

the wondrous curse  
of interoperability

*what interop is,  
and how to cope*



steve loughran  
hp laboratories  
slo@hplb.hpl.hp.com  
january 2003

presentation for padnug.org; <http://www.padnug.org/padnug/WebServicesSIG.aspx>

Interop: What were they thinking of?

**why is S/W dev like mountaineering?**

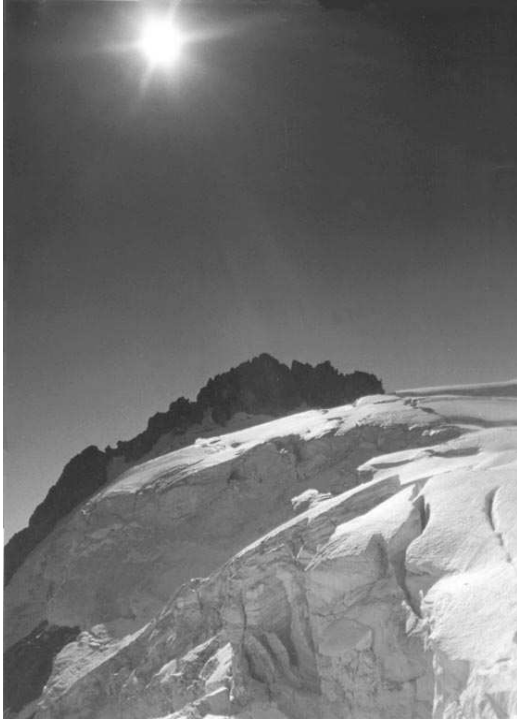


**all looks fine from 30,000'**

**vision**

A bit of a distraction before we get into the detail, with the definitive mountaineering metaphor slideshow.

Most uses of mountaineering 'many people work on an Everest expedition, only two summit' are done by HR and managers who don't know the details of modern Alpinism. But I do, so here is a comparison accompanied by photos mostly from Ranier, some from Mt Hood and one from the Alps (Ecrins Massif).



**up close  
it's terrifying**

**reality**

**and it is too late to turn back**



**commitment!**

The only way out is through.

## how do you survive?



- equipment
- process
- practice
- planning
- fear management

There is one big difference: get it wrong in the mountains and you are dead, survivors make money selling airport books about near-death-experiences, for reading by passengers who don't find fly-by-wire planes programmed by us developers scary enough.

Get it wrong in most software projects and you are still alive, just with a different employer. And those who survive in the same company end up vilified and fielding support calls.

## equipment



- good CPU, lots of RAM, HDD, WLAN...
- IntelliJ IDEA
- WSDL editor
- Ant+JUnit/HttpUnit/Cactus
- cellphone & caffeine
  
- *Continuous Integration*
- *Automated Deployment*

Note that a cellphone actually goes onto the list of both types of entertainment.

## process, planning, preparation



- alpine starts
- speed
- maps, milestones, checkpoints
- practice: crevasse rescue, first aid, navigation in zero-visibility, snow-caves...
- *heaviest person goes first*

-Set off early so that if anything goes pear shaped you have most of the day

-know what your objectives are, have maps, milestones, don't be afraid to replan

-don't climb Rainier (or even Mt hood) without some practice under your belt. And that means more than practice of nice weather days out; you need to practice things going wrong: crevasse rescue, bad weather days, dehydration days.

-send the person most likely to fall into a crevasse first. Its better for everyone if things go wrong ahead rather than behind or in the middle of the team. This doesn't mean they are expendable, just more likely to find problems.

