Web Application Description Language (WADL)

Marc J. Hadley, Sun Microsystems Inc.

November 9, 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 description of 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 description of HTTP-based Web applications.

1.1 Web Applications

For the purposes of this specification, 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 a self-describing data format such as XML or JSON. XML is particularly suitable since it allows the definition of a complete custom schema for the application domain or the embedding of 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 data schemas in use.

1.2 Use Cases

The current state-of-the-art in Web application description is textual documentation plus one or more data format definitions, e.g. XML schemata. Whilst entirely adequate for human consumption, this level of description precludes the following use cases which require a more machine-friendly 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/2006/10 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/2006/10">
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"
21              style="query" required="true"/>
22            <param name="query" type="xsd:string"
23              style="query" required="true"/>
24            <param name="type" style="query" default="all">
25              <option value="all"/>
26              <option value="any"/>
27              <option value="phrase"/>
28            </param>
29            <param name="results" style="query" type="xsd:int" default="10"/>
30            <param name="start" style="query" type="xsd:int" default="1"/>
31            <param name="sort" style="query" default="rank">
32              <option value="rank"/>
33              <option value="date"/>
34            </param>
35            <param name="language" style="query" type="xsd:string"/>
36          </request>
37          <response>
38            <representation mediaType="application/xml"
39              element="yn:ResultSet"/>
40            <fault status="400" mediaType="application/xml"
41              element="ya:Error"/>
42          </response>
43        </method>
44      </resource>
45    </resources>
46
47  </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 16–45 describe the Yahoo News Search Web resource and the HTTP methods it supports. Lines 18–43 describe the ‘search’ GET method: lines 19–36 describe the input; lines 37–42 describe the possible outputs.

2 Description Components

All WADL elements have the following XML namespace name:

- `http://research.sun.com/wadl/2006/10`

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_type` elements – see section 2.6.
   - `method` elements – see section 2.7.
   - `representation` elements – see section 2.10.
   - `fault` elements – see section 2.11.

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) describe the resources provided by the application.

2.5 Resource

A `resource` element describes a set of resources, each identified by a URI that follows a common pattern. A `resource` element has the following attributes:

`id` An optional attribute of type `xsd:ID` that identifies the `resource` element.

`path` An optional attribute of type `xsd:string`. If present, it provides a relative URI template[3] for the identifier of the resource. The resource's base URI is given by the `resource` element's parent `resource` or `resources` element.

`type` An optional attribute whose type is a space-separated list of of `xsd:anyURI`. Each value in the list identifies a `resource_type` element (see section 2.6) that defines a set of methods supported by the resource.

`queryType` Defines the media type for the query component of the resource URI. Defaults to 'application/x-www-form-urlencoded' if not specified which results in query strings being formatted as specified in section 17.13 of HTML 4.01[4].

A `resource` element contains the following child elements:

- Zero or more `doc` elements – see section 2.2.
- Zero or more `param` elements (see section 2.12) with one of the following values for its `style` attribute:

  `template` Provides additional information about an embedded template parameter, see above. Child `param` elements whose `name` attribute value does not match the *name* of an embedded template parameter are ignored.

  `matrix` Specifies a matrix URI parameter

  `query` Specifies a global URI query parameter for all child `method` elements of the resource. Does not apply to methods inherited from a `resource_type` specified using the `type` attribute.

  `header` Specifies a global HTTP header for use in the request part of all child `method` elements of the resource. Does not apply to methods inherited from a `resource_type` specified using the `type` attribute.

- Zero or more `method` (see section 2.7) elements, each of which describes the input to and output from an HTTP protocol method that can be applied to the resource. Such locally-defined methods are added to any methods included in `resource_type` elements referred to using the `type` attribute.
- Zero or more `resource` elements that describe sub-resources. Such sub-resources inherit matrix and template parameters from the parent resource since their URI is relative to that of the parent resource but they do not inherit query or header parameters specified globally for the parent resource.

The value of the `path` attribute may be static or may contain embedded template parameters, see [3]. At runtime, the values of template parameters are substituted into the resource identifier when the resource is used, see section 2.5.1 for a detailed example.

Additional information about embedded template parameters can be conveyed using a child `param` element with a `style` attribute value 'template' whose `name` attribute value matches the name of the parameter embedded in the template. E.g., in the following the type of the `widgetId` template parameter is specified by the child `param` element:

```
1  <resource path="widgets/{widgetId}">
2    <param name="widgetId" style="template" type="xsd:int"/>
3    ...
4  </resource>
```

2.5.1 Generating Resource Identifiers

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

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

2. If *identifier* doesn't end with a '/' then append a '/' character to *identifier*
3. Substitute the values of any URI template parameters into the value of the `path` attribute according to [3]
4. Append the value obtained in the previous step to *identifier*
5. For each child `param` element (see section 2.12), 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 Resource Type

A `resource_type` element describes a set of methods that, together, define the behavior of a type of resource. A `resource_type` may be used to define resource behavior that is expected to be supported by multiple resources.

