Rethinking the Java SOAP Stack

Steve Loughran
HP Laboratories
steve_loughran@hpl.hp.com

Edmund Smith
University of Edinburgh
esmith4@inf.ed.ac.uk

Hello, I'm Steve Loughran from HP Labs. My co-author, Ed Smith, and I are about to show why the Java SOAP stack is fundamentally the wrong way to expose SOAP to developers. I'm making this argument based on the experience of writing and using Web Services over a number of years, and also as a committer on the Axis SOAP stack.

What did SOAP promise?

1. Interoperability
2. Flexibility
3. Loose coupling
4. Long-haul communications

Page 2

These are what SOAP was meant to deliver. I'm going to argue that if these are what we want, then JAXRPC does not deliver.

-interoperability: language, object model neutral
-flexibility: dynamic extensibility of messages
-Loose coupling: avoid the brittleness of existing distributed object systems
-long haul, through firewall

What Java delivers: JAX-RPC

JAX-RPC 1.x

- Resembles Java Remote Method Invocation
- Focus on WSDL-from-code: contract last

JAX-RPC 2.0/JAX-WS

- Only a draft
- Retains focus on WSDL-from-code

This talk focuses on JAX-RPC 1.0

The primary API for soap in Java is JAXRPC: Java API for XML RPC. It uses Java RMI as its model.
In Java RMI, you make method calls of remote objects via local proxy classes, sending serialised Java objects over the wire.

We are going to look at JAXRPC1.x here, as it is the sole version that anyone has practical experience of.

People will argue that JAX-WS will correct all our criticisms, but that wont be obvious until it has been used in the field. All we can say is that a focus on generating the tyoe system and WSDL From Java source is still there; a goal of "ease-of-use" over "workable"

1. Interoperability

1. *Interoperability*
2. *Flexibility*
3. *Loose Coupling*
4. *Long-haul*

JAX-RPC focuses on WSDL generation from code

An implementation is not an interface:

- Code is not generally portable.
- It's impossible to keep the interface stable
- It's impossible to predict or control interoperability

Wasn't IDL invented to solve this?

In SOAP, interop is meant to be achieved by defining your types in XSD, your service in WSDL, both of which can be independent of an implementation platform.

JAXRPC has opted to make it easy for Java developers to send Java classes to other Java programs.

One way it does that is emphasise contract-last development. No need to write WSDL, no need to write XSD -just write Java code and JAXRPC will handle the rest. This is completely the wrong way to produce a vaguely-stable interface, which would seem to be a prerequisite to a stable production site.

It also completely cripples interop, because you are starting off with one platforms datatypes. At best, a different program written on a the same platform will be able to bind the same datatypes to the message elements, based on the generated WSDL. At worst: stuff gets lost.

2. Flexibility

1. *Interoperability*
2. *Flexibility*
3. *Loose Coupling*
4. *Long-haul*

JAX-RPC builds data classes from WSDL —callers see the classes, not the XML

- What if less data is sent than expected?
- What if more XML is sent?
- How can you handle extra attributes?
- How can the classes adapt to change?
- What navigation options are there?

JAXRPC hides from the XML. Instead of being given access to a document, you have to specify -at compile time- what you are going to receive.

This loses the ability to handle interesting incoming XML: documents you were not expecting.
We also think it removes your resilence to change. Can you find out of something came in with whitespace preserve set in an attribute? Or handle extra data of unknown type? Not easily.

Sometimes you can get at the XML data, but in a way that is best described as "potentially very inefficient". E.g Axis1.x gives you the DOM, but only by regenerating the tree into XML and parsing it again.

3. Loose Coupling

1. *Interoperability*
2. *Flexibility*
3. *Loose Coupling*
4. *Long-haul*

JAX-RPC : focus on 'Serializing' Java classes, operations on proxy classes.

This resembles “distributed objects”

- Assumes that the remote implementation matches the local one
- raises the expectation that SOAP is a distributed object system.

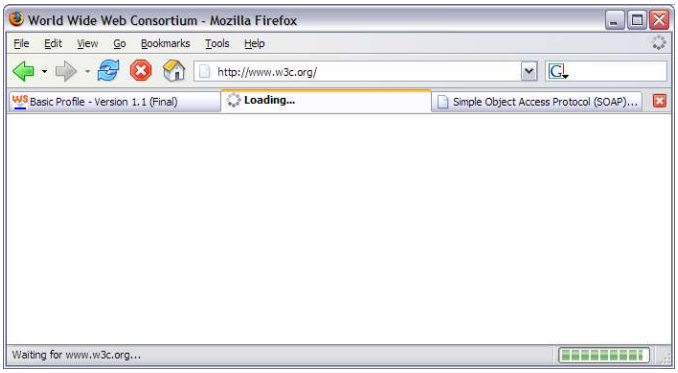
SOAP is not a distributed object system, and the limitations of that pattern are well-known.

