Web Application Description Language (WADL)

Marc J. Hadley, Sun Microsystems Inc.

August 2, 2006

Abstract

This specification describes the Web Application Description Language (WADL). An increasing number of Web-based enterprises (Google, Yahoo, Amazon, Flickr to name but a few) are developing HTTP-based applications that provide programatic access to their internal data. Typically these applications are described using textual documentation that is sometimes supplemented with more formal specifications such as XML schema for XML-based data formats. WADL is designed to provide a machine process-able protocol description format for use with such HTTP-based Web applications.

1 Introduction

This specification describes the Web Application Description Language (WADL). WADL is designed to provide a machine process-able protocol description format for use with HTTP-based Web applications.

1.1 Web Applications

For the purposes of this article, a Web application is defined as a HTTP-based application whose interactions are amenable to machine processing. While many existing Web sites are examples of 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 the application beyond the browser

- Enable composition with other Web or desktop applications

---

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

- Require semantic clarity in content (representations) exchanged during their use

The latter requirement can be fulfilled by the 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:

Set 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 any XML schemas in use.

1.2 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 the following use cases which require a more machine usable 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.

Configuration Configuration of client and server using a portable format.

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

1.3 Example WADL Description

The following listing shows an example of a WADL description for the Yahoo News Search[1] application.

```
1   <?xml version="1.0"?>
2   <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://research.sun.com/wadl/2006/07 wadl.xsd"
4    xmlns:tns="urn:yahoo:yn"
5    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
 6    xmlns:yn="urn:yahoo:yn"
 7    xmlns:ya="urn:yahoo:api"
 8    xmlns="http://research.sun.com/wadl/2006/07">
 9     <grammars>
10       <include
11         href="NewsSearchResponse.xsd"/>
12       <include
13         href="Error.xsd"/>
14     </grammars>
15
16     <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
17       <resource path="newsSearch">
18         <method name="GET" id="search">
19           <request>
20             <param name="appid" type="xsd:string" required="true"/>
21             <param name="query" type="xsd:string" required="true"/>
22             <param name="type" default="all">
23               <option value="all"/>
24               <option value="any"/>
25               <option value="phrase"/>
26             </param>
27             <param name="results" type="xsd:int" default="10"/>
28             <param name="start" type="xsd:int" default="1"/>
29             <param name="sort" default="rank">
30               <option value="rank"/>
31               <option value="date"/>
32             </param>
33             <param name="language" type="xsd:string"/>
34           </request>
35           <response>
36             <representation mediaType="application/xml"
37               element="yn:ResultSet"/>
38             <fault status="400" mediaType="application/xml"
39               element="ya:Error"/>
40           </response>
41         </method>
42       </resource>
43     </resources>
44
45   </application>
```

Lines 2–8 begin an application description and define the XML namespaces used elsewhere in the service description. Lines 9–14 define the XML grammars used by the service, in this case two W3C XML Schema files are included by reference. Lines 16–43 describe the Yahoo News Search Web resource and the HTTP methods it supports. Lines 18–40 describe the 'search' GET method: lines 19–34 describe the input; lines 35–40 describe the possible outputs.

2 Description Components

All WADL elements have the following XML namespace name:

- `http://research.sun.com/wadl/2006/07`

This section describes each component of a WADL document in detail.

2.1 Application

The `application` element forms the root of a WADL description and contains the following:

1. Zero or more `doc` elements – see section 2.2.

2. An optional `grammars` element – see section 2.3.

3. An optional `resources` element – see section 2.4.

4. Zero or more of the following:

   - `resource` elements – see section 2.5.
   - `method` elements – see section 2.6.
   - `representation` elements – see section 2.9.
   - `fault` elements – see section 2.10.

2.2 Documentation

Each WADL-defined element can have one or more child `doc` elements that can be used to document that element. The `doc` element has the following attributes:

xml:lang Defines the language for the `title` attribute value and the contents of the `doc` element. If an element contains more than one `doc` element then they MUST have distinct values for their `xml:lang` attribute.

title A short plain text description of the element being documented, the value SHOULD be suitable for use as a title for the contained documentation.

The `doc` element has mixed content and may contain text and zero or more child elements that form the body of the documentation. It is RECOMMENDED that the child elements be members of the text, list or table modules of XHTML[2].

2.3 Grammars

The grammars element acts as a container for definitions of the format of data exchanged during execution of the protocol described by the WADL document. Such definitions may be included inline or by reference using the include element (see section 2.3.1). No particular data format definition language language is mandated; sections 3 and 4 describe use of RelaxNG and W3C XML Schema with WADL, respectively.

It is permissible to include multiple definitions of a particular format: such definitions are assumed to be equivalent and consumers of a WADL description are free to choose amongst the alternatives or even combine them if they support that capability.

2.3.1 Include

The include element allows the definitions of one or more data format descriptions to be included by reference. The href attribute provides a URI for the referenced definitions and is of type xsd:anyURI. Use of the include element is logically equivalent to in-lining the referenced document within the WADL grammars element.

2.4 Resources

The resources element acts as a container for the resources provided by the application. A resources element has a base attribute of type xsd:anyURI that provides the base URI for each child resource identifier. Descendent resource elements (see section 2.5) each describe a single resource provided by the application.

2.5 Resource

A resource element describes a single resource, identified by a URI, provided by the Web application. A resource element can either be a resource definition or a reference to a resource defined elsewhere.

