Representational State Transfer (REST) for Feature Configuration in Session Initiation Protocol (SIP)
draft-griffin-bliss-rest-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on April 30, 2009.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Abstract

In order to provide interoperable implementations of certain Session Initiation Protocol (SIP) based features, such as automated call handling, it is necessary for devices to configure network servers with feature data. An example is a call forwarding number for a call forwarding service. This document introduces the concept of Representational State Transfer (REST) and gives an example of how REST can be used for such configuration.
Table of Contents

1. Introduction ................................................. 3
2. Overview of REST ........................................... 3
   2.1. Design Principles of a REST Based System ............... 4
   2.2. Advantages and Disadvantages of a REST Based
        Architecture ........................................... 4
           2.2.1. REST Advantages ................................. 5
           2.2.2. REST Disadvantages ............................. 5
   2.3. REST Architecture ..................................... 5
3. Existing REST Example ........................................ 6
4. Example REST API for Call Forwarding ....................... 8
5. Security Considerations ..................................... 9
6. Informational References .................................... 9
Authors’ Addresses ............................................. 10
Intellectual Property and Copyright Statements ............. 11
1. Introduction

The Session Initiation Protocol (SIP) [RFC3261] can be used to realize numerous communications features. These include basic telephony functions, such as call hold, transfer, and automated call handling. However, in practice, there has been limited interoperability for many of these advanced features. The BLISS working group has been chartered to improve this interoperability [I-D.ietf-bliss-problem-statement]. One of the current areas of study is automated call handling (ACH) [I-D.ietf-bliss-ach-analysis]. ACH includes features like call forwarding, forward to voicemail, and do-not-disturb, all of which involve an automated decision on treatment of a call based on some kind of condition.

In order to achieve interoperability for ACH types of features, it is necessary to have a common mechanism by which endpoints - such as hard phones and soft clients - can configure data required for operation of those features. As a simple example, when a user presses the ‘call forward all’ button on their phone, this needs to cause a change in call handling configuration on their server. That configuration information - whether call forwarding is enabled or not - is then used by the server when receiving an incoming call.

This document introduces the concept of Representational State Transfer (REST) as a potential solution to this problem. It gives an example of an existing web service using REST, and shows some example REST exchanges for configuration of call forwarding. This document is not meant as a concrete proposal for specific REST schemas, but rather, to facilitate discussion around the concept.

2. Overview of REST

Representational State Transfer or REST is an architectural style for networked systems as described in Dr. Roy Fieldings Ph.D. dissertation [Fielding-PHD]. It is not dependant upon any particular protocol. This means that it is possible to create a REST based system that is not built upon HTTP. However, most practical implementations of REST are built on HTTP. REST is not a standard however it does prescribe the use of standards, including but not limited to:

- HTTP
- URL
- XML, HTML, GIF, JPEG (representations of resources)
2.1. Design Principles of a REST Based System

Stateless: Every request from client to server must contain all the information required to execute the request and must not rely on information known to the server.

Uniform interface to support state transfer consisting of:

* A constrained set of well defined operations e.g. the HTTP methods GET, PUT, POST, DELETE.
* A constrained set of content types e.g. text/xml, image/jpeg.

Client server pull interaction: consuming clients pull representations.

Uniquely Named Resources: A URI is used to name the resources which comprise the system.

Layered, interconnected resource representations: resource representations are interconnected using URLs enabling a client to progress through states.

Cacheable responses to promote network efficiency.

An important principle of REST is that of resources. A resource is a source of specific information which is named by a URI. Resources are manipulated by network components using a uniform interface e.g. HTTP. Resulting state changes for the resource are returned as representations e.g. XML document. As REST interfaces are by definition highly connected via URI, deeply linked representations are typically used over wide or shallow representations. This means that a representation returns specifically the data that it represents but is tightly linked to related data that an application might be interested in.

2.2. Advantages and Disadvantages of a REST Based Architecture

There are several advantages and disadvantages to REST based architectures. Indeed in Chapter 5 of the previously referenced doctoral dissertation from Dr Roy Fielding, many of these are described and discussed. What follows are some of the more commonly perceived advantages and disadvantages associated with REST.
2.2.1. REST Advantages

- Offers possibilities for thin client development as less client code is required.
- Does not require explicit resource discovery mechanism due to hyperlinking.
- Scalable architecture compared with those that require stateful servers.
- Caching promotes network efficiency and fast response times.
- Software versioning benefits including support of document type evolution such as HTML and XML without impacting backward or forward compatibility.
- Resource extensibility, allowing support for new content types without impacting existing and legacy content types.

2.2.2. REST Disadvantages

- HTTP as a uniform interface presents technical challenges for real time asynchronous events to a thin client or browser based application.
- Managing URI Namespace can be cumbersome.
- Lacks supporting software tools
- Can impact network performance by encouraging more frequent client-server requests and responses.