Rediscovering distributed objects just with different transport doesn't change the things that made them not work originally.

4. Long-haul Communications

1. *Interoperability*
2. *Flexibility*
3. *Loose Coupling*
4. *Long-haul*

JAX-RPC says service calls are method calls, but lots of things can go wrong on a WAN...



Blocking RPC is the wrong paradigm

Notice the cancel button and progress indicator on thus (not responding) web page. Here the proxy server is live, so the connection has begun, it is just not finishing.

Implementation issues

- JAXRPC puts a hard split between developer-side “XML” and user-side “Java” code.
- Lots of problems related to WSDL-Java conversion
- The test kit (TCK) is secret

This handicaps open-source implementations

- End users struggle to become developers
- Developers have extra workload
- Testing a secret TCK is hard

These are some personal observations as an Axis developer. Too often the user mail lists are full of people trying serialize a complex Java graph and complaining that axis is failing with an obscure error message. These people suffer, but are not in the position to fix the code (ok, they could improve the error message). By having the framework hide the XML, they are kept hidden from understanding what is going on, and kept unable to fix it.

As a consequence, many of the OSS stack developers are J2EE/JMS vendors. Not end users.

Identified Problems

JAX-RPC

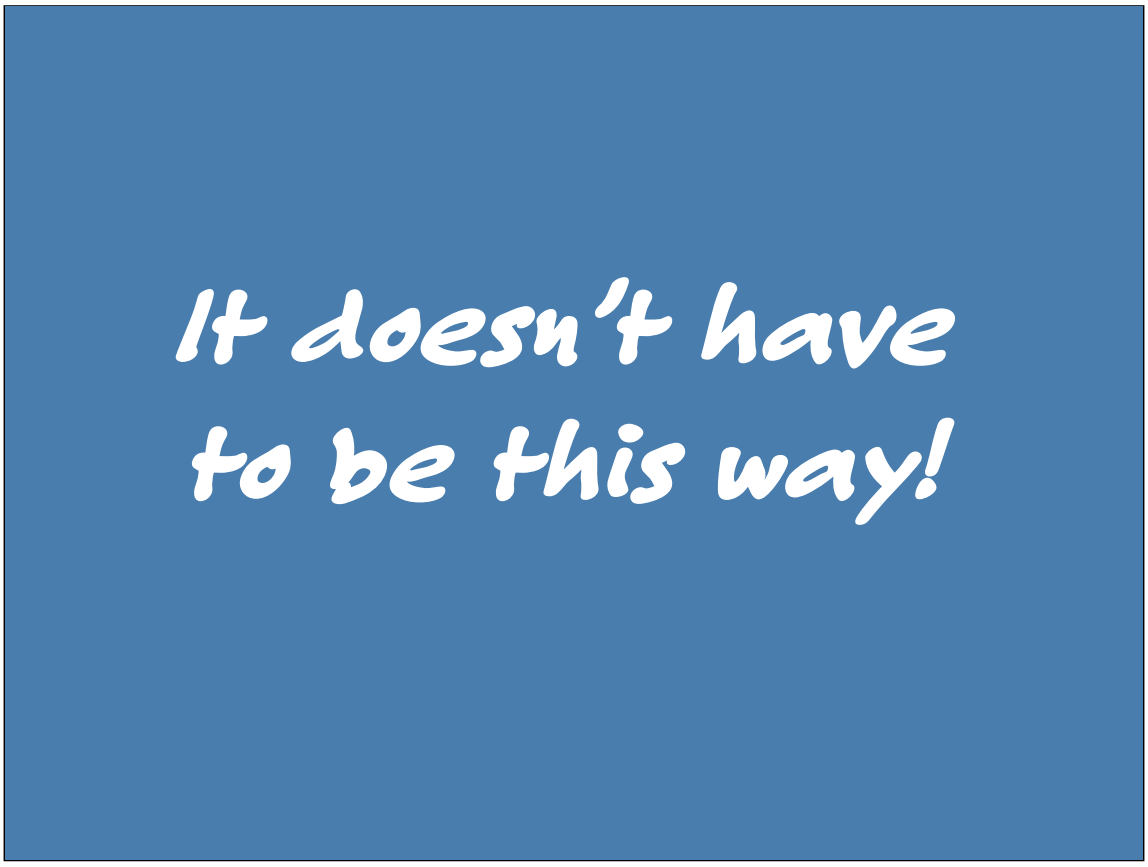
- Has crippled interoperability.
- Lacks the flexibility XML promised.
- Handles long haul connections badly.
- Is expensive to implement.

All due to hiding from the XML

By adopting the metaphor of Java RMI, JAXRPC hides from the XML.