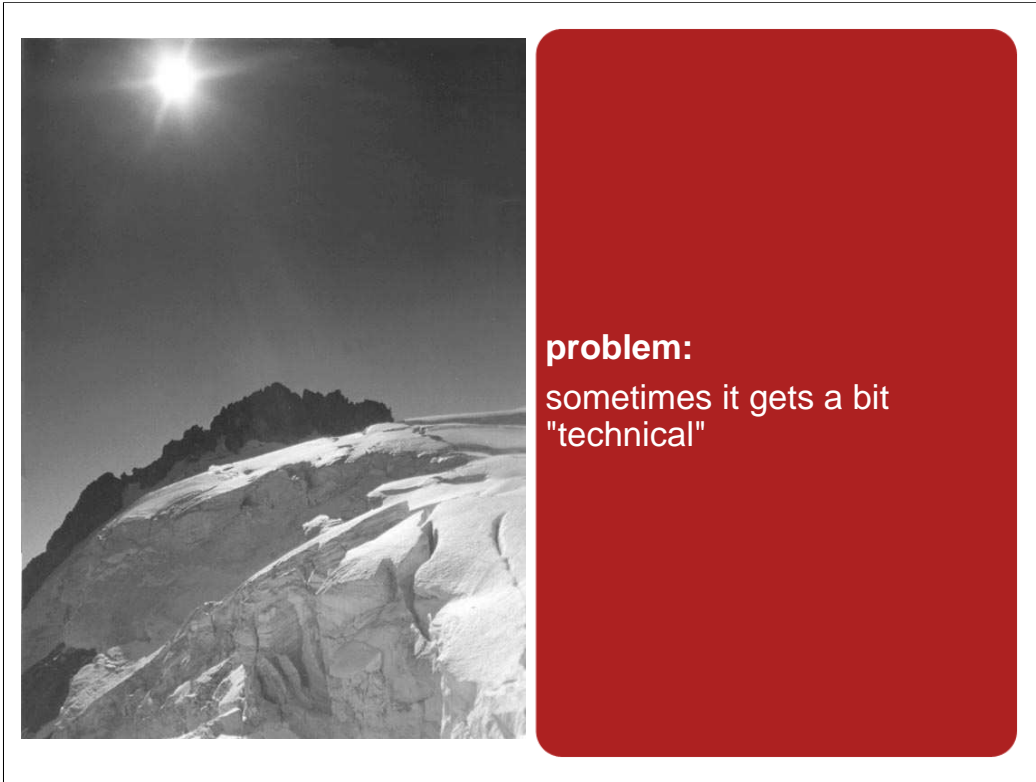
Which brings us to web services, interop and the testing, thereof.

## the Web Services vision



XML over HTTP lets  
everything talk to everything  
else

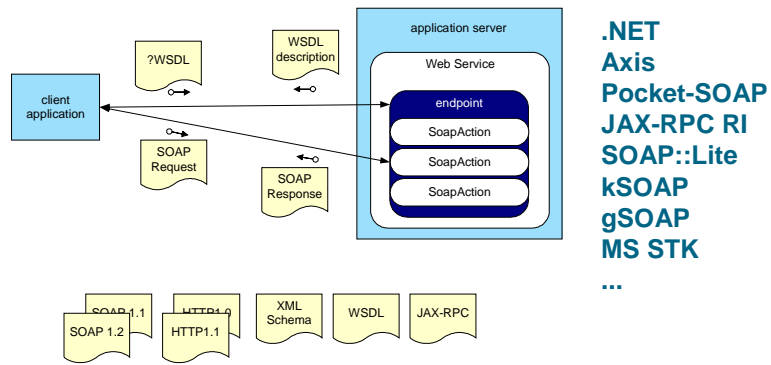
the 30,000' view



Up close.

Good days ::= days when the network is good, person writing the service knew what they were doing, no attempt to share binaries, manage sessions, etc.

## Web Services: quick summary



**.NET**  
**Axis**  
**Pocket-SOAP**  
**JAX-RPC RI**  
**SOAP::Lite**  
**kSOAP**  
**gSOAP**  
**MS STK**  
...

many specs

many languages + many implementations =  
interop problems

if you are ignorant of SOAP, here is the one page reminder. note that WSDL is optional, SOAP messages have envelopes, headers and bodies, and there are two ways of encoding stuff: RPC/Encoded and DOC/literal

**key problem:  
naïve and unrealistic expectations**

- you can't send things like `java.io.File` over the wire.
- you shouldn't send serializable objects like `javax.swing.JPanel` over the wire.
- you can't do transactions (yet)
- there is no good authentication system (yet? ever?)

***SOAP is not Corba, RMI, RMI-IIOP, .NET remoting,  
DCOM(+), or any other classic  
remote object invocation mechanism***



## enumerations

### C# bitfield enums

```
[Flags]
public enum Rating : int {
    unknown=0, fun=1, scary=2, hard=4, painful=8
}

[WebMethod]
public Rating HowWasIt(String route) {
    return (Rating)random.Next((int)Rating.painful*2);
}
```

are marshalled as an XML Schema list

```
<>fun scary hard</>
```

that many things (Axis, Castor) cannot handle yet

We can't even do enums reliably if the far end cant handle the XSD space-separated-list-in-an-XML-element form of lists.