2.5.1 Resource Reference

A resource reference element has a href attribute of type xsd:anyURI. The value of the href attribute is a URI reference to a resource definition element. A resource reference element MAY have zero or more child param elements (see section 2.11) and MUST NOT have any other WADL-defined attributes or contain any other WADL-defined child elements.

This form of resource element may be used to reduce duplication when the same resource hierarchy is offered at multiple URIs.

2.5.2 Resource Definition

A `resource` definition element has the following attributes:

`id` An optional identifier of the resource definition, required for globally defined resources, not allowed on locally embedded resources. Resource definitions are identified by an XML ID and are referred to using a URI reference.

`path` An optional attribute of type `xsd:string`. If present, it provides a template for the identifier of the resource as a relative URI whose base URI is given by the `resource` element's parent `resource` or `resources` element.

The value of the `path` attribute may be static or may contain embedded path parameters. A path parameter is represented within the `path` value as '{' *name* '}' where *name* is the name of the parameter. At runtime, the values of path parameters are substituted into the resource identifier when the resource is used, see section 2.5.4 for a detailed example.

Embedded path parameters have limited expressivity; for more complex generative URIs, use of the `param` element (see section 2.11) should be considered: a child `param` element whose `name` attribute value matches the *name* of an embedded path parameter provides additional information about that path parameter.

A `resource` definition element contains the following child elements:

- Zero or more `doc` elements – see section 2.2.

- Zero or more `param` elements (see section 2.11), each of which define parameterized components of the `resource` element's URI identifier.

- Zero or more `method` (see section 2.6) elements, each of which describes the input to and output from an HTTP protocol method that can be applied to the resource.

- Zero or more `resource` elements that describe sub-resources.

2.5.3 Path Parameters

Child `param` elements (see section 2.11) may be used to parameterize the identifier of their parent `resource` element. If the value of the `name` attribute of a child `param` element matches the name of a path parameter embedded in the parent resource's `path` attribute then the `param` element contains additional information about the embedded parameter rather than declaring a new path parameter.

2.5.4 Generating Resource Identifiers

The URI for a `resource` element is obtained using the following rules:

1. If the resource element is a reference then substitute the referenced element

2. Set *identifier* equal to the URI computed (using this process) for the parent element (`resource` or `resources`)

3. If *identifier* doesn't end with a '/' then append a '/' character to *identifier*

4. Substitute the values of any embedded path parameters into the value of the `path` attribute

5. Append the value obtained in the previous step to *identifier*

6. For each child `param` element (see section 2.11), in document order, that has a value of 'matrix' for its `style` attribute, append a representation of the parameter value to *identifier* according to the following rules:

   - Non-boolean matrix parameters are represented as: ';' *name* '=' *value*
   - Boolean matrix parameters are represented as: ';' *name* when *value* is `true` and are omitted from *identifier* when *value* is `false`

   where *name* is the value of the `param` element's `name` attribute and *value* is the runtime value of the parameter.

The following example illustrates these rules and shows an extract from a Web application description that provides multiple resources:

```
1   <resources base="http://example.com/">
2     <resource path="widgets">
3       <resource path="reports/stock">
4         <param name="instockonly" style="matrix"
5           type="xsd:boolean"/>
6         ...
7       </resource>
8       <resource path="{widgetId}">
9         ...
10      </resource>
11      ...
12    </resource>
13    <resource path="accounts/{accountId}">
14      ...
15    </resource>
16  </resources>
```

The above describes the following resources:

- A resource identified by a static URI: http://example.com/widgets
- A resource identified by a static URI: http://example.com/widgets/reports/stock
- A resource identified by a matrix URI: http://example.com/widgets/reports/stock;instockonly

- Multiple resources identified by generative URIs: http://example.com/widgets/*widgetId*, where the *widgetId* component of the URI is replaced at runtime with the value of a runtime parameter called widgetId.

- Multiple resources identified by generative URIs: http://example.com/accounts/*accountId*, where the *accountId* component of the URI is replaced at runtime with the value of a runtime parameter called accountId.

2.6 Method

A `method` element describes the input to and output from an HTTP protocol method that may be applied to a resource. A `method` element can either be a method definition or a reference to a method defined elsewhere.

2.6.1 Method Reference

A `method` reference element is a child of a `resource` element that has an `href` attribute whose type is `xsd:anyURI`. The value of the `href` attribute is a URI reference to a `method` definition element. A `method` reference element MUST NOT have any other WADL-defined attributes or contain any WADL-defined child elements.

This form of `method` element may be used to reduce duplication when the same method applies to more than one resource.

2.6.2 Method Definition

A `method` definition element is a child of a `resource` or `application` element and has the following attributes:

name Indicates the HTTP method used.

id An identifier for the method, required for globally defined methods, not allowed on locally embedded methods. Methods are identified by an XML ID and are referred to using a URI reference.

It is permissible to have multiple child `method` elements that have the same value of the `name` attribute for a given resource; such siblings represent distinct variations of the same HTTP method and will typically have different input data.

A `method` element has the following child elements:

doc Zero or more `doc` elements – see section 2.2.

request Describes the input to the method as a collection of parameters and an optional resource representation – see section 2.7.

response Describes the output of the method as a collection of alternate resource representations – see section 2.8.

