      <representation href="#entry"/>
    </response>
  </method>

  <method name="PUT" id="updateGenericCollectionMember">
    <request>
      <representation href="#entry"/>
      <representation />
    </request>
    <response>
      <representation href="#entry"/>
      <representation />
    </response>
  </method>

</application>
```

Given the preceding document, one can create a WADL version of the prior Atom service document:

```
<application xmlns="http://research.sun.com/wadl"
  xml:base="http://purl.org/atom/app.wadl"
  xmlns:app="http://purl.org/atom/app#">

  <resources base="http://example.org/">
    <resource uri="reilly">
      <resource uri="main" app:member-type="entry"
        app:title="My Blog Entries">
        <method href="#getFeed"/>
        <method href="#addEntryCollectionMember"/>
        <resource>
          <path_variable name="range" type="app:indexRange"/>
          <method href="#getFeed"/>
        </resource>
        <resource>
          <path_variable name="entryId"/>
```

```
          <method href="#readEntryCollectionMember"/>
          <method href="#deleteCollectionMember"/>
          <method href="#updateEntryCollectionMember"/>
        </resource>
      </resource>
    </resource>
  </resources>
</application>
```

Note the use of the xml:base attribute to allow use of relative URIs in method references. The above WADL document describes three resources:

- http://example.org/reilly/main

  This resource supports HTTP GET to retrieve an Atom feed document and HTTP POST to add a new entry to the feed.

- http://example.org/reilly/main/{range}

  Where {range} is a generative path segment that allows selection of a range of entries from the feed. This resource supports HTTP GET to retrieve an Atom feed document.

- http://example.org/reilly/main/{entryId}

  Where {entryId} is a generative path segment that allows selections of a particular entry in the feed. This resource supports HTTP GET to retrieve an Atom entry document, HTTP PUT to replace an Atom entry in the feed and HTTP DELETE to remove an entry from the feed.

The above document also includes some Atom-specific extension elements to provide the same metadata as the prior Atom service document.

Bibliography

[WSDL11] *Web Services Description Language (WSDL) 1.1*, E. Christensen, et al, W3C, March 2001. Available at http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[WSDL20] *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, R. Chinnici, et al, W3C, May 2005. Available at http://www.w3.org/TR/2005/WD-wsdl20-20050510

[SDLs] *SDLs Continued: Finding the Value Proposition in Describing Web Services*, D Hinchcliffe, May 2005. Available at http://hinchcliffe.org/archive/2005/05/16/231.aspx

[WADL] *Web Application Description Language (WADL)*, M Hadley, Sun Microsystems, Inc, November 2005. Available at http://weblogs.java.net/blog/mhadley/wadl.pdf

[XSD] *XML Schema Part 1: Structures Second Edition*, H Thompson, et al, W3C, October 2004. Available at http://www.w3.org/TR/xmlschema-1/

[RELAXNG] *RELAX NG Specification*, J Clark, et al, OASIS, December 2001. Available at http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

[YAHOO] *Yahoo! Web APIs*, Yahoo!, 2005. Available at http://developer.yahoo.net/

[AMAZON] *Amazon Web Services*, Amazon.com, 2005. Available at http://www.amazon.com/

[ATOM] *The Atom Publishing Protocol*, J Gregorio, et al, IETF, Nov 2005. Available at http://bitworking.org/projects/atom/draft-ietf-atompub-protocol-06.html

Biography

Marc **Hadley**
Senior Staff Engineer
Sun Microsystems, Inc. [http://www.sun.com]
Burlington
Massachusetts
United States of America

Dr. Marc J. Hadley is a Senior Staff Engineer in the Web Technologies and Standards division of Sun Microsystems. He currently represents Sun on the W3C XML Protocol Working Group where he is co-editor of the SOAP 1.2 specification. Marc is the technical lead for Sun's participation at the Web Services Interoperability Organisation (WS-I) and is co-spec lead of the JAX-RPC 2.0 specification currently under development in the JCP. Marc has presented at numerous conferences including XML 200X, XML World, XML One, JavaOne and the O'Reilly Conference on Enterprise Java.Prior to joining Sun Microsystems, Marc worked for Chrystal Software as a principle consultant and chief engineer developing content management and Web delivery systems based on XML.Marc has more than ten years experience in the software industry and holds a D.Phil in Physics from the University of York.