A `resource_type` element has the following attributes:

`id` A required attribute of type `xsd:ID` that identifies the `resource_type` element.

A `resource_type` element contains the following child elements:

- Zero or more `doc` elements – see section 2.2.
- Zero or more `param` elements (see section 2.12) with one of the following values for its `style` attribute:

  `query` Specifies a URI query parameter for all child `method` elements of the resource type.

  `header` Specifies a HTTP header for use in the request part of all child `method` elements of the resource type.
- Zero or more `method` (see section 2.7) elements, each of which describes an HTTP protocol method that can be applied to a resource of this type.

2.7 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.

2.7.1 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.

2.7.2 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 2.8.

`response` Describes the output of the method as a collection of alternate resource representations – see section 2.9.

2.8 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 2.10. 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 section 2.12) with one of the following values for their `style` attribute:

   `query` Specifies a URI query parameter for all methods that apply to this resource, see section 2.8.1

   `header` Specifies a HTTP header for use in the request

2.8.1 Query Parameters

Child `param` elements (see section 2.12) of a `resource` or `request` with a `style` value of 'query' represent URI query parameters as described in section 17.13 of HTML 4.01[4]. 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 two optional query parameters:

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

If the value of the `widgetId` parameter is '123456' the value of the `customerId` parameter is 'cust1234' and the value of the `verbose` parameter is 'true' then the URI on which the HTTP GET will be performed is:

```
http://example.com/widgets/123456?customerId=cust1234&verbose=true
```

2.9 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 2.10), each of which 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.
- Zero or more `fault` elements (see section 2.11), each of which 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.

- Zero or more `param` elements (see section 2.12) with a value of 'header' for their `style` attribute, each of which specifies the details of a HTTP header for the response

2.10 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.

2.10.1 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.

2.10.2 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 2.12.2).

`status` Optionally present on response representations, provides a list of HTTP status codes associated with a particular representation. Note that multiple sibling `representation` elements may share one or more HTTP status codes: such elements may provide equivalent information in different formats.

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 2.10.3).

2.10.3 Representation Parameters

A child `param` element (see section 2.12) 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.

2.11 Fault

A `fault` element is similar to a `representation` element (see section 2.10) in structure but differs in that it denotes an error condition. A `fault` element has the same attributes as a `representation` element. As is the case with `representation` elements, multiple sibling `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.

2.11.1 Fault Parameters

Fault parameters are `param` elements (see section 2.12) that are direct children of a `fault` element. Fault parameters perform the same function for `fault` elements that representation parameters (see section 2.10.3) perform for `representation` elements.

2.12 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 2.8), `response` (see section 2.9), `representation` (see section 2.10), or a `fault` (see section 2.11) element. A `param` element has zero or more `doc` child elements (see section 2.2), zero or more `option` child elements (see section 2.12.1), an optional `link` child element (see section 2.12.2) and has the following attributes:

`id` An optional identifier that may be used to refer to a parameter definition using a URI reference.

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

`style` Indicates the parameter style, table 1 on page 14 lists the allowed values and shows the context(s) in which each value may be used.

`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.

`path` When the parent element is a `representation` element, this attribute optionally provides a path to the value of the parameter within the representation. For XML representations, use of XPath 1.0[5] is recommended.

`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.

2.12.1 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.

2.12.2 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:

`resource_type` An optional URI reference to a `resource_type` element that defines the capabilities of 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.

Table 1: Values of `style` attribute and context for use

Value	`param` Parent Element(s)	Usage
`matrix`	`resource`	Specifies a matrix URI component.
`header`	`resource`, `resource_type`, `request` or `response`	Specifies a HTTP header that pertains to the HTTP request (`resource` or `request`) or HTTP response (`response`)
`query`	`resource`, `resource_type` or `request`	Specifies a URI query parameter represented according to the rules for the query component media type specified by the `queryType` attribute.
`query`	`representation` or `fault`	Specifies a component of the representation as a name value pair formatted according to the rules of the media type. Typically used with media type 'application/x-www-form-urlencoded' or 'multipart/form-data'.
`template`	`resource`	The parameter is represented as a string encoding of the parameter value and is substituted into the value of the `path` attribute of the `resource` element as described in section 2.5.1.
`plain`	`representation` or `fault`	Specifies a component of the representation formatted as a string encoding of the parameter value according to the rules of the media type.