2.7 Request

A `request` element describes the input to be included when applying an HTTP method to a resource. A `request` element has no attributes and may contain the following child elements:

1. Zero or more `doc` elements – see section 2.2.

2. Zero or more `representation` elements – see section 2.9. Note that use of `representation` elements is confined to HTTP methods that accept an entity body in the request (e.g., PUT or POST). Sibling `representation` elements represent logically equivalent alternatives, e.g., a particular resource might support multiple XML grammars for a particular request.

3. Zero or more `param` elements – see sections 2.7.1 and 2.11.

2.7.1 Query Parameters

Child `param` elements (see section 2.11) represent URI query parameters as described in section 17.13 of HTML 4.01[3]. The runtime values of query parameters are sent as URI query parameters when the HTTP method is invoked.

The following example shows a resource with a generative URI that supports a single HTTP method with a single optional query parameter:

```
1   <resources base="http://example.com/widgets">
2     <resource path="{widgetId}">
3       <method name="GET">
4         <request>
5           <param name="verbose" type="xsd:boolean"/>
6         </request>
7         <response>
8            ...
9         </response>
10      </method>
11    </resource>
12  </resources>
```

If the value of the `widgetId` parameter is '1234567890' and the value of the `verbose` parameter is 'true' then the URI on which the HTTP GET will be performed is:

9

```
http://example.com/widgets/1234567890?verbose=true
```

2.8   Response

A `response` element describes the output that results from performing an HTTP method on a resource. It may contain the following child elements:

- Zero or more `doc` elements – see section 2.2.

- Zero or more `representation` elements - see section 2.9.

- Zero or more `fault` elements - see section 2.10.

Each child `representation` element describes a resource representation that may result from performing the method. Sibling `representation` elements indicate logically equivalent alternatives; normal HTTP mechanisms may be used to select a particular alternative. Each child `fault` element describes a fault condition that may occur – note that not all possible fault conditions are likely to be described and client applications should be prepared to handle the full range of possible HTTP error conditions.

2.9   Representation

A `representation` element describes a representation of a resource's state. A `representation` element can either be a representation definition or a reference to a representation defined elsewhere.

2.9.1   Representation Reference

A `representation` reference element can be a child of a `request` or `response` element. It has a `href` attribute of type `xsd:anyURI`. The value of the `href` attribute is a URI reference to a `representation` definition element. A `representation` reference element MUST NOT have any other WADL-defined attributes or contain any WADL-defined child elements.

This form of `representation` element may be used to reduce duplication when the same representation is used in multiple locations.

2.9.2   Representation Definition

A `representation` definition element can be a child of a `request`, `response` or `application` element. It has the following attributes:

id An identifier for the representation, required for globally defined representations, not allowed on locally embedded representations. Representations are identified by an XML ID and are referred to using a URI reference.

mediaType Indicates the media type of the representation.

element For XML-based representations, specifies the qualified name of the root element as described within the `grammars` section – see section 2.3.

profile Similar to the HTML `profile` attribute, gives the location of one or more meta data profiles, separated by white space. The meta-data profiles define the meaning of the `rel` and `rev` attributes of descendent `link` elements (see section 2.11.2).

In addition to the attributes listed above, a `representation` definition element can have zero or more child `doc` elements (see section 2.2) and `param` elements (see section 2.11).

2.9.3 Representation Parameters

A child `param` element (see section 2.11) is used to parameterize its parent `representation` element. Representation parameters can have one of two different functions depending on the media type of the representation:

1. Define the content of the representation. For `representation` elements with a `mediaType` attribute whose value is either 'application/x-www-form-urlencoded' or 'multipart/form-data' the representation parameters define the content of the representation which is formatted according to the media type. The same may apply to other media types.

2. Provide a hint to processors about items of interest within a representation. For XML based representations, representation parameters can be used to identify items of interest with the XML. The `path` attribute of a representation parameter indicates the path to the value of the parameter within the representation. For XML-based representations this is an XPath expression.

2.10 Fault

A `fault` element is similar to a `representation` element (see section 2.9) in structure but differs in that it denotes an error condition. A `fault` element has the same attributes as a `representation` element but may also have an additional `status` attribute that provides a list of HTTP status codes associated with a particular error condition. Note that multiple `fault` elements may share one or more HTTP status codes: such elements may describe more granular fault conditions or may provide equivalent information in different formats.

2.10.1 Fault Parameters

Fault parameters are `param` elements (see section 2.11) that are direct children of a `fault` element. Fault parameters perform the same function for `fault` elements that representation parameters (see section 2.9.3) perform for `representation` elements.

2.11 Parameter

A `param` element describes a parameterized component of its parent element and may be a child of a `resource` (see section 2.5), `request` (see section 2.7), or `representation` (see section 2.9) element. A `param` element has zero or more `doc` child elements (see section 2.2), zero or more `option` child elements (see section 2.11.1), an optional `link` child element (see section 2.11.2) and has the following attributes:

name The name of the parameter as an xsd:NMTOKEN. Required.

type Optionally indicates the type of the parameter as an XML qualified name, defaults to `xsd:string`.

default Optionally provides a value that is considered identical to an unspecified parameter value.

style Optionally indicates the parameter style as one of the following values:

> **plain** The parameter is represented as a string encoding of the parameter value. This is the default when the parent element is `resource` (see section 2.5) or `representation` (see section 2.9) whose media type is neither 'application/x-www-form-urlencoded' nor 'multipart/form-data'.
>
> **form** The parameter is represented as a name value pair according to the rules specified in section 17.13 of HTML 4.01[3]. When the parent element is `request` (see section 2.7) the parameter becomes a URI query parameter. When the parent element is `representation` (see section 2.9) whose media type is either 'application/x-www-form-urlencoded' or 'multipart/form-data' the parameter is embedded within the representation.
>
> **matrix** The parameter is represented as a matrix URI component.

path When the parent element is a `representation` element, this attribute optionally provides a path to the value of the parameter within the representation.

required Optionally indicates whether the parameter is required to be present or not, defaults to false (parameter not required).

repeating Optionally indicates whether the parameter is single valued or may have multiple values, defaults to false (parameter is single valued).

fixed Optionally provides a fixed value for the parameter.

