Using canonical tree data structures for web services

Josef Spillner <js177634@inf.tu-dresden.de>

December 18, 2005

In today's argumentation for web services, interoperability is often praised as one of the strongest advantages over legacy services. The basis for this argumentation is the usage of the SOAP protocol, which is an XML container format for messages, which in the case of web services are also formatted in XML, and specified using the XML Schema Definition within the web service description files which are kept in WSDL format.

It is however known that SOAP is far from being efficient for the scenarios which occur in the domain of web services. It includes type information which is redundantly available in the static message definition part of the WSDL, and due to the strict XML compliance, a lot of namespace information is sent across the wire which only contains boilerplate assignments like those for XSI and XSD (XML Schema), SOAP-ENC, SOAP-ENV, and the SOAP encoding style.

An alternative message format for web services has been proposed under the name REST, and is widely popular with many service providers. It consists of a list of key-value pairs, which are simply passed to the service. The obvious drawback is that only flat information can be passed, no tree structures can be preserved.

In this paper, it is suggested to combine the simplicity aspects of REST with the tree structure power of SOAP to result in a message format which, under certain circumstances, can actually be fully downward compatible with pure REST. Note that this is not always needed though - the proposed format is transport independent, while the domain of REST is clearly bound to HTTP, which just like in the case of SOAP could lead to another discussion about interoperability versus efficiacy.

1 Canonical tree structures

While imperative programming languages do in most cases not support trees as native data types, list-oriented languages which can handle nested lists have existed for a long time, with LISP probably being one of the most prominent examples. In LISP, a method call is expressed as `(method param)` , that is, a concatenation of the method name and all of its arguments. The notation of

this paper suggests using LISP-style trees, with two important remarks: First, for the sake of simplicity, no string markers in the form of `(method 'string)` or `'('list 'with' 'four 'strings)` are used. Second, the format is considered identical to the imperative model of saying `method(param)` . Both are syntactical differences only which might depend on the success of the adoption of the suggested canonical tree structures by web service providers.

2 Modelling method calls

There are basically three different kinds of parameters which will have to be modelled in the canonical tree structures. The first of them is a simple method call with a dictionary-based key-value mapping. In the tree structure, this call looks like the following: `(Request (param val))` The second kind is a slight extension, permitting multiple parameters to be passed in an anonymous list, that is, a list which does not have a name: `(Request (param val) (param val))` The third kind of parameter is a named list, which in the tree structure refers to a branch node, with all further parameters being the leaves. `(Request (List firstval secondval))`

Now when considering a response as given by a web service, the following message takes the abovementioned cases into account and represents a tree node with four leaves, all of which are being named.

```
(Test:QueryResponse
(url http://localhost/)
(server SomeServer)
(version 1.0)
(ext:binary true))
```

The use of namespaces has not yet been discussed, but is suggested to be included in a generic way, that is, namespaces could be part of the message using a reserved name: `(ns Test http://localhost/namespace/test)`

3 Usage over HTTP

Just like with the case of SOAP, the canonical tree messages are best sent as payload data, either embedded into a host protocol like HTTP, or even standalone. The reason for this recommendation is that if the messages get very large, sending them as part of a HTTP URL (in a GET/POST operation) might not be supported by the server, or be restricted by the HTTP protocol in size.

Yet for smaller messages, a possible means of attaching tree structures directly to URLs will be discussed here. In the generic form, the parameter will look like the following:

```
foo.cgi?query=(Test:QueryResponse%20(url%20http://localhost))
```

As one can see, spaces have been replaced by %20, as is mandated by the url-encoding standard. The number after the percent sign equals the ASCII code of the replaced character. Similarly, opening and closing brackets within

strings might have to be escaped by %28 and %29, respectively, since as opposed to the case of sending the structure as a payload message, line-based formatting is not available and therefore creating a deterministic parser would be difficult to impossible.

Likewise, strings which contain spaces should be marked as being one string, as otherwise the space decoding would split them apart. It is recommended to surround them with quote signs, which in turn means that existing quote signs in strings should be url-encoded with %22.

An example is explained now. Assuming that the string `'hello world' '(""cliché"")'` should be sent (with the ' signs only visualising the string boundaries), encoding happens in multiple steps. At first, the custom url-encoding is applied, which results in `'hello world' '%28%22cliché%22%29'`. Afterwards, the strings are marked, so that the visualisation used here can now be dropped: `""hello world"" %28%22cliché%22%29` Right afterwards, the generic url-encoding happens, which not only replaces the space signs, but also non-ASCII characters like é which becomes %C3%A9. In a query, the following result is obtained: `(query%20""hello%20world""%20%28%22clich%C3%A9%22%29)`

Resolving this query string is only a matter of applying the previous encoding steps in reverse order.

4 REST compatibility

To achieve REST compatibility, one has to consider that the concatenation of conventional key-value parameters happens with the & sign, and assignments with the = operator. The tree structure `(method (param value onemore) (list (param value)))` would then be translated to

```
?method[0]=p:1:param&1:param[0]=value&1:param[1]=onemore
&method[1]=p:1:list&1:list[0]=p:2:param&2:param[0]=value
```

The p: namespace refers to other parameters and solves the problem of name clashes.

5 Summary

This paper has presented a viable approach of introducing canonical tree structures into web services, which is particularly useful for the case of REST-based services. The main advantage is getting rid of a lot of overhead of the SOAP protocol, which in most cases does not seem to be needed, nor used in practice.

No examination of higher-level web service concepts like the WS-* family of extensions has been done. A follow-up paper could look into including those concepts into the realm of canonical tree structures.