This isnt a fundamental issue, just an engineering one.



## exceptions

- how do you send exceptions down the wire
- how does the far end handle them?
- what about extra data in a SoapFault?
- what about languages with no notion of exceptions?

JAX-RPC tries to send any Exception over the wire.

*axis: throw AxisFault with info in the details*

See the JAX-RPC spec for Java exception handling. Basically JAX-RPC is trying to marshall Java exceptions in a way that they can be recreated. This is because Java methods need to state the exceptions they throw, the exception set can be included in the WSDL, client versions written, etc. But how does this work across languages? How does it work across implementations of JAX-RPC?

## framework objects and collections

- ArrayList
- DataSet
- java.util.HashTable

*"the hashtable problem"*

The problem here is that just because you can send stuff over the wire, doesn't mean the far end can handle it. Sometimes they cant, sometimes they can, but not by creating the obvious bindings you'd expect.

## sessions

```
[WebMethod(
  EnableSession=true
)]

stub.
  setMaintainSession(true)
```

- 1. cookies
- 2. SoapHeaders
- 3. dynamic endpoints
- 4. sessionID in API

-W3C push for SOAP GET is incompatible with SOAP Headers

-WSDL for dynamic endpoints?

-we need a SoapHeader standard

The flaws with cookies are well known, with the extra caveat that whoever maintains `java.net.URLConnection` has never read the bit of the HTTP spec that covers them. For that, the pains with different paths to the service serving up cookies differently, makes them something to avoid. Also not all clients support them, and it aint always by default. e.g Apache Axis, you must enable it client side.

SoapHeaders are the cleaner and more elegant way. But it does require

Dynamic endpoints: returning new URLs after a login. I like this model, but it is hard to work with, too 'RESTful'; but maps nicely to objects. .NET remoting gives you this.

API: include session ID in messages. Ugly, at odds with AOP, but works well, works consistently, and is easily tested. Guess which I use.

see W3c: <http://www.w3.org/TR/soap12-part0/> on the subject of SOAP and GET

"What were they thinking of?"

## binaries:

- SOAP with Attachments (SwA) is in JAX-RPC
- DIME is in WS-toolkit for .NET
- Axis: SwA and DIME (but not WSDL)
- base-64 is the fallback

***no attachments in  
Basic .NET 1.0***

|           | SwA | DIME |
|-----------|-----|------|
| Axis      |     |      |
| Sun JWSDP |     |      |
| .NET 1.0  |     |      |
| .NET WSE  |     |      |

You could maybe point the blame at this for XML; there is no way to reliably include arbitrary binary code in an XML file. If only they'd had a length+data way of encoding binary, life would be simple.

The other blame is at Team .NET for leaving out SwA because DIME was better. Maybe it is, but they didn't ship with DIME either. So leaving out SwA was either a schedule driven play or an attempt to stop SwA becoming ubiquitous.

For now: `xs:base64binary`, `xs:hexBinary`. But try sending 2GB of print data in base 64 format. Actually, try sending 2GB of data in any SOAP attachment format.

## other surprises

- RPC/ENC versus DOC/LIT
- floating point: decimal places, infinity
- DateTime & timezones
- object references
- ragged arrays
- Sun's JAX-RPC reference implementation bails out if it can't handle *all* methods at an endpoint.

This is just other little showstoppers that will hurt you if you are not careful.

## why did we get into this mess?

- Web Services grew by accretion, not planning.
- nobody wants to scare developers off.
- coding 'IDL' is considered harder than exporting methods.
- closed world systems (DCOM, RMI) don't have interop issues.

From a CS perspective, SOAP+XSD+WSDL is ugly, only their mothers would love them, and even then they'd hope the next kid will look better.

But ugly is irrelevant.

(nb, notice that Corba can have interop; but when used as a C++ to C++ system it is generally ok, because everything is well specified in the very large CORBA spec and its mostly one language everywhere)

***there is no point writing a  
Web Service unless you  
want interoperability***

Why are you writing a web service unless you want people to call you? And can you dictate what language they will be using, or do you want all customers, regardless of what they call you with.