Note that some combinations of the above attributes might not make sense in all cases. E.g. matrix URI parameters are normally optional so a `param` element with a `style` value of 'matrix' and a `required` value of 'true' might be unwise.

2.11.1 Option

An `option` element defines one of a set of possible values for the parameter represented by its parent `param` element. An `option` element has a required `value` attribute that defines the value and zero or more `doc` elements that document the meaning of the value.

2.11.2 Link

A `link` element is used to identify links to resources within representations. A `link` element is a child of a `param` element whose `path` attribute identifies the portion of its parent representation that contains a link URI.

A `link` element contains zero or more `doc` elements (see section 2.2 and has the following attributes:

href An optional URI reference to a `resource` element that defines the resource that the link identifies.

rel An optional token that identifies the relationship of the resource identified by the link to the resource whose representation the link is embedded in. The value is scoped by the value of the ancestor `representation` (or `fault`) element's `profile` attribute.

rev An optional token that identifies the relationship of the resource whose representation the link is embedded in to the resource identified by the link. This is the reverse relationship to that identified by the `rel` attribute. The value is scoped by the value of the ancestor `representation` (or `fault`) element's `profile` attribute.

The following example shows an XML-based resource representation and two possible alternative WADL representation elements:

```
1   <!-- XML-based representation of a widget -->
2   <w:widget xmlns:w="http://example.com/widgets">
3     <w:loc>http://example.com/widgets/110113</w:loc>
4     <w:name>A Widget</w:name>
5     <w:description>A very useful gizmo.</w:description>
6     <w:price currency="USD">19.99</w:price>
7     <w:list>http://example.com/widgets</w:list>
8   </w:widget>
9
10  <!-- WADL fragment describing the widget representation
11       without parameters-->
12  <representation mediaType="application/xml"
13    element="w:widget"/>
14
15  <!-- WADL fragment describing the widget representation
16       with parameters -->
17  <representation mediaType="application/xml"
```

```
18    element="w:widget">
19    <param name="location"
20      type="xsd:anyURI" path="/w:widget/w:loc">
21      <link href="#widget" rel="self"/>
22    </param>
23    <param name="index"
24      type="xsd:anyURI" path="/w:widget/w:list">
25      <link href="#widgets" rel="index" rev="child"/>
26    </param>
27  </representation>
```

The second version identifies two links within a widget representation:

location The URI of the widget resource being represented. A widget resource is described by the WADL `resource` element whose `id` is 'widget'.

index The URI of a resource that acts as an index of widgets. The index resource is described by the WADL `resource` element whose `id` is 'widgets'.

2.12 Extensibility

Most WADL-defined elements are extensible using either elements or attributes from foreign namespaces. A WADL processor MAY ignore extensions that it does not understand and extension authors should design extensions with this in mind.

3 Use of RelaxNG with WADL

One or more legal RelaxNG schemas may be embedded within a WADL `grammars` element or may be included by reference using an `include` element. Multiple RelaxNG schemas may be combined within a single schema using the facilities provided by RelaxNG (e.g., `rng:include`). The default namespace for an included RelaxNG grammar is the default namespace of the WADL `grammars` element.

The `element` attribute of `representation` and `fault` elements refers to a corresponding RelaxNG element pattern using the XML qualified name of the element.

4 Use of W3C XML Schema with WADL

One or more legal W3C XML Schemas may be embedded within a WADL `grammars` element or may be included by reference using a `include` element. Multiple W3C XML Schemas may be combined within a single schema using the facilities provided by W3C XML Schema (e.g., `xsd:include`).

The `element` attribute of `representation` and `fault` elements refers to a corresponding W3C XML Schema global element declaration using the XML qualified name of the element.

5   WADL Media Type

WADL documents should be served using the `application/vnd.sun.wadl+xml` media type and use a `.wadl` filename extension. See the WADL media type registration[4] for full details.

A   Additional Examples

A.1   Amazon Item Search

The following shows a WADL description of the Amazon item search service[5]:

```
1   <application xmlns="http://research.sun.com/wadl/2006/07"
2     xmlns:aws="http://webservices.amazon.com/AWSECommerceService/2005-07-26"
3     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4
5     <grammars>
6       <include href="AWSECommerceService.xsd"/>
7     </grammars>
8
9     <resources base="http://webservices.amazon.com/onca/">
10      <resource path="xml">
11        <method href="#ItemSearch"/>
12      </resource>
13    </resources>
14
15    <method name="GET" id="ItemSearch">
16      <request>
17        <param name="Service" fixed="AWSECommerceService"/>
18        <param name="Version" fixed="2005-07-26"/>
19        <param name="Operation" fixed="ItemSearch"/>
20        <param name="SubscriptionId" type="xsd:string"
21          required="true"/>
22        <param name="SearchIndex" type="aws:SearchIndexType"
23          required="true"/>
24        <param name="Keywords" type="aws:KeywordList"
25          required="true"/>
26        <param name="ResponseGroup" type="aws:ResponseGroupType"/>
27      </request>
28      <response>
29        <representation mediaType="text/xml"
30          element="aws:ItemSearchResponse"/>
31      </response>
32    </method>
33  </application>
```

Note the following:

- The method is attached to the resource as a reference to a globally defined method rather than being embedded directly. 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 parameters are marked as fixed value. The Amazon API uses 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 would look like one single method with many parameters.