As we have argued, this comes at a price. Here is what you lose.

Unless we are very much mistaken, these are the core reasons for SOAP.



We do not have to suffer like this. It is entirely an option of our own choosing.

Interlude: Himalayan vs. Alpine style

Slow teams take paid tourists up big mountains

Individuals travel light and fast

Page 11

This seems like a little diversion, but it will make sense at the end.

There are many ways to climb big mountains. In historic order, it goes

-classic alpine style: hemp ropes, hobnailed boots. Doing stuff (eiger nordwand) never done before. (read: the white spider, Harrer)

-Himalayan style: brute force ascent, military-grade operations. Originally to climb the impossible (here, kanchenjunga from darjeeling), now to take tourists up for many $. Read: Harrer, 7 years in Tibet; Krakenaur: into thin air

-Modern Alpine style: state of the art equipment. Travel fast and light, survival through speed. Here descending pelvoux from a night in the hut, courtesy of visa card bookings made over GSM phones on the way up. Returning to our car that will take us back to the UK in a 12 hour sustained drive.

-Super-Alpine: apply alpine techniques to the Himalayan, Andean peaks. Read: Simpson, Touching the void.

Alpine is about fast, light agile. There is risk, but if managed well you can be out and back before the weather gets bad. Himalayan style needs to be set up for storms, as the duration of a climb makes it inevitable.

SOAP Uses XML

Embrace the angle-brackets
—instead of hiding from them

- WSDL+XSD contracts come first.
- No more complicated, error-prone object mapping.
- No more self-defeating interface generation.

SOAP is about Messaging

Use a message/queue API

- No need to pretend there are remote “objects”
- Queuing a message isn't naturally blocking.

RPC doesn’t work over long haul links, we all know that. So lets go straight to a queued model, one in which you can pump stuff off.

We will still need to model a session, but this can be done as an explicit session context object, not implicitly in a proxy class.

Alpine: a proposed alternative

- Embrace XML & XPath
- Use the latest XML tools
- Drop rpc/encoded SOAP
- Queued/Asynchronous API
- WSDL-first

We are exploring an alternative. It is called 'alpine' based on alpine style mountaineering: fast, light, no support teams, use the best technology we have to hand.

Alpine “M32” prototype

- XOM based
- Factory chain to generate extended Xom nodes.
- More testing needed (where are the SOAP tests?)

```
for(Node n:
      body.xpath("m:GetLastTrade/m:symbol",
      context))
{
  String value = n.getValue();
  log(value);
}
```

A crude SOAP stack in three days

Page 15

If you are driving from Bristol to the Alps, via Eurotunnel, the M32 is the first road of any significance. You can go quite quickly down it, till something like the M4 and M25 get in the way.

XOM underneath. Context permits dynamic generation of helper classes.

One problem is WS-A; if you start to generate stuff then you have existing context-dependent content before you can determine the ultimate destination. This complicates my idea for endpoint-specific Xom node factories.

It only took three days to do. Think about it: a SOAP stack in under a week, with XPath access to the payload. How so fast? By using Xom and Jaxen, and not trying to generate Java classes from WSDL, WSDL from Java or pay any attention to SOAP section 5 encoding.

Now, the thing is not ready for use, we need a lot more testing. But the core of the stack is there.

We are not alone:

- Axis 2 core
  http://ws.apache.org/
- ActiveSOAP
  http://activesoap.codehaus.org/
- xFire
  http://xfire.codehaus.org/

XmlBeans + lightweight core runtimes; StAX parsers

All of these are 3rd generation SOAP stacks based on lessons from the field.

Axis2 is a complete rewrite of Axis1; essentially the third generation Apache SOAP stack (the first was an IBM donation).
I think it is a nice start, but with a lot of focus still on JAXRPC support, the inherent problems are still there.

One of the nice things is that it will be mainstream, so you can write handlers or client code that works direct with Axiom. But people who want the dubious benefits of WSDL-from Java can stick with Jaxb bindings.

AXIOM + XmlBeans; *XPath promised*
Engineering effort for JAX-WS still needed

ActiveSOA: light

Conclusions

- SOAP is all about XML messages
- Hiding the messages and the XML doesn't work
- The way forward is lightweight XML messaging stacks.
- If ease of use is the issue, then fix XSD and WSDL

Questions?

Addendum: JAX-WS

- Delegates O/X mapping to JAXB
- This moves the blame onto someone else
- Some asynchronous operations.
- @WebMethod, @WebService annotations
- Engineering costs still too high
- Test cases still secret

Doesn't JAX-WS fix this? Nope. JAXB does support a wider subset of XML than before, but it is still a subset, and it is still focused on mapping between custom Java classes and the XML message

What is worse, this will be built into Java 6.0. With a built in HTTP server for callbacks. This is scary.