But lets look at what we have:

- +finding: UDDI, WSIF, SLAv2)
- +binding: WSDL
- +invocation: SOAP
- +data representation: XSD

The finding and binding features of SOAP, indeed WSDL on its own, are what makes SOAP so interesting today. The fact that interop is a big issue is a detail, compared to the profound notion that you can browse to a web page, download some description of endpoints and suddenly start posting structured data to it.



WS-I: I havent seen an enum or session working group here, that is the kind of implementation detail that architects don't bother with. Plus WS-I is very political.

SoapBuilders: This is where interop does seem to get solved. But we could do with Collections, exceptions, , char, enum and a big fat 'guide to interoperability' paper

Axis: well, you are axis devs if you are users; work with us on the docs, exceptions, better interop tests, etc.

W3C: add xml headers to HTTP GET s.t. SOAP GET works; char on Schema.

Sun: give us unsigned data, add HashTable &c to JAX-RPC, fix the exceptions model.

MS: Give us SwA on .NET, document DataSet, add HashTable &c. Oh, and Zip support too; you can't send or receive zipped data right now.

You: fix your own app to not have interop issues.

## stay out of trouble



1. understand the nature of the problem
2. avoid the big issues
3. drive off the XSD & WSDL
4. test for interop

Question: what are the hazards in this picture, taken below the summit of Mt Hood?

Most people think of the usual ice and rock problems, but a big safety issue here is that bare patch of rock to the centre-right of the scene. Anything at that angle at that altitude should normally be covered in snow –but it isn't. This could be because of an avalanche, but there is no crown-wall, and you'd expect to see the base layer of snow unless the low-friction base was actually frozen turf.

The reason the rocks are bare is because we are climbing a volcano, this is the crater, and they are being heated by gases coming up from below. The atmosphere near the rocks is too toxic to be breathable. If you have done your research, you know this. If you haven't, and you go over to take a look, you are in trouble.



**avoid the big issues**

- no unsigned datatypes, char, enums
- don't send fancy objects
- stick to your own simple 'structs' and arrays
- support multiple binary upload mechanisms
- worry about sessions

This is all very dumbed down, but it is pragmatic. Arrays work. Simple objects with attributes and get/set methods work.

If you keep everything simple and avoid well known trouble spots, all should work

## **start with the XML Schema**

- the schema defines your serialization format
- this should be independent of use (files, SOAP, REST)
- but you *must* use the subset of that interoperates over SOAP.

## **drive off the WSDL**

- create a WSDL file using the Schema
- create server stubs from the WSDL
- just as if you were writing an IDL specification
- only uglier

This is a hard one. I hate looking at WSDL, it is just so convoluted. Same for XSD, though that one I can handle, slowly, even if something like XML Spy or VS.NET is needed to make it easier. Omniopera and CapeClear are alternatives.

One q: Why do you need to buy a new tool to edit WSDL files? Was there ever such a thing as an IDL editor other than a text editor? And so somehow WSDL is better than IDL? I don't understand that.