A.2   Atom Publishing Protocol

The Atom publishing protocol[6] defines a set of methods to introspect, view and update entries in an Atom feed. The publishing protocol is bootstrapped by performing a HTTP GET on a known URI for a particular set of feeds. The response consists of an XML document, of media type application/atomserv+xml, that describes the available feeds. An example of such is shown below:

```
1    <service xmlns="http://purl.org/atom/app#">
2      <workspace title="Main Site" >
3        <collection
4          title="My Blog Entries"
5          href="http://example.org/reilly/main" >
6          <member-type>entry</member-type>
7          <list-template>http://example.org/{index}</list-template>
8        </collection>
9        <collection
10         title="Pictures"
11         href="http://example.org/reilly/pic" >
12         <member-type>media</member-type>
13         <list-template>http://example.org/p/{index}</list-template>
14       </collection>
15     </workspace>
16   </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, associated representations and resource templates. 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.

```
1    <application xmlns="http://research.sun.com/wadl/2006/07"
2      xmlns:app="http://purl.org/atom/app#"
3      xmlns:atom="http://www.w3.org/2005/Atom">
4
5      <grammars>
6        <include href="http://purl.org/atom/app.xsd"/>
7      </grammars>
8
9      <resource id="feed_resource" path="{feedId}" app:member-type="entry">
```

```
10      <method href="#getFeed"/>
11      <method href="#addEntryCollectionMember"/>
12    </resource>
13
14    <resource id="entry_resource" path="{entries}{entryId}">
15      <method href="#readEntryCollectionMember"/>
16      <method href="#deleteCollectionMember"/>
17      <method href="#updateEntryCollectionMember"/>
18    </resource>
19
20    <representation id="entry" mediaType="application/atom+xml"
21      element="atom:entry"/>
22
23    <representation id="feed" mediaType="application/atom+xml"
24      element="atom:feed">
25      <param name="first_link"
26        path="/atom:feed/atom:link[@rel='first']">
27        <link href="#feed_resource" rel="first"/>
28      </representation>
29      <param name="next_link"
30        path="/atom:feed/atom:link[@rel='next']">
31        <link href="#feed_resource" rel="next" rev="previous"/>
32      </param>
33      <param name="prev_link"
34        path="/atom:feed/atom:link[@rel='previous']">
35        <link href="#feed_resource" rel="previous" rev="next"/>
36      </param>
37      <param name="last_link"
38        path="/atom:feed/atom:link[@rel='last']">
39        <link href="#feed_resource" rel="last"/>
40      </param>
41    </representation>
42
43    <method name="GET" id="getFeed">
44      <response>
45        <representation href="#feed"/>
46      </response>
47    </method>
48
49    <method name="POST" id="addEntryCollectionMember">
50      <request>
51        <representation href="#entry"/>
52      </request>
53    </method>
54
55    <method name="POST" id="addGenericCollectionMember">
56      <request>
57        <representation href="#entry"/>
58        <representation />
59      </request>
60    </method>
61
62    <method name="DELETE" id="deleteCollectionMember"/>
63
```

```
64    <method name="GET" id="readEntryCollectionMember">
65      <response>
66        <representation href="#entry"/>
67      </response>
68    </method>
69
70    <method name="GET" id="readGenericCollectionMember">
71      <response>
72        <representation href="#entry"/>
73        <representation />
74      </response>
75    </method>
76
77    <method name="PUT" id="updateEntryCollectionMember">
78      <request>
79        <representation href="#entry"/>
80      </request>
81      <response>
82        <representation href="#entry"/>
83      </response>
84    </method>
85
86    <method name="PUT" id="updateGenericCollectionMember">
87      <request>
88        <representation href="#entry"/>
89        <representation />
90      </request>
91      <response>
92        <representation href="#entry"/>
93        <representation />
94      </response>
95    </method>
96
97  </application>
```

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

```
1   <application xmlns="http://research.sun.com/wadl/2006/07"
2     xml:base="http://purl.org/atom/app.wadl"
3     xmlns:app="http://purl.org/atom/app#">
4
5     <resources base="http://example.org/">
6       <resource href="#feed_resource" app:member-type="entry">
7         <param name="feedId" fixed="reilly/main"/>
8       </resource>
9       <resource href="#entry_resource">
10        <param name="entries" fixed="reilly/main"/>
11      </resource>
12    </resources>
13  </application>
```

Note the use of the xml:base attribute to allow use of relative URIs in method references. The above WADL document describes the following 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/{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 an Atom-specific extension element (`app:member-type`) to provide the same metadata as the Atom service document.

B RelaxNG Schema for WADL

