

# Why SOAP Sucks, Why SOAP Rocks

**Matt Sergeant**  
**AxKit.com Ltd**

# Roadmap

- What's wrong with SOAP?
- What's right with SOAP?
- How to build something better than SOAP!
- How to build mod\_perl web services with SOAP and XML-RPC
- NB: SOAP in these slides also means XMLRPC

# Why SOAP Sucks

# Hides HTTP Semantics

- Picture the following API (from Meerkat):

```
getCategories()  
getChannels()  
getChannelsByCategory()
```

- Now the process of calling that:
  - ◆ Construct XML request body
  - ◆ POST that to the url containing the service
  - ◆ receive 200 OK HTTP response
  - ◆ with XML response body detailing results and/or errors

# What happened to HTTP???

- HTTP supports error codes (5xx range)
- HTTP supports GET semantics
- HTTP supports different resources (Categories, Channels) with URLs
- HTTP is an applications protocol!

# Why this is an issue

- HTTP is a very evolved system
- GET is known to not modify state : cachability!
- Proxies can't do their job with SOAP or XML-RPC
- ... most "RPC" APIs are doing GET/SET (GET/PUT) type stuff
- A firewall might be installed to block HTTP PUT and DELETE...
- ... but RPC over just POST circumvents the firewall
  - ◆ Is this a good thing or a bad thing?

# Data hidden

- SOAP wraps data structures in an envelope
- This slightly obscures the data
- Why not just use plain XML?
- Alternatively, we could use MIME

# No Logging

- SOAP requires manual logging to be done
- e.g. to Syslog
- HTTP GET provides parameters in log file
- And you can analyse http logs with e.g. analog

# No partial transfers

- HTTP can resume transfers of large sets of data
- But only with GET
- POST has no resume facility
- Could be useful for perldoc.SOAP, and Red Hat Network XML RPC RPM installer

# Why SOAP Rocks

# Single URL

- Many development environments don't give you control of the server
- This creates the illusion that you can't create many URLs with one app
- But... it can work with PATH\_INFO
- e.g. `http://server/cgi-bin/server.cgi/Categories`
- With SOAP (or XMLRPC) all requests go to one URL
  - ◆ Gives you slightly more control

# Standardised!

- We could invent HTTP Web Services (NB: Not RPC!)
- But SOAP is here, and SOAP works
- Note: We could use SOAP serialisation over HTTP
- e.g. GET `http://server/Goats` returns SOAP serialisation of all Goats

# Development Tools

- A key advantage
- SOAP::Lite
- .NET
- IBM WebServices Developer Kit
- Nothing exists for HTTP Web Services

# How to build a web services application using 1990s technology (theoretically)

# Technologies used

- Use mod\_perl to control URL space
- Use SOAP serialisation (from SOAP::Lite)
- Develop client libraries using perl HTTP modules

# Anatomy of a request

- RPC equivalent of getGoats() :
- GET http://server/Goats
- 200 OK
- ... SOAP Serialisedz Goats ...

# Anatomy of an Error

- GET http://server/Goaties
- 404 Not Found
  - ◆ Note: Not "file not found", but "resource not found"
- GET http://server/Goat?name=dotty
  - ◆ (throws an exception)
- 500 Internal Server Error
- Body contains SOAP serialisation of the error
- \*\*\* Just like SOAP RPC!!! \*\*\*

# Client Side

- GET requests to retrieve resources
- PUT requests to set resources
- POST to send data to a resource
- Apply meaning to HTTP response codes
- 200 is OK, body is response
- 500 is an error, body is error in SOAP encoding
- Note: Body does not need to be SOAP, e.g.  
<http://google.com/xml?q=axkit>

# Google Example

- GET <http://google.com/xml?q=axkit>

```
Date: Fri, 20 Jul 2001 19:41:12 GMT
Server: GWS/1.11
Connection: close
Content-Type: text/xml
Transfer-Encoding: chunked
```

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE GSP SYSTEM "google.dtd">
<GSP VER="3.1">
<TM>0.383937</TM><Q>axkit</Q>
<CAT><GN>gwd/Top/Computers/Programming/Languages/Perl/WWW</GN>
<FVN>Top/Computers/Programming/Languages/Perl/WWW</FVN></CAT>
<CAT><GN>gwd/Top/Computers/Software/Internet/Servers/WWW/Apache/Modules/Mod_Perl</GN>
<FVN>Top/Computers/Software/Internet/Servers/WWW/Apache/Modules/Mod_Perl</FVN></CAT>
<RES SN="1" EN="10">
<M>11800</M>
<FI/><NB><NU>/xml?q=axkit&site=search&safe=off&start=10&sa=N</NU>
</NB>

<R N="1"><U>http://axkit.org/</U><T>&lt;b&gt;AxKit&lt;/b&gt;</T><RK>10</RK>
<S> &lt;b&gt;...&lt;/b&gt; http://&lt;b&gt;axkit&lt;/b&gt;.org/ :: Welcome to &lt;b&gt;AxKit&lt;/b&gt;! &lt;b&gt;
These pages are copyright 2000 &lt;b&gt;AxKit&lt;/b&gt;.com Ltd.
</S><HAS><DI>
<CAT><GN>gwd/Top/Computers/Software/Internet/Servers/WWW/Apache/Modules/Mod_Perl</GN>
<FVN>Top/Computers/Software/Internet/Servers/WWW/Apache/Modules/Mod_Perl</FVN></CAT>
<DT>&lt;b&gt;Axkit&lt;/b&gt; </DT><DS>XML web publishing with Apache and mod_perl. </DS></DI><L/>
<C SZ="13k"/><RT/></HAS></R>
<R N="2" L="2"><U>http://axkit.org/faq.xml</U><T>&lt;b&gt;AxKit&lt;/b&gt; - FAQ</T><RK>3</RK>
<S>Please visit our sponsor. &lt;b&gt;...&lt;/b&gt; Frequently Asked Questions&lt;br&gt;
about &lt;b&gt;AxKit&lt;/b&gt;. Upon startup &lt;b&gt;...&lt;/b&gt;
</S><HAS><L/><C SZ="14k"/><RT/></HAS><HN>axkit.org</HN></R>
```

# Confused?

- This is confusing and complex
- But it's no more confusing than the SOAP encoded equivalent of the same data
- And this is much simpler to use...
- No SOAP libraries required - just an XML parser
- But! Some might argue this requires more collaboration between service implementor and service user
- ... however so does SOAP... You need to agree on API vs agree on data format

# Downside

- URLs are less structured than XML
- URLs have no validation syntax (unlike DTDs, W3C XML Schemas)
- XML can be sent to the server in a POST or PUT request
- but that is less transparent
- Again, use SOAP encoding, or XMLRPC encoding, but do not override the protocol semantics

# How to build SOAP and XML-RPC applications with mod\_perl

# SOAP::Lite makes this a joy!

- httpd.conf

```
<Location /soap_server>  
  SetHandler perl-script  
  PerlHandler Apache::SOAP  
  PerlSetVar dispatch_to "Module::Name"  
</Location>
```

# Simple huh?

- SOAP::Lite makes all this easy
- ... because it does all the complex work for you
- But with the 1990s RPC mechanism we can hide all the complex work also
- It just hasn't been done yet

# Conclusions

- Non-SOAP Web Services retain semantics of HTTP
- And are reasonably easy to develop
- But SOAP based Web Services are now easier to develop
- ... even though they destroy HTTP semantics
- So what do you care about more?