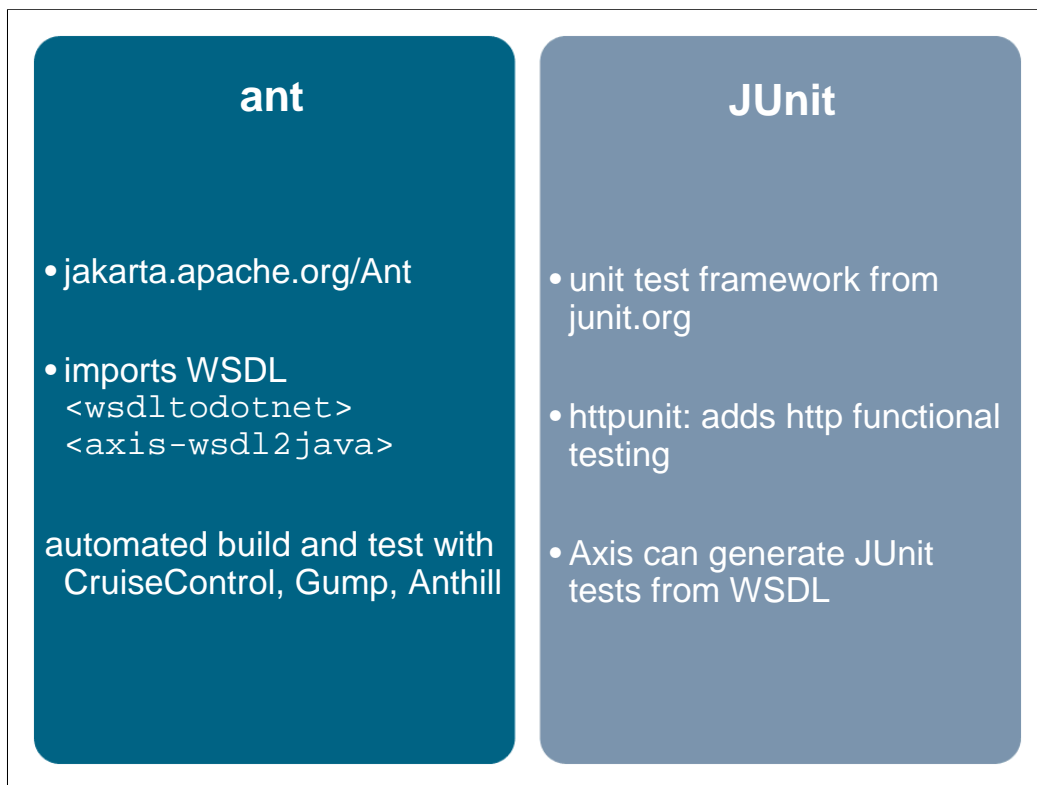
## test for interop

- pick the likely 'other' platforms
- generate client bindings from the WSDL
- write test cases to call the service.
- compile, run, fix
- repeat

The way to test for interop is to call your service from different toolkits and languages, which means replicating the usual client coding process.

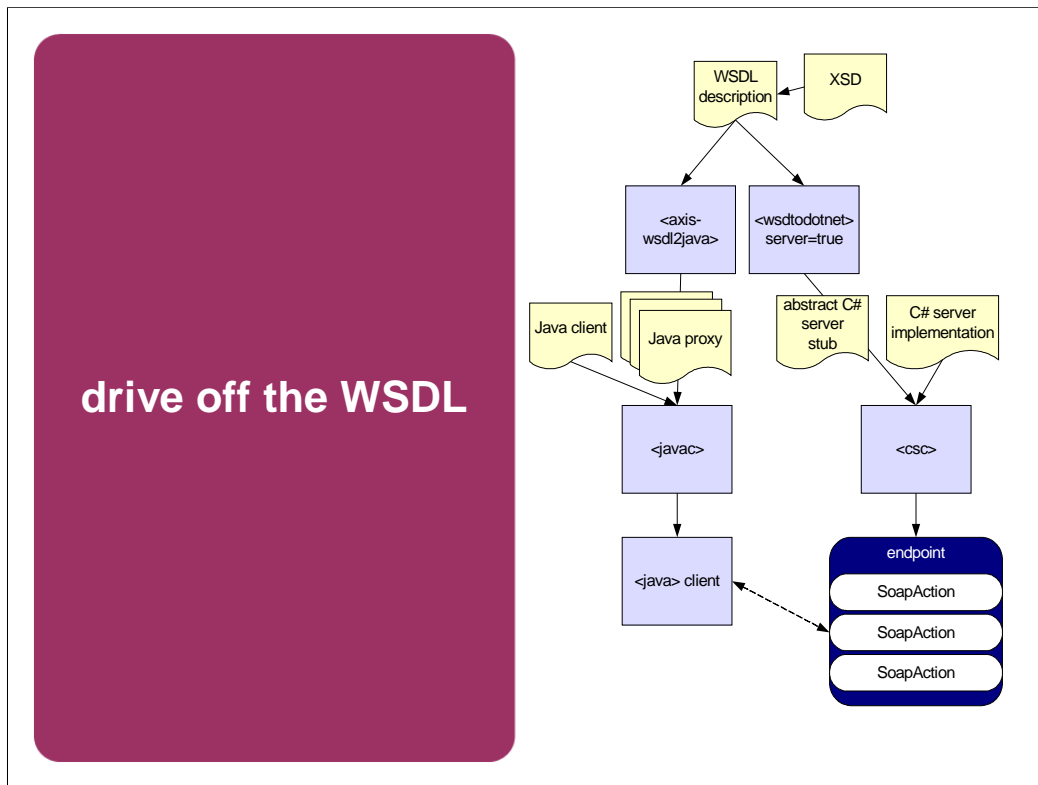
To do this in a test case environment, you need to automate the process.