```
1   namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
2   namespace local = ""
3   namespace wadl = "http://research.sun.com/wadl/2006/07"
4
5   start =
6     element wadl:application {
7       doc*,
8       grammars?,
9       resources?,
10      (resource | method | representation | fault)*,
11      foreign-attribute,
12      foreign-element
13    }
14  languageTag = xsd:string {
15     pattern = "[A-Za-z]{1,8}(-[A-Za-z0-9]{1,8})*"
16  }
17  doc =
18    element wadl:doc {
19      attribute xml:lang { languageTag }?,
20      attribute title { text }?,
21      (text | foreign-element)*,
22      foreign-attribute
23    }
24  grammars =
25    element wadl:grammars {
26      doc*,
27      incl*,
28      foreign-element
29    }
30  incl =
31    element wadl:include {
32      doc*,
33      attribute href { xsd:anyURI },
34      foreign-attribute
35    }
36  resources =
37    element wadl:resources {
38      doc*,
39      resource+,
40      attribute base { xsd:anyURI },
41      foreign-attribute,
42      foreign-element
43    }
44  resource =
45    element wadl:resource {
46      (attribute href { xsd:anyURI }
47       | (doc*,
48          param*,
49          (method | resource)*,
50          attribute path { text }?,
51          attribute id { xsd:token }?
```

```
52            )),
53        foreign-element,
54        foreign-attribute
55      }
56    method =
57      element wadl:method {
58        (attribute href { xsd:anyURI }
59         | (doc*,
60            request?,
61            response?,
62            attribute id { xsd:token }?,
63            attribute name {
64               "DELETE" | "GET" | "HEAD" | "POST" | "PUT" | xsd:token
65            })),
66        foreign-element,
67        foreign-attribute
68      }
69    request =
70      element wadl:request {
71        doc*,
72        representation*,
73        param*,
74        foreign-attribute,
75        foreign-element
76      }
77    response =
78      element wadl:response {
79        doc*,
80        (representation | fault)*,
81        foreign-attribute,
82        foreign-element
83      }
84    representation =
85      element wadl:representation {
86        (attribute href { xsd:anyURI }
87         | (doc*,
88            param*,
89            attribute id { xsd:token }?,
90            attribute element { xsd:QName }?,
91            attribute mediaType { text }?,
92            attribute profile { list { xsd:anyURI} }?)),
93        foreign-attribute,
94        foreign-element
95      }
96    fault =
97      element wadl:fault {
98        (attribute href { xsd:anyURI }
99         | (doc*,
100           param*,
101           attribute id { xsd:token }?,
102           attribute element { xsd:QName }?,
103           attribute mediaType { text }?,
104           attribute status {
105              list { xsd:int+ }
```