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:

```
<!-- XML-based representation of a widget -->
<w:widget xmlns:w="http://example.com/widgets">
  <w:loc>http://example.com/widgets/110113</w:loc>
  <w:name>A Widget</w:name>
  <w:description>A very useful gizmo.</w:description>
  <w:price currency="USD">19.99</w:price>
  <w:list>http://example.com/widgets</w:list>
</w:widget>

<!-- WADL fragment describing the widget representation
     without parameters-->
<representation mediaType="application/xml"
  element="w:widget"/>

<!-- WADL fragment describing the widget representation
     with parameters -->
<representation mediaType="application/xml"
  element="w:widget">
  <param name="location" style="plain"
    type="xsd:anyURI" path="/w:widget/w:loc">
    <link resource_type="#widget" rel="self"/>
  </param>
  <param name="index" style="plain"
    type="xsd:anyURI" path="/w:widget/w:list">
    <link resource_type="#widgets" rel="index" rev="child"/>
  </param>
</representation>
```

The second version identifies two links within a widget representation:

location The URI of a widget resource. A widget resource is described by the WADL `resource_type` 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_type` element whose `id` is 'widgets'.

2.13 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[6] for full details.

A Additional Examples

A.1 Amazon Item Search

The following shows a WADL description of the Amazon item search service[7]:

```
<application xmlns="http://research.sun.com/wadl/2006/07"
  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 path="xml">
      <method href="#ItemSearch"/>
    </resource>
  </resources>

  <method name="GET" id="ItemSearch">
    <request>
      <param name="Service" style="query"
        fixed="AWSECommerceService"/>
      <param name="Version" style="query" fixed="2005-07-26"/>
      <param name="Operation" style="query" fixed="ItemSearch"/>
      <param name="SubscriptionId" style="query"
         type="xsd:string" required="true"/>
      <param name="SearchIndex" style="query"
         type="aws:SearchIndexType" required="true">
         <option value="Books"/>
         <option value="DVD"/>
         <option value="Music"/>
      </param>
      <param name="Keywords" style="query"
        type="aws:KeywordList" required="true"/>
      <param name="ResponseGroup" style="query"
        type="aws:ResponseGroupType" repeating="true">
         <option value="Small"/>
         <option value="Medium"/>
         <option value="Large"/>
         <option value="Images"/>
      </param>
    </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. 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.
- The `SearchIndex` and `RepsonseGroup` parameters both have an enumerated set of possible values. In both cases only a subset of possible values is shown to minimize the length of the example.

A.2 Atom Publishing Protocol

The Atom publishing protocol[8] 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      </collection>
8      <collection
9        title="Pictures"
10       href="http://example.org/reilly/pic" >
11       <member-type>media</member-type>
12     </collection>
13   </workspace>
14 </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 the Atom protocol methods associated with feeds, associated representations and resource types. 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/2006/07"
  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>

  <resource_type id="entry_feed">
    <method href="#getFeed"/>
    <method href="#addEntryCollectionMember"/>
  </resource>

  <resource_type id="media_feed">
    <method href="#getFeed"/>
    <method href="#addEntryCollectionMember"/>
    <method href="#addMediaCollectionMember"/>
  </resource>

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

  <representation id="feed" mediaType="application/atom+xml"
    element="atom:feed">
    <param name="first_link" style="plain"
      path="/atom:feed/atom:link[@rel='first']">
      <link href="#feed_resource" rel="first"/>
    </representation>
    <param name="next_link" style="plain"
      path="/atom:feed/atom:link[@rel='next']">
      <link href="#feed_resource" rel="next" rev="previous"/>
    </param>
    <param name="prev_link" style="plain"
      path="/atom:feed/atom:link[@rel='previous']">
      <link href="#feed_resource" rel="previous" rev="next"/>
    </param>
    <param name="last_link" style="plain"
      path="/atom:feed/atom:link[@rel='last']">
      <link href="#feed_resource" rel="last"/>
    </param>
  </representation>

  <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="addMediaCollectionMember">
      <request>
        <representation href="#entry"/>
        <representation />
      </request>
    </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/2006/07"
    xml:base="http://purl.org/atom/app.wadl"
    xmlns:app="http://purl.org/atom/app#">

    <resources base="http://example.org/">
      <resource path="reilly/main"
        type="http://purl.org/atom/app.wadl#entry_feed"
        app:member-type="entry"/>
      <resource path="reilly/pic"
        type="http://purl.org/atom/app.wadl#media_feed"
        app:member-type="media"/>
    </resources>
  </application>
```

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/pic`
  This resource supports HTTP GET to retrieve an Atom feed document and HTTP POST to add a new entry or other media to 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 although that information is also implicit in the type of each resource.