I'm going to look at automating the Java side test process

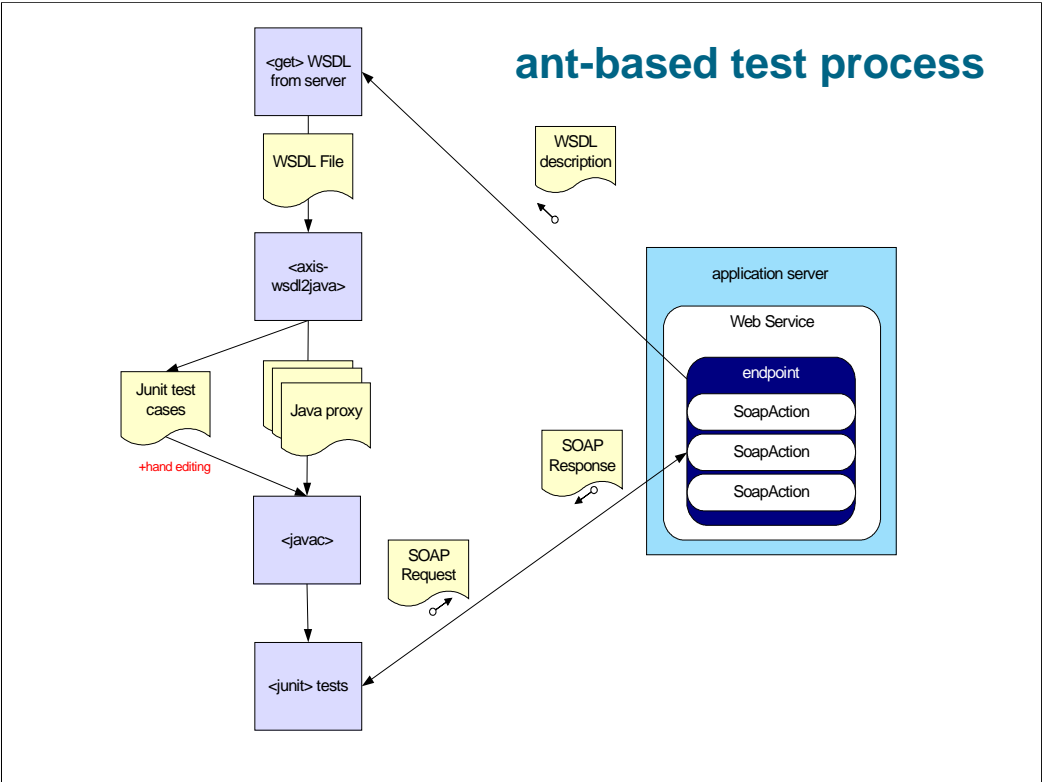


These two tools are the secret of why Java projects are staying in control; everyone is using the same test framework, and the same build tools, tools that understand testing is integral. Together nothing gets deployed that doesn't pass the tests.

As well as using them for pure Java dev, we can use them for .NET interop testing.



I'm not going to demo this, just show that you can work with Schema and WSDL as if it were an IDL spec of a service, that both the service and the clients adhere to.



this is our test workflow, what we will do in Ant.

## result: you know what to avoid

Unit Test Results  
Designed for use with [JUnit](#) and [Ant](#).