```
106          }?)),
107      foreign-attribute,
108      foreign-element
109    }
110  param =
111    element wadl:param {
112      doc*,
113      option*,
114      link?,
115      attribute name {xsd:token },
116      attribute type { text }?,
117      attribute default { text }?,
118      attribute path { text }?,
119      attribute style { "plain" | "form" | "matrix" }?,
120      attribute required { xsd:boolean }?,
121      attribute repeating { xsd:boolean }?,
122      attribute fixed { text }?,
123      foreign-element,
124      foreign-attribute
125    }
126  option =
127    element wadl:option {
128      doc*,
129      attribute value { xsd:string },
130      foreign-element,
131      foreign-attribute
132    }
133  link =
134    element wadl:link {
135      doc*,
136      attribute href { xsd:anyURI }?,
137      attribute rel { xsd:token }?,
138      attribute rev { xsd:token }?,
139      foreign-element,
140      foreign-attribute
141    }
142  foreign-attribute = attribute * - (wadl:* | local:* | xml:*) { text }*
143  foreign-element =
144    element * - (wadl:* | local:*) {
145      (attribute * { text }
146       | text
147       | any-element)*
148    }*
149  any-element =
150    element * {
151      (attribute * { text }
152       | text
153       | any-element)*
154    }*
```

C   XML Schema for WADL

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3      targetNamespace="http://research.sun.com/wadl/2006/07"
4      xmlns:tns="http://research.sun.com/wadl/2006/07"
5      xmlns:xml="http://www.w3.org/XML/1998/namespace"
6      elementFormDefault="qualified">
7
8      <xs:import namespace="http://www.w3.org/XML/1998/namespace"
9        schemaLocation="http://www.w3.org/2001/xml.xsd"/>
10
11     <xs:element name="application">
12       <xs:complexType>
13         <xs:sequence>
14           <xs:element ref="tns:doc" minOccurs="0"
15             maxOccurs="unbounded"/>
16           <xs:element ref="tns:grammars" minOccurs="0"/>
17           <xs:element ref="tns:resources" minOccurs="0"/>
18           <xs:choice minOccurs="0" maxOccurs="unbounded">
19             <xs:element ref="tns:resource"/>
20             <xs:element ref="tns:method"/>
21             <xs:element ref="tns:representation"/>
22             <xs:element ref="tns:fault"/>
23           </xs:choice>
24           <xs:any namespace="##other" processContents="lax"
25             minOccurs="0" maxOccurs="unbounded"/>
26         </xs:sequence>
27       </xs:complexType>
28     </xs:element>
29
30     <xs:element name="doc">
31       <xs:complexType mixed="true">
32         <xs:sequence>
33           <xs:any namespace="##other" processContents="lax"
34             minOccurs="0" maxOccurs="unbounded"/>
35         </xs:sequence>
36         <xs:attribute name="title" type="xs:string"/>
37         <xs:attribute ref="xml:lang"/>
38         <xs:anyAttribute namespace="##other" processContents="lax"/>
39       </xs:complexType>
40     </xs:element>
41
42     <xs:element name="grammars">
43       <xs:complexType>
44         <xs:sequence>
45           <xs:element ref="tns:doc" minOccurs="0"
46             maxOccurs="unbounded"/>
47           <xs:element minOccurs="0" maxOccurs="unbounded"
48             ref="tns:include"/>
49           <xs:any namespace="##other" processContents="lax"
50             minOccurs="0" maxOccurs="unbounded"/>
51         </xs:sequence>
```

```
52        </xs:complexType>
53      </xs:element>
54
55      <xs:element name="resources">
56        <xs:complexType>
57          <xs:sequence>
58            <xs:element ref="tns:doc" minOccurs="0"
59              maxOccurs="unbounded"/>
60            <xs:element ref="tns:resource" maxOccurs="unbounded"/>
61            <xs:any namespace="##other" processContents="lax"
62              minOccurs="0" maxOccurs="unbounded"/>
63          </xs:sequence>
64          <xs:attribute name="base" type="xs:anyURI"/>
65          <xs:anyAttribute namespace="##other" processContents="lax"/>
66        </xs:complexType>
67      </xs:element>
68
69      <xs:element name="resource">
70        <xs:complexType>
71          <xs:sequence>
72            <xs:element ref="tns:doc" minOccurs="0"
73              maxOccurs="unbounded"/>
74            <xs:element ref="tns:param" minOccurs="0"
75              maxOccurs="unbounded"/>
76            <xs:choice minOccurs="0" maxOccurs="unbounded">
77              <xs:element ref="tns:method"/>
78              <xs:element ref="tns:resource"/>
79            </xs:choice>
80            <xs:any minOccurs="0" maxOccurs="unbounded"
81              namespace="##other" processContents="lax"/>
82          </xs:sequence>
83          <xs:attribute name="id" type="xs:ID"/>
84          <xs:attribute name="href" type="xs:anyURI"/>
85          <xs:attribute name="path" type="xs:string"/>
86          <xs:anyAttribute namespace="##other" processContents="lax"/>
87        </xs:complexType>
88      </xs:element>
89
90      <xs:element name="method">
91        <xs:complexType>
92          <xs:sequence>
93            <xs:element ref="tns:doc" minOccurs="0"
94              maxOccurs="unbounded"/>
95            <xs:element ref="tns:request" minOccurs="0"/>
96            <xs:element ref="tns:response" minOccurs="0"/>
97            <xs:any namespace="##other" processContents="lax"
98              minOccurs="0" maxOccurs="unbounded"/>
99          </xs:sequence>
100         <xs:attribute name="id" type="xs:ID"/>
101         <xs:attribute name="name" type="tns:Method"/>
102         <xs:attribute name="href" type="xs:anyURI"/>
103         <xs:anyAttribute namespace="##other" processContents="lax"/>
104       </xs:complexType>
105     </xs:element>
```

```
106
107     <xs:simpleType name="Method">
108       <xs:union memberTypes="tns:HTTPMethods xs:NMTOKEN"/>
109     </xs:simpleType>
110
111     <xs:simpleType name="HTTPMethods">
112       <xs:restriction base="xs:NMTOKEN">
113         <xs:enumeration value="GET"/>
114         <xs:enumeration value="POST"/>
115         <xs:enumeration value="PUT"/>
116         <xs:enumeration value="HEAD"/>
117         <xs:enumeration value="DELETE"/>
118       </xs:restriction>
119     </xs:simpleType>
120
121     <xs:element name="include">
122       <xs:complexType>
123         <xs:sequence>
124           <xs:element ref="tns:doc" minOccurs="0"
125             maxOccurs="unbounded"/>
126         </xs:sequence>
127         <xs:attribute name="href" type="xs:anyURI"/>
128         <xs:anyAttribute namespace="##other" processContents="lax"/>
129       </xs:complexType>
130     </xs:element>
131
132     <xs:element name="request">
133       <xs:complexType>
134         <xs:sequence>
