***I**nternational*

***V**irtual*

***O**bservatory*

***A**lliance*

REST in the VO

Version 0.22

IVOA Working Draft 2008 October 28

This version:
http://www.ivoa.net/Documents/WD/GWS/VOREST-20081028.doc
Latest version:
http://www.ivoa.net/Documents/latest/VOREST.html
Previous version(s):
…

Author(s):
André Schaaff, Norman Gray
IVOA Grid and Web Services Working Group

Abstract

In the early years of the VO, the SOAP Web Service paradigm was an important element of the IVOA Architecture. Developments around these services are more and more complex with an increasing number of standards (WS-* …). REST [3] is not a standard but a formalization of the URL use and it is easy to implement it. A service is RESTful if it follows a set of rules (which are not defined in a standard document). As there is no standard we think that it is necessary to define a minimal guideline about the “RESTfullness” in the VO context.

Status of This Document

This is an IVOA Working Draft. The first release of this document was 2008 May 05.

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress".

A list of current IVOA Recommendations and other technical documents *can be found at http://www.ivoa.net/Documents/.*

Acknowledgements

This document derives from discussions among the Grid and Web Services working group of the IVOA in Beijing (May 2007), Cambridge (September 2007) and Trieste (May 2008).

Contents

1 Introduction 3
2 Resource Oriented Architecture [9] 3
3 Representational State Transfer 3
3.1 Quick definition of REST and RESTful 3
3.2 Quick comparison with SOAP services 4
3.3 How to describe a service? 4
3.4 WADL 5
4 REST oriented frameworks and tools (not exhaustive) 7
4.1 Ruby on Rails [4] 7
4.2 Restlet [5] 7
4.3 Django [6] 7
4.4 NetKernel [10] 7
4.5 Apache CXF [11] 7
4.6 Comments 7
5 Restfulness in the VO 8
5.1 Interoperability problems 8
5.2 Recommendations 8
5.3 Work to done 8
6 Conclusion 8
Appendix A: "Appendix Title" 9
References 9

1 Introduction

REST (Representational State Transfer) [3] is not a standard but a formalization of the URL use. It was introduced in 2000 in Roy Fielding’s Ph.D thesis [3]. It refers to a set of network architecture principles which describe how to define and address resources. There is no REST [3] standardization process and the term “REST” is often used when the service is not based on standards like SOAP.

A service is RESTful if it follows a small set of rules (which are not defined in a standard document). As there is no standard we think that it is necessary to define a minimal guideline about the “RESTfullness” in the VO context.

2 Resource Oriented Architecture [9]

A resource Oriented Architecture can be resumed to 4 concepts (resources, theirs names (URIs), their representation, the links between them) and 4 properties (addressability, statelessness, connectedness, uniform interface).

3 Representational State Transfer

3.1 *Quick definition of REST and RESTful*

In the REST approach, it may be sufficient to know the URI to access to a resource.

Examples:

1) http://www.example.com/sky/m31/pictures

2) http://www.example.com/sky/m31/picture/1

In these examples it is possible to access to the information through a simple URL without the use of a specific tool (for C#, Java, Perl …) on the client side, the client has just to read simply the URL.
In example 1) the URL returns the number of available pictures for m31 and in example 2) the URL returns the first picture.

Main feature of a REST service, from [9]:

- Architectural style of the Web.
- Resources are addressable (URIs).
- Interact with representations of resources.
- State is maintained within a resource representation.
- Small set of methods that can be applied to any resource (HTTP methods).
- Scaleable, low cost of coordination.

To be RESTful a service must be compliant with the following principles:

- Addressability
- Stateless
- Connectivity
- Uniform interface

3.2 Quick comparison with SOAP services

If we take again the two previous examples of REST [3] URLs,

1) http://www.example.com/sky/m31/pictures

2) http://www.example.com/sky/m31/picture/1

In SOAP we will have to define something like int getPictures (String object) for 1) and to use for example the SOAP with an attachment mechanism or to return the URL of the image for 2).

