

An Overview of the REST Architecture

Alan Trick

Abstract—REST is an architectural style that is used throughout the World Wide Web. REST defines a number of constraints. REST is a client-server architecture and connections are stateless. REST uses standardized interfaces and self-descriptive messages (hypertext). REST is layered; these layers provide encapsulation, caching, and reduced complexity.

Index Terms—REST, Web Architecture, WWW.

I. INTRODUCTION

ONE of the fundamental aspects of modern computing is communication. Not only do people communicate with each other, but machines (or parts of machines) communicate together. A web browser requests a web page from a favourite web site, an email client asks a spell-checking library to spell-check an email, an operating system asks a graphics card how much memory space it has.

The architects of the World Wide Web realized that its full potential would only be realized if they could provide a universal method of electronic communication [1]. A standard naming system was required and certain constraints were needed to ensure scalability. The result of these requirements is an architectural style known as Representational State Transfer (REST).

II. DESCRIPTION OF REST

A. Definition of Key Terms

The most important element in REST is a resource. A resource is identified to by a URI (e.g. `http://example.org/foo/bar`). A resource has a representation (an HTML document, or a PNG image). Resources may have multiple representations.

In a REST system, blocks of communications consist of requests and responses. A client requests a resource from a server and then receives a response.

B. REST as an architectural style

A key characteristic of REST is loose coupling. The goal is to reduce the amount of artificial dependency and complexity within a given system [2]. To accomplish this, several rules have been set in place. REST is stateless; and it consists of transparent, encapsulated layers.

1) *The Importance of Being Stateless*: When communicating over a large electronic network, certain things should not be taken for granted. Often, the amount of time between messages is relatively long. Servers get restarted. Clients drop out in the middle of a conversation. The result is that the state of things is liable to change without warning.

Due to the volatile nature of state, REST has been designed to be stateless. Each message is self-contained [1]. Thus, the servers do not need to keep track of each individual request and where it came from, all they do is respond to requests as they come.

To further demonstrate the nature of state, compare the lists of shopping interactions in Appendix I and II. After the first interaction in Appendix I is completed, the shopper has 'state' (i.e. they are at the grocery store). If the shopper gets distracted, as shoppers often do, and goes to the video game store, the state is changed. In this case the next interaction will probably fail, because video game stores don't sell milk. In some cases it might succeed, but with unpredictable and incorrect results (e.g. asking for a RAID controller at a grocery store will probably result in a curious look and can of insecticide).

On the other hand, each interaction in the second list contains all the information needed to complete the request. The shopper does not need to keep track of it's state in between interactions. The first interaction can be skipped and the second one will still succeed.

2) *Transparent and Encapsulation*: Another feature of large electronic networks is that they are fluid. Technology and infrastructure changes rapidly so it is important that the network is able to work independent of the underlying details. Because of the inherent complexity of large networks, the importance of one component being able to work independent of another cannot be understated [3].

C. HTTP and REST

The language of HTTP contains two primary parts: nouns and verbs, adjectives. In addition, there are two more sections which that are often used: meta-data and content. Consider the following interaction:

Request: Get milk from grocery store. I want 1%, but 2% is acceptable too.

Response: Ok, here is 1% milk: [milk contents].

In the language of HTTP headers, the request might look something like this:

```
GET /milk HTTP/1.1
Host: grocery-store
Connection: close
Accept: example/milk-1p,example/milk-2p;q=0.5
```

And the response:

```
HTTP/1.1 200 OK
Content-Type: example/milk-1p
Connection: close
```

[the contents of the milk]

1) *Nouns*: In HTTP, the noun is a Uniform¹ Resource Identifier (URI). In the interaction above the noun is "milk from grocery store". As a URI it might look like `http://grocery-store/milk`. A URI may be split into two parts, resource and fragment (or Fragment Identifier).

The example URI was simply a resource (it had no fragment). A fragment is specified by a hash sign "#" with some text following it). Of the two parts, the Resource is the only one that is visible in HTTP interactions [5]. This is intentional. Continuing along with the above example, consider the URI `http://grocery-store/milk#ingredient-list`. In this example, it is still have same resource (milk from the grocery store), but a it specifies a different view of the resources (specifically, the list of ingredients in the milk).

Another important point about REST is that the name resources have should remain as consistent as possible (although its representation may change) [6]. This is increasingly important with the advent of hypertext and links. The World Wide Web is an interactive medium: resources are perpetually created and updated. Often they link to other resources. If a resource is renamed or goes missing, this causes a problem unless somethings stays behind to redirect incoming users to the updated location [7]. URIs should be designed to last.

One practise with naming URIs is to remove any non-essential information [8]. Consider the URL `https://www1.twu.ca/twupass/login/after.aspx`. The worst element is the `aspx` extension at the end. If the web application switches to another system, say PHP, the URI will have to be changed as well. Another problem is the `www1` subdomain, although changes to this this can be resolved easily with DNS, its still extra information that is completely irrelevant to the page itself.²

2) *Verbs*: The verb in HTTP is called a Method. Two common methods are `GET` and `POST`. A full list of methods is available in section 9 of RFC 2616 [9]. Each method has certain predefined semantics and rules associated with it. For example, `GET` is an idempotent which means that a `GET` request should not be used for say, posting a comment on a blog. Unfortunately, web authors have frequently abused these rules [10].