B RelaxNG Schema for WADL

```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace local = ""
namespace wadl = "http://research.sun.com/wadl/2006/10"

start =
  element wadl:application {
    doc*,
    grammars?,
    resources?,
    ( resource_type | method | representation | fault )*,
    foreign-attribute,
    foreign-element
  }
doc =
  element wadl:doc {
    attribute xml:lang { languageTag }?,
    attribute title { text }?,
    ( text | foreign-element )*,
    foreign-attribute
  }
grammars =
  element wadl:grammars {
    doc*,
    incl*,
    foreign-element
  }
incl =
  element wadl:include {
    doc*,
    attribute href { xsd:anyURI },
    foreign-attribute
  }
resources =
  element wadl:resources {
    doc*,
    resource+,
    attribute base { xsd:anyURI },
    foreign-attribute,
    foreign-element
  }
resource_type =
  element wadl:resource_type {
    doc*,
    param*,
    method*,
    attribute id { xsd:token }?,
    foreign-element,
    foreign-attribute
  }
resource =
  element wadl:resource {
```

```
      doc*,
      param*,
      (method | resource)*,
      attribute type { list {xsd:anyURI} } ?,
      attribute path { text }?,
      attribute id { xsd:token }?,
      attribute queryType { text }?,
      foreign-element,
      foreign-attribute
    }
  method =
    element wadl:method {
      (
        (
          attribute href { xsd:anyURI }
        ) | (
          doc*,
          request?,
          response?,
          attribute id { xsd:token }?,
          attribute name {
            "DELETE" | "GET" | "HEAD" | "POST" | "PUT" | xsd:token
          }
        )
      ),
      foreign-element,
      foreign-attribute
    }
  request =
    element wadl:request {
      doc*,
      param*,
      representation*,
      foreign-attribute,
      foreign-element
    }
  response =
    element wadl:response {
      doc*,
      param*,
      (representation | fault)*,
      foreign-attribute,
      foreign-element
    }
  representation_type =
    (
      (
        attribute href { xsd:anyURI }
      ) | (
        doc*,
        param*,
        attribute id { xsd:token }?,
        attribute element { xsd:QName }?,
        attribute mediaType { text }?,
```

```
      attribute profile { list { xsd:anyURI} }?,
      attribute status { list { xsd:int+ } }?
    )
  )
representation =
  element wadl:representation {
    representation_type,
    foreign-attribute,
    foreign-element
  }
fault =
  element wadl:fault {
    representation_type,
    foreign-attribute,
    foreign-element
  }
param =
  element wadl:param {
    doc*,
    option*,
    link?,
    attribute name {xsd:token },
    attribute style {
      "plain" | "query" | "matrix" | "header" | "template"
    },
    attribute id { xsd:token }?,
    attribute type { text }?,
    attribute default { text }?,
    attribute path { text }?,
    attribute required { xsd:boolean }?,
    attribute repeating { xsd:boolean }?,
    attribute fixed { text }?,
    foreign-element,
    foreign-attribute
  }
option =
  element wadl:option {
    doc*,
    attribute value { xsd:string },
    foreign-element,
    foreign-attribute
  }
link =
  element wadl:link {
    doc*,
    attribute resource_type { xsd:anyURI }?,
    attribute rel { xsd:token }?,
    attribute rev { xsd:token }?,
    foreign-element,
    foreign-attribute
  }
foreign-attribute = attribute * - (wadl:* | local:* | xml:*) { text }*
foreign-element =
  element * - (wadl:* | local:*) {
```

```
     (attribute * { text }
      | text
      | any-element)*
  }*
any-element =
  element * {
     (attribute * { text }
      | text
      | any-element)*
  }*
languageTag = xsd:string {
   pattern = "[A-Za-z]{1,8}(-[A-Za-z0-9]{1,8})*"
}
```

C XML Schema for WADL

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://research.sun.com/wadl/2006/10"
  xmlns:tns="http://research.sun.com/wadl/2006/10"
  xmlns:xml="http://www.w3.org/XML/1998/namespace"
  elementFormDefault="qualified">