SOAP Web Service engines are also evolving by implementing the REST [3] alternative. For example, in Axis 2 it is possible for a client to specify that he want to access the service following the REST [3] paradigm.

3.3 How to describe a service?

WSDL (Web Services Description Language) [7] has been created to describe SOAP services. These services are self-described by interrogation of the service endpoint URL with an extension like “?wsdl”. It is very difficult to use a SOAP Web Service without this description like when you try to use a Java API without the corresponding Javadoc. For REST [3], there is no standard way like WSDL [7] but it is possible to use for example WADL (Web Application Description Language) [8]. RESTful services have simpler interfaces and the description is not as important as for SOAP WS. But in the case of automatic creation or use of the services by tools it is necessary to provide a description of the services. If a

WADL [8] description is available it is then possible to generate for example the Java client code to query the service.

3.4 WADL

WADL (Web Application Description Language) [8] is a draft specification for an analogue to the WSDL language, specialized to RESTful interfaces.

The WADL distribution includes an XML schema for WADL files, schema documentation, and some tools which generate documentation and client-side Java stubs from an input WADL file. The specification is an advanced draft, but it is not clear where further standardization will take place, nor when. As with WSDL, the goal with WADL is to document an interface in a machine-readable form. For example, the following WADL file describes a simple interface which allows clients to GET HTML representations of resources:

```
<application xmlns="http://research.sun.com/wadl/2006/10">
  <resources base='http://example.org/resource'>
    <resource path='{resourceName}'>
      <doc>This resource is one of a set of resources</doc>
      <param name='resourceName' style='template'>
        <doc>The name of the resource being described</doc>
      </param>
      <method name='GET'>
        <response>
          <representation status='200' mediaType='text/html'>
            <doc>Returns a description of the resource</doc>
          </representation>
          <fault status='404' mediaType='text/html'>
            <doc>If the document is not found, return an explanation</doc>
          </fault>
        </response>
      </method>
    </resource>
  </resources>
</application>
```

This describes a set of resources http://example.org/resource/{resourceName} for different values of the 'variable' {resourceName}. A GET request may produce one of two responses, namely a text/html response with a 200 status, or another text/html response, describing an error, with a 404 status.

In the case of WS-* services, a WSDL description is almost essential; there are so many technicalities involved in making a WSDL call, that a client application author is almost certain to make mistakes if they attempt to implement the client interface by hand. Also, the expectation of WS-* services is that the contents of the request and response are representations of program objects, which must be serialized into a request, and desterilized from a response, again introducing many opportunities for error.

The REST paradigm, however, avoids the fragility of WS-* services, by insisting that the contents of HTTP responses (and HTTP requests where appropriate) be _representations_ of the corresponding resources, encoded in one or other MIME type included in the HTTP request or response. This implies that the problems of serialization and deserialization are external to the protocol; this makes the interface easier to describe, with the strong advantage that the separation between the interface and the representations it carries is more clearly distinct.

Although this appears to place more of a burden on the client application, this is not the case in practice: since an application generally has to ingest representations of resources anyway, from files or other URLs, it is usually capable of ingesting representations from a RESTful interface without difficulty. What this means in turn is that a RESTful interface is generally rather straightforward to implement on the client side, with a lot less 'glue' which is specific to the interface.

In consequence, there is a much lesser need for the sort of generated code stubs which are a crucial output of WSDL tools. This is fortunate, since the code-generation tools distributed with the WADL standard seem immature, sometimes failing on valid input, and appearing to work naturally only for that subset of services which have a predominantly keyword-value GET interface.

In the experience of one of the present authors ([12], NG), a WADL file is a useful component of a RESTful service, even without any generated client code. The WADL file provides a usefully explicit specification of the service's interface, which was used to generate human-readable documentation and, using another custom XSL transformation, to generate code which verified that the service's test cases exercised the entire interface, and did not violate it at any point. A variant of this checking code could have been (but was not in fact) included within the server to guarantee that the service's responses matched its interface promises.