3) *Meta-data*: There were two pieces of meta-data in the example. The first was in the request: the request asked for either 1 percent milk or 2 percent, although 1 percent is preferred twice as much. Note that these, values correspond to media types (or mime types), the particular values here are not real media types, but rather ones which are reserved to be used as examples [11]. The response contains a statement that it is sending back 1 percent milk. There are many other kinds of meta-data in HTTP. Fields that specify the time of the response, what program is making the request, what software is running on the server, what language the response should be in, whether it can be compressed with `g-zip`, and so on.

¹Often called Universal Resource Locator, which was the term Tim Berners-Lee originally used. In RFC 2396 'Universal' was changed to 'Uniform' cause several the term 'Universal' was too ambitious [4].

²That said, the whole URL is something of a mistake. I'm honestly not sure it it's supposed to do. I don't think it knows either. When I found it, my browser was in the middle of redirecting itself in an infinite loop between that URL an another inscrutable URL.

4) *Content*: This is the part that the end user normally ends up viewing. In the above case, it is the milk itself. On the web it's often an HTML page or another electronic format like an PNG image. Content such as HTML may contain links to other resources on the Web. With the advent of XML, the ability to embed machine readable information is emerging. The result is something that is sometimes refered to as the "semantic web" and includes things like RDF and microformats [4].

III. PROBLEMS WITH REST

A. *Stateless isn't (quite) Stateless*

An astute reader may have noticed some details missing in the example of shopping instructions. There actually is some important stateful elements to the 'stateless' example. The customer starts out with money and exchanges it for milk and a RAID controller. The state has changed.

The goal of a 'stateless' protocol is not to entirely eliminate state, but to reduce and isolate it as much as possible. Because of this, a mechanism called cookies was added to the HTTP protocol [12]. However, this extension is a source for much confusion and problems among web applications and web browsers [1]. Misuse of cookies may cause side-effects such cookie stealing, cross site scripting, broken website navigation, and allow unauthorized access to sensitive data.

B. *Embedded Objects*

Not all content on the web is in hypertext markup and some of the other formats do not work well in a RESTful system. Adobe's proprietary Flash format is one example. The links within Flash 'videos' are not URIs, so navigating does not actually cause the browser to go anywhere. Flash does not understand Fragment Identifiers so it is impossible to link to a specific place within the clip.

C. *Scripting and XMLHTTPRequest*

Over the last few years, techniques to use scripting to make HTTP requests have emerged. The most common of which is XMLHTTPRequest. Some times this can be used to deliver useful and effective web page interactivity. Unfortunately, a common use of it is to avoid requesting full web pages. Instead of going to another URL, selecting a link will call a script that loads the pages content another way.

The upshot of it is that it breaks navigation in similar way to embedded objects like Flash. It is possible to get around this problem using Fragment Identifiers, but often such usability features are ignored by developers. This also creates accessibility issues because it requires the User Agent to have a functional Javascript interpreter.

IV. CONCLUSION

The REST Architecture and the styles that it brings forward is and important driver in building the Web into what it was designed to be — a universal store of information. The concept of universal resources opens many possibilities as well as challenges. A firm understanding of the issues at hand is critical if we hope to solve them.

APPENDIX I
STATEFUL SHOPPING INSTRUCTIONS

- 1) Go to grocery store
- 2) Buy milk
- 3) Go to computer store
- 4) Buy RAID controller

APPENDIX II
STATELESS SHOPPING INSTRUCTIONS

- Buy milk at grocery store
- Buy RAID controller at computer store

REFERENCES

- [1] R. T. Fielding and R. N. Taylor. (2000) Principled design of the modern web architecture. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/webarch_icse2000.pdf
- [2] H. He. (2003) What is Service-Oriented Architecture. [Online]. Available: <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>
- [3] M. zur Muehlen, J. V. Nickerson, and K. D. Swenson, "Developing web services choreography standards—the case of REST vs. SOAP," *Decision Support Systems*, vol. 40, no. 1, pp. 9–29, 2005.
- [4] T. Berners-Lee. (1998) Web architecture from 50,000 feet. [Online]. Available: <http://www.w3.org/DesignIssues/Architecture.html>
- [5] ——. (1998) Universal resource identifiers – axioms of web architecture. [Online]. Available: <http://www.w3.org/DesignIssues/Model.html>
- [6] (2006) Cool URIs don't change. [Online]. Available: <http://www.w3.org/Provider/Style/URI>
- [7] J. Udell, "Hyperlinks matter," *InfoWorld*, vol. 24, no. 20, p. 15, 2002.
- [8] T. Berners-Lee. (1995) The name myth – axioms of web architecture. [Online]. Available: <http://www.w3.org/DesignIssues/NameMyth.html>
- [9] "Hypertext transfer protocol – http/1.1," RFC 2616, June 1999.
- [10] J. Udell, "End http abuse," *InfoWorld*, vol. 27, no. 17, p. 42, 2005.
- [11] "Example media types for use in documentation," RFC 4735, Oct. 2006.
- [12] "Http state management mechanism," RFC 2965, Oct. 2000.