  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <xs:element name="application">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:grammars" minOccurs="0"/>
        <xs:element ref="tns:resources" minOccurs="0"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="tns:resource_type"/>
          <xs:element ref="tns:method"/>
          <xs:element ref="tns:representation"/>
          <xs:element ref="tns:fault"/>
        </xs:choice>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="doc">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="title" type="xs:string"/>
      <xs:attribute ref="xml:lang"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="grammars">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="tns:include"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```
  <xs:element name="resources">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:resource" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="base" type="xs:anyURI"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="resource">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:param" minOccurs="0" maxOccurs="unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="tns:method"/>
          <xs:element ref="tns:resource"/>
        </xs:choice>
        <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##other"
          processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="type" type="tns:resource_type_list"/>
      <xs:attribute name="queryType" type="xs:string"
        default="application/x-www-form-urlencoded"/>
      <xs:attribute name="path" type="xs:string"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="resource_type_list">
    <xs:list itemType="xs:anyURI"/>
  </xs:simpleType>

  <xs:element name="resource_type">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:param" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:method" maxOccurs="unbounded"/>
        <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##other"
          processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="method">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:request" minOccurs="0"/>
        <xs:element ref="tns:response" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="name" type="tns:Method"/>
      <xs:attribute name="href" type="xs:anyURI"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="Method">
    <xs:union memberTypes="tns:HTTPMethods xs:NMTOKEN"/>
  </xs:simpleType>

  <xs:simpleType name="HTTPMethods">
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="GET"/>
      <xs:enumeration value="POST"/>
      <xs:enumeration value="PUT"/>
      <xs:enumeration value="HEAD"/>
      <xs:enumeration value="DELETE"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="include">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="href" type="xs:anyURI"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="request">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:param" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:representation" minOccurs="0"
          maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="response">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:param" minOccurs="0" maxOccurs="unbounded"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="tns:representation"/>
          <xs:element ref="tns:fault"/>
        </xs:choice>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="uriList">
    <xs:list itemType="xs:anyURI"/>
  </xs:simpleType>

  <xs:complexType name="representation_type">
    <xs:sequence>
      <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="tns:param" minOccurs="0" maxOccurs="unbounded"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
    <xs:attribute name="element" type="xs:QName"/>
    <xs:attribute name="status" type="tns:statusCodeList"/>
    <xs:attribute name="mediaType" type="xs:string"/>
    <xs:attribute name="href" type="xs:anyURI"/>
    <xs:attribute name="profile" type="tns:uriList"/>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:simpleType name="statusCodeList">
    <xs:list itemType="xs:unsignedInt"/>
  </xs:simpleType>

  <xs:element name="representation" type="tns:representation_type"/>

  <xs:element name="fault" type="tns:representation_type"/>

  <xs:simpleType name="ParamStyle">
    <xs:restriction base="xs:string">
      <xs:enumeration value="plain"/>
      <xs:enumeration value="query"/>
      <xs:enumeration value="matrix"/>
      <xs:enumeration value="header"/>
      <xs:enumeration value="template"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="param">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:option" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="tns:link" minOccurs="0"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
      <xs:attribute name="style" type="tns:ParamStyle" use="required"/>
      <xs:attribute name="id" type="xs:ID"/>
      <xs:attribute name="type" type="xs:QName" default="xs:string"/>
      <xs:attribute name="default" type="xs:string"/>
      <xs:attribute name="required" type="xs:boolean" default="false"/>
      <xs:attribute name="repeating" type="xs:boolean" default="false"/>
      <xs:attribute name="fixed" type="xs:string"/>
      <xs:attribute name="path" type="xs:string"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="option">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="value" type="xs:string" use="required"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="link">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tns:doc" minOccurs="0" maxOccurs="unbounded"/>
        <xs:any namespace="##other" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="resource_type" type="xs:anyURI"/>
      <xs:attribute name="rel" type="xs:token"/>
      <xs:attribute name="rev" type="xs:token"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:complexType>
  </xs:element>

</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] J. Gregorio, M. Hadley, M. Nottingham, and D. Orchard. URI Template. Internet Draft, IETF, October 2006. See http://www.ietf.org/internet-drafts/draft-gregorio-uritemplate-00.txt.

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

[5] James Clark and Steve DeRose. XML Path Language (XPath) 1.0. Recommendation, W3C, November 1999. See http://www.w3.org/TR/xpath.

[6] 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.

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

[8] 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

Thanks to the members of the http://lists.w3.org/Archives/Public/public-web-http-desc/ mailing list who provided useful feedback on several iterations of this specification. Mark Nottingham and John Nienart (Yahoo!) provided extensive feedback and helped structure the overall design.

Copyright Notice