SOAP Stack Interop

Steve Loughran
stevel@apache.org

A Quick presentation to the Global Grid Forum on interop testing.

SOAPBuilders

* Evolution of the initial SOAP interop tests (rounds 1-5)
* Public endpoints for remote testing (listed on whitemesa.com)
* Simple SOAP1.0 and rpc/enc SOAP
  - # round-tripping datatypes
  - # headers and mustUnderstand
* Mailing list on Yahoo! groups
* Authority to resolve ambiguities

Key is the last point: if there were problems, the developers could reach consensus and fix it.

SOAPBuilders limitations

- Too simple
  e.g. stateless tests didn't catch Axis1.1 mustUnderstand bug
- Endpoints don't explore failure modes of stacks/network
- Needs to be online to work
- frozen at SOAP1.0 + rpc/enc
- gradual stagnation & loss of test nodes

They may seem too simple now, but at the time they were fantastic. Everyone who exchanges something as complex as an integer array between two stacks owes the operations success to the SOAPBuilder tests

WS-I

+ laid down good/bad WSDL rules
+ formalised HTTP response rules
+ Wrote tool to validate WSDL
+ Good press

- punted on valid subset of XSD (xsd:dateTime, ulong ...)
- pushed DIME for attachments
- stopped SOAPBuilders from resolving ambiguities

WSI marked interop being perceived as important enough to put bigger groups on it. But suddenly it got all political. Nobody want to change their stack's impl (esp shipping things like .NET1.0, so it became more of a place to push for your interpretation. Attachments DIME vs SwA were the big example. As for which subset of XSD to support, they avoided the problem as too controversial.

One of the saddest things is that people think that WS-I have solved interop.

PlugFests/F2F sessions

* First SOAPBuilders F2F
* Later: single-vendor PlugFests
* Good for contacts across teams
* Expensive to attend
  (barrier to OSS participation)
* Needs a good extreme-programming
  development setup
  (LAN, caffeine, time, no jet-lag)
* No good for regression testing

Face 2 Face sessions can do the following things

1-demonstrate interop

2-fix problems

3-lead to finger pointing and blame assignment

1. Not tests: press events where the outcome should be known in advance.

2. this needs a test suite to run against each other, developer participation (ability to fix code during the session), plus authority to resolve ambiguities in the specs (it's a good time to discuss problems)

3. This is what you end up with without a formal compliance suite and major differences in interpretation;

Other interop sources

* Trace messages to SOAPBuilder EPRs
* Trace messages from SOAPBuilder EPRs
* Troublesome WSDL from bugreps (but not user-error WSDL)
* tcpmon traces in bugreps
* Known problems in old stacks

If SOAPBuilders is inactive and WS-I punted on the hard problems, where does that leave us? Informal testing using bugreps from users, packet logs and WSDL from troublespots.

Message traces can be replayed and work for stateless endpoints; good for offline and repetitive work.

The other area is experience. We known what XML formats .NET 1.0 doesn't like; Axis has switches to turn on extra compatibility there. But without adequate regression testing, its easy for trouble to creep back in...

CDDLM Interoperability

* Reference implementations
  # HP, NEC, Softricity, UFCG
* CDL unit test suite working
* Functional Tests for
  # deployment API
  # component model
* Plan to bring up public EPRs
  Cross-test clients and servers
* May resort to sharing VMWare images
* Concern: WSRF/WSA interop

This is what we are going to do in the CDDLM working group. We have a well defined test suite for the language (sourceforge hosted), and test plans for the SOAP API and component model behind it. If every team implements test clients for the latter, they should be able to point them at any implementation to see if they work or not.

Thus the functional tests of the server become the interop tests of the system.

All we need are the other endpoints. We may host them publicly with some authentication layer, or resort to sharing VMware images that groups can host locally and run against locally. This is cute as it gives better debugging and offers online use.