Thus on this and similar cases, the WADL file was useful enough, internal to code-base, to justify its use, and offering the WADL file to users of the service was an added bonus.

Different works about the generation of a WADL file for a service and about the code generation in different language from a WADL description (Java, Python, Ruby…) are on going. See and try for example [13].

4 REST oriented frameworks and tools (not exhaustive)

4.1 Ruby on Rails [4]

Rails is a framework written in Ruby and dedicated to Web developments.
See [4] for more details.

4.2 Restlet [5]

Restlet is a framework for the Java platform providing native REST [3] support.
See [5] for more details.

4.3 Django [6]

Django is an open source framework for the Python expected to provide a native REST [3] support in the coming months.
See [6] for more details.

4.4 NetKernel [10]

NetKernel is an implementation of a resource-oriented computing abstraction. It can be thought of as an internet-like operating system running on a microkernel within a single computer. It is available with 2 kinds of license: open source and commercial.
See [10 for more details.

4.5 Apache CXF [11]

Apache CXF is an open source services framework designed to build and develop services using front-end programming APIs. Protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and of transports such as HTTP, JMS or JBI are possible.
See [11] for more details.

4.6 Comments

REST development or compliant frameworks are evolving quickly so we think that it is difficult to recommend a restricted set of them.

5 Restfulness in the VO

5.1 Interoperability problems

As said in a previous part of this document, REST [3] is not a standard. The SOAP approach which was a key element in the IVOA architecture is a standard but is also crushed under a huge stack of related standards (WS-*).
As VO services are not just designed for humans but also to be queried by another tools, it would be very efficient to have a “standardized” description of the REST [3] services provided in the frame of the IVOA.
It could also be useful to have a tool to check if the service follows a minimal set of rules.

5.2 Recommendations

Since REST is not a formal standard, but instead a set of good practices, services cannot be required to 'conform' to REST. We RECOMMEND, however, that future services should conform to these good practices wherever possible, and that service authors should seek feedback on their interface design from the GWS WG, with a view to ensuring that the service conforms to the spirit of these practices as much as possible.

Although WADL is not, or not yet, a standard, we cannot require conformance to that, either. However the practical advantages to a service implementation of having a machine-readable interface specification (as described in section 3.4 above), and the interoperability advantages of having a public commitment to an interface, are substantial enough that we RECOMMEND that services publish a WADL file.

5.3 Work to done

The WADL spec (section 5) suggests that WADL documents should be served using the application/vnd.sun.wadl+xml MIME type, and there are suggestions elsewhere that they should be available at a http://example.org/service?wadl URL (though this is not mentioned in the WADL spec). Should we echo this, and generally be more prescriptive here?

6 Conclusion

Compared to SOAP Web Services REST [3] is a lite way to provide a Web service following just a reduced set of rules. But it is perhaps necessary to define clearly the basis of what an interoperable RESTful VO service must be. REST [3] is more human oriented than SOAP but it defines no standard concerning the description of the service which is important in the case of a dynamic use by other services. At least we recommend to provide the description of a REST service through a formalism like WADL.

Appendix A: “Appendix Title”

…

References

[1] R. Hanisch, *Resource Metadata for the Virtual Observatory* , http://www.ivoa.net/Documents/latest/RM.html
[2] R. Hanisch, M. Dolensky, M. Leoni, *Document Standards Management: Guidelines and Procedure* , http://www.ivoa.net/Documents/latest/DocStdProc.html
[3] R. Fielding, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[4] Ruby on Rails, http://www.rubyonrails.org/
[5] Restlet, http://www.restlet.org/
[6] Django, http://www.djangoproject.com/
[7] WSDL, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/
[8] WADL, https://wadl.dev.java.net/
[9] RESTful Web Services, L. Richardson and S. Ruby, O’Reilly
[10] NetKernel, www.1060.org
[11] Apache CXF, http://cxf.apache.org
[12] SKUA project, http://myskua.org/
[13] REST Describe (and compile), http://tomayac.de/rest-describe/latest/RestDescribe.html