Class EndpointTestCase

| Name                             | Tests | Errors | Failures | Time(s) |
|----------------------------------|-------|--------|----------|---------|
| <a href="#">EndpointTestCase</a> | 14    | 0      | 5        | 7.109   |


Tests

| Name                                 | Status  | Type   | Time(s) |
|--------------------------------------|---------|--|---------|
| test1EndpointSoapVectorize           | Success |  | 4.109   |
| test2EndpointSoapBoxify              | Success |  | 0.141   |
| test3EndpointSoapIncrement           | Success |  | 0.187   |
| test4EndpointSoapGetSessionCallCount | Failure | expected:<2> but was:<1><br>junit.framework.AssertionFailedError: expected:<2><br>but was:<1><br>at<br>EndpointTestCase.test4EndpointSoapGetSessionCallCount<br>(EndpointTestCase.java:98)<br>at sun.reflect.NativeMethodAccessorImpl.invoke0<br>(Native Method)<br>at sun.reflect.NativeMethodAccessorImpl.invoke<br>(NativeMethodAccessorImpl.java:39)<br>at sun.reflect.DelegatingMethodAccessorImpl.invoke<br>(DelegatingMethodAccessorImpl.java:25) | 0.156   |
| test5EndpointSoapSetSessionMessage   | Failure | expected:<1> but was:<null>  | 0.157   |

This is the output: a list of how many tests ran, what the failures were.

**summary**

**interop issues:**  
find them  
before you fall into  
them

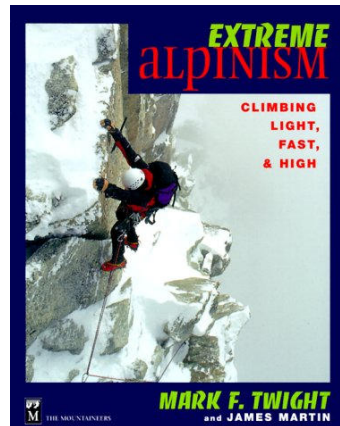
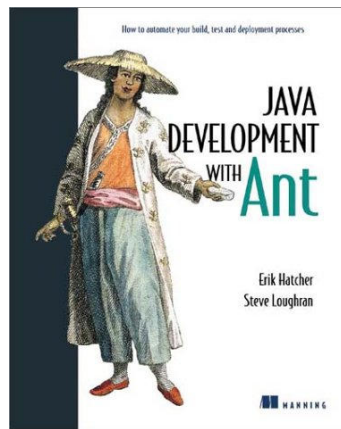


And that's it.

We've looked at areas of trouble, ranging from language issues (uint), framework differences and just bits of SOAP that aren't nailed down yet. If your app strays into the trouble areas, you are going to get hurt.

The trick is not to stay off the mountain, but to go through the crevasses carefully. Probe the snowbridges, know where trouble lies and plan your route, and just be careful.

## read these



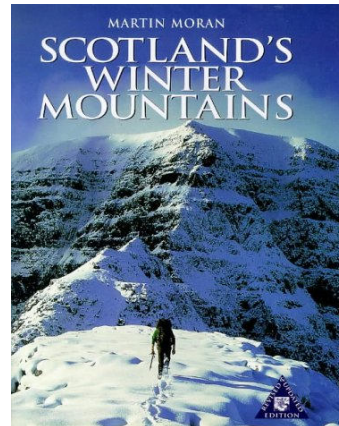
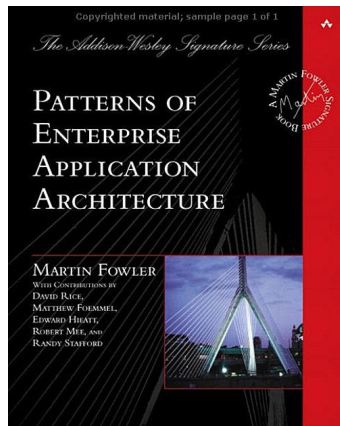
<http://iseran.com/Steve/papers/interop>

copyright © 2002-2003 Hewlett-Packard Company

Java Development with Ant, by me and Erik, shows how to use ant, how to interop test Axis servers with .net clients, set up continuous integration servers and many other useful things.

also, <http://iseran.com/Steve/papers.html>

and maybe these



Patterns of enterprise app arch by martin fowler: good enterprise patterns, including a critique of EJB  
flaws: maybe too much emphasis on data binding, not the other aspects of big server side apps

Scotland's Winter Mountains, Martin Moran. Best book on mountaineering ever. Full of facts, figures and anecdotes.

flaws: second edition still covers the nutritional values of fried breakfasts and hangovers, but omits the religious debate between bars of chocolate and honey slices. Also by virtue of a focus on Scotland, has nothing on glacier work.