2.3. REST Architecture

Figure 1 shows an architecture for client-server communications based on REST. It can be seen that the uniform interface is presented by HTTP and the resource representations are shared over this interface. The REST aware client code shown is typically part of a web page that is loaded from a web server. JavaScript is commonly used as a client side language embedded in HTML to manipulate resources. Where data is returned it is parsed and acted upon by this client side code based on the logic of the application. Later we will see how the interface is defined with operations in the uniform interface (HTTP) with resources represented as URIs. The architecture also shows how several underlying standards such as HTTP, URL, HTML and XML can be used to support a REST based architecture.
HTTP Method
GET, POST, PUT DELETE

REST Payload
Representation with URI

<table>
<thead>
<tr>
<th>HTTP Client</th>
<th>REST Aware Client</th>
<th>REST Code</th>
<th>HTTP Server</th>
<th>REST Aware Server</th>
<th>REST Code</th>
<th>HTTP Response</th>
</tr>
</thead>
</table>

REST XML Return Data

Figure 1: REST Architecture

It is useful to map the HTTP methods to database CRUD operations. The mapping is quite straightforward:

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>CRUD Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>Create</td>
</tr>
<tr>
<td>GET</td>
<td>Read</td>
</tr>
<tr>
<td>PUT</td>
<td>Update</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete</td>
</tr>
</tbody>
</table>

3. Existing REST Example

This section provides a brief example of an existing REST-based API used in the Web.

The API at [http://apidoc.digg.com/ListStories](http://apidoc.digg.com/ListStories) is used by the Digg web application, to allow a client to retrieve stories about a particular topic. An example to retrieve three most recent stories about Apple:

```
http://services.digg.com/stories/topic/apple?count=3
&appkey=http%3A%2F%2Fexample.com%2Fapplication
```

Would produce the following example response:
<stories timestamp="1196894147" min_date="1194302130" total="2636" offset="0" count="3">
    <story id="4368401" link="http://maxsangalli.altervista.org/?p=45" submit_date="1196891534" diggs="1" comments="0" status="upcoming" media="news" href="http://digg.com/linux_unix/Jukebox_con_Linux">
        <title>Jukebox con Linux</title>
        <description>Jukebox with Linux</description>
        <user name="ilsanga" icon="http://digg.com/img/udl.png" registered="1196891377" profileviews="0" />
        <container name="Technology" short_name="technology" />
        <thumbnail originalwidth="390" originalheight="387" contentType="image/jpeg" src="http://digg.com/linux_unix/Jukebox_con_Linux/t.jpg" width="80" height="80" />
    </story>
    <story id="4367880" link="http://www.itsyourip.com/ip-tools/arping-ping-using-address-resolution-protocolarp/" submit_date="1196888736" diggs="2" comments="0" status="upcoming" media="news" href="http://digg.com/linux_unix/ARPing_Ping_using_Address_Resolution_Protocol_ARP">
        <title>ARPing - Ping using Address Resolution Protocol (ARP)</title>
        <description>ARPing is an utility that can be used to ping using Address Resolution Protocol (ARP) requests instead of using Internet Control Message Protocol (ICMP) as used in standard Ping utility. ARPing can be used for Duplicate Address Detection (DAD)</description>
        <user name="rbkumaran" icon="http://digg.com/users/rbkumaran/l.png" registered="1151050300" profileviews="42" />
        <container name="Technology" short_name="technology" />
    </story>
    <story id="4367579" link="http://linuxdevices.com/news/NS9372729455.html" submit_date="1196887377" diggs="5" comments="0" status="upcoming" media="news" href="http://digg.com/linux_unix/EMF_changes_tune_hails_embedded_Linux">
        <title>EMF changes tune, hails embedded Linux</title>
        <description>Embedded Market Forecasters has issued a report claiming that embedded Linux is as dependable as other real time operating systems (RTOSs). The independently funded report appears to recant EMF's controversial Microsoft-funded report in 2003 that claimed that embedded Windows OSes were far faster and cheaper than embedded Linux. ...</description>
        <user name="gadgeek" icon="http://digg.com/img/udl.png" registered="1131035306" profileviews="1717" />
        <container name="Technology" short_name="technology" />
    </story>
</stories>
In general, the HTTP URI is structured to provide different data depending on the fields present. As an example:

GET /stories
All stories.
GET /stories/popular
Popular stories.
GET /stories/upcoming
Upcoming stories.
GET /stories/top
Top stories.
GET /stories/hot
Hot stories.
GET /stories/container/{container name}
All stories from a given container.
  404 Not Found if the container does not exist.
GET /stories/container/{container name}/popular
Popular stories from a given container.
  404 Not Found if the container does not exist.
GET /stories/container/{container name}/upcoming
Upcoming stories from a given container.
  404 Not Found if the container does not exist.
GET /stories/container/{container name}/top
Top stories from a given container.
  404 Not Found if the container does not exist.

See how the natural structure of the URI itself provides a hierarchical view on the data, and how URI parameters are used to represent parameters of queries.

4. Example REST API for Call Forwarding

Based on this simple model, the following is a view on how one might define a REST API for setting a call forward no-answer service.

To set CFNA:

PUT systemusers/bob/cfna HTTP/1.1
<?xml version="1.0"?>
<cfna>enable</cfna>

To get current CFNA state:

GET systemsusers/bob/cfna HTTP/1.1

Would return:

HTTP/1.1 200 OK
<?xml version="1.0" ?>
<cfna>enabled</cfna>

It is also possible to define a model for setting CFNA on a line-by-line basis, rather than per user. In that case, the URI would be structured as device/line/CFA. In the case of a device called "BobsPhone" and line 4622:

PUT BobsPhone/4622/cfna HTTP/1.1
<?xml version="1.0" ?>
<cfna>enable</cfna>

5. Security Considerations

There are no security considerations associated with this specification.

6. Informational References


[I-D.ietf-bliss-ach-analysis]

[Fielding-PHD]

Authors’ Addresses

Keith Griffin
Cisco
Oranmore Business Park, Galway
Ireland

Email: kegriffi@cisco.com

Jonathan Rosenberg
Cisco
Iselin, NJ
US

Email: jdrosen@cisco.com
URI: http://www.jdrosen.net