135           <xs:element ref="tns:doc" minOccurs="0"
136             maxOccurs="unbounded"/>
137           <xs:element ref="tns:representation" minOccurs="0"
138             maxOccurs="unbounded"/>
139           <xs:element ref="tns:param" minOccurs="0"
140             maxOccurs="unbounded"/>
141           <xs:any namespace="##other" processContents="lax"
142             minOccurs="0" maxOccurs="unbounded"/>
143         </xs:sequence>
144         <xs:anyAttribute namespace="##other" processContents="lax"/>
145       </xs:complexType>
146     </xs:element>
147
148     <xs:element name="response">
149       <xs:complexType>
150         <xs:sequence>
151           <xs:element ref="tns:doc" minOccurs="0"
152             maxOccurs="unbounded"/>
153           <xs:choice minOccurs="0" maxOccurs="unbounded">
154             <xs:element ref="tns:representation"/>
155             <xs:element ref="tns:fault"/>
156           </xs:choice>
157           <xs:any namespace="##other" processContents="lax"
158             minOccurs="0" maxOccurs="unbounded"/>
159         </xs:sequence>
```

```
160         <xs:anyAttribute namespace="##other" processContents="lax"/>
161       </xs:complexType>
162     </xs:element>
163
164     <xs:simpleType name="uriList">
165       <xs:list itemType="xs:anyURI"/>
166     </xs:simpleType>
167
168     <xs:element name="representation">
169       <xs:complexType>
170         <xs:sequence>
171           <xs:element ref="tns:doc" minOccurs="0"
172             maxOccurs="unbounded"/>
173           <xs:element ref="tns:param" minOccurs="0"
174             maxOccurs="unbounded"/>
175           <xs:any namespace="##other" processContents="lax"
176             minOccurs="0" maxOccurs="unbounded"/>
177         </xs:sequence>
178         <xs:attribute name="id" type="xs:ID"/>
179         <xs:attribute name="element" type="xs:QName"/>
180         <xs:attribute name="mediaType" type="xs:string"/>
181         <xs:attribute name="href" type="xs:anyURI"/>
182         <xs:attribute name="profile" type="tns:uriList"/>
183         <xs:anyAttribute namespace="##other" processContents="lax"/>
184       </xs:complexType>
185     </xs:element>
186
187     <xs:simpleType name="faultCodeList">
188       <xs:list itemType="xs:unsignedInt"/>
189     </xs:simpleType>
190
191     <xs:element name="fault">
192       <xs:complexType>
193         <xs:sequence>
194           <xs:element ref="tns:doc" minOccurs="0"
195             maxOccurs="unbounded"/>
196           <xs:element ref="tns:param" minOccurs="0"
197             maxOccurs="unbounded"/>
198           <xs:any namespace="##other" processContents="lax"
199             minOccurs="0" maxOccurs="unbounded"/>
200         </xs:sequence>
201         <xs:attribute name="id" type="xs:ID" use="required"/>
202         <xs:attribute name="element" type="xs:QName"/>
203         <xs:attribute name="status" type="tns:faultCodeList"/>
204         <xs:attribute name="mediaType" type="xs:string"/>
205         <xs:attribute name="href" type="xs:anyURI"/>
206         <xs:anyAttribute namespace="##other" processContents="lax"/>
207       </xs:complexType>
208     </xs:element>
209
210     <xs:simpleType name="ParamStyle">
211       <xs:restriction base="xs:string">
212         <xs:enumeration value="plain"/>
213         <xs:enumeration value="form"/>
```

```
214        <xs:enumeration value="matrix"/>
215      </xs:restriction>
216    </xs:simpleType>
217
218    <xs:element name="param">
219      <xs:complexType>
220        <xs:sequence>
221          <xs:element ref="tns:doc" minOccurs="0"
222            maxOccurs="unbounded"/>
223          <xs:element ref="tns:option" minOccurs="0"
224            maxOccurs="unbounded"/>
225          <xs:element ref="tns:link" minOccurs="0"/>
226          <xs:any namespace="##other" processContents="lax"
227            minOccurs="0" maxOccurs="unbounded"/>
228        </xs:sequence>
229        <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
230        <xs:attribute name="type" type="xs:QName" default="xs:string"/>
231        <xs:attribute name="default" type="xs:string"/>
232        <xs:attribute name="style" type="tns:ParamStyle"/>
233        <xs:attribute name="required" type="xs:boolean"
234          default="false"/>
235        <xs:attribute name="repeating" type="xs:boolean"
236          default="false"/>
237        <xs:attribute name="fixed" type="xs:string"/>
238        <xs:attribute name="path" type="xs:string"/>
239        <xs:anyAttribute namespace="##other" processContents="lax"/>
240      </xs:complexType>
241    </xs:element>
242
243    <xs:element name="option">
244      <xs:complexType>
245        <xs:sequence>
246          <xs:element ref="tns:doc" minOccurs="0"
247            maxOccurs="unbounded"/>
248          <xs:any namespace="##other" processContents="lax"
249            minOccurs="0" maxOccurs="unbounded"></xs:any>
250        </xs:sequence>
251        <xs:attribute name="value" type="xs:string" use="required"/>
252        <xs:anyAttribute namespace="##other" processContents="lax"/>
253      </xs:complexType>
254    </xs:element>
255
256    <xs:element name="link">
257      <xs:complexType>
258        <xs:sequence>
259          <xs:element ref="tns:doc" minOccurs="0"
260            maxOccurs="unbounded"/>
261          <xs:any namespace="##other" processContents="lax"
262            minOccurs="0" maxOccurs="unbounded"></xs:any>
263        </xs:sequence>
264        <xs:attribute name="href" type="xs:anyURI"/>
265        <xs:attribute name="rel" type="xs:token"/>
266        <xs:attribute name="rev" type="xs:token"/>
267        <xs:anyAttribute namespace="##other" processContents="lax"/>
```

```
268         </xs:complexType>
269     </xs:element>
270
271   </xs:schema>
```

References

[1] Yahoo! Web APIs. Technical report, Yahoo!, 2005. See http://developer.yahoo.net/.

[2] Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron, Sebastian Schnitzenbaumer, and Ted Wugofski. Modularization of XHTML. Recommendation, W3C, April 2001. See http://www.w3.org/TR/xhtml-modularization.

[3] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. Recommendation, W3C, December 1999. See http://www.w3.org/TR/html4/.

[4] M. Hadley. The application/vd.sun.wadl+xml Media Type. Media Type, IANA, March 2006. See http://www.iana.org/assignments/media-types/application/vnd.sun.wadl+xml.

[5] Amazon.com. Amazon Web Services. Technical report, Amazon.com, 2005. See http://www.amazon.com/.

[6] J.C. Gregorio and B. de hOra. The Atom Publishing Protocol. Internet Draft, IETF, January 2006. See http://bitworking.org/projects/atom/draft-ietf-atompub-protocol-07.html.

Acknowledgments

Copyright Notice