Using REST and WS-* for SOA

Dr Mark Little
Technical Development Manager, Red Hat

Background

- **Research into fault-tolerant distributed systems since 1986**
  - Arjuna, Argus, Isis/Horus, Emerald, Xerox, …
  - DCE, DCOM, CORBA, JavaRMI, HTTP, Web Services, …
- **Active in OMG, OASIS, W3C, JCP, GGF, OSGi and others**
- **Involved with Web Services since 1999**
  - Co-author of a number of WS-* specifications and standards
- **Involved with REST/HTTP since at least 2000**
  - W3Objects
- **IBM, Arjuna Solutions, Bluestone, HP, Arjuna Technologies, JBoss, Red Hat**
- **Newcastle University**

Overview

- **SOA in a nutshell**
- **WS-* and REST**
- **Where should you start?**
  - Don’t panic!
- **One size does not fit all**
  - Don’t believe everything you read
- **Bridging the divide**
  - Can REST be used with WS-*?

The debate

- **Actually more about WS-* and RESTful HTTP (REST/HTTP)**
  - REST is a valid approach to SOA
- **No REST bashing**
  - REST is a valid architectural approach
  - HTTP is one way of implementing it
- **No WS-* bashing**
  - And specifically no WSDL bashing please
- **These types of debate have raged throughout history**
  - BetaMax vs VHS
  - Blu-Ray vs HD-DVD
- **Hybrid systems are the norm**
  - Very few places can afford to rip-n-replace

SOA in a nutshell

- SOA is an architectural style to achieve *loose coupling*
  - **A service is a unit of work done by a service provider to achieve desired end results for a consumer**
- SOA is deliberately not prescriptive about what happens behind service endpoints
  - **We are only concerned with the transfer of structured data between parties**
- SOA turns business functions into services that can be reused and accessed through standard interfaces.
  - **Should be accessible through different applications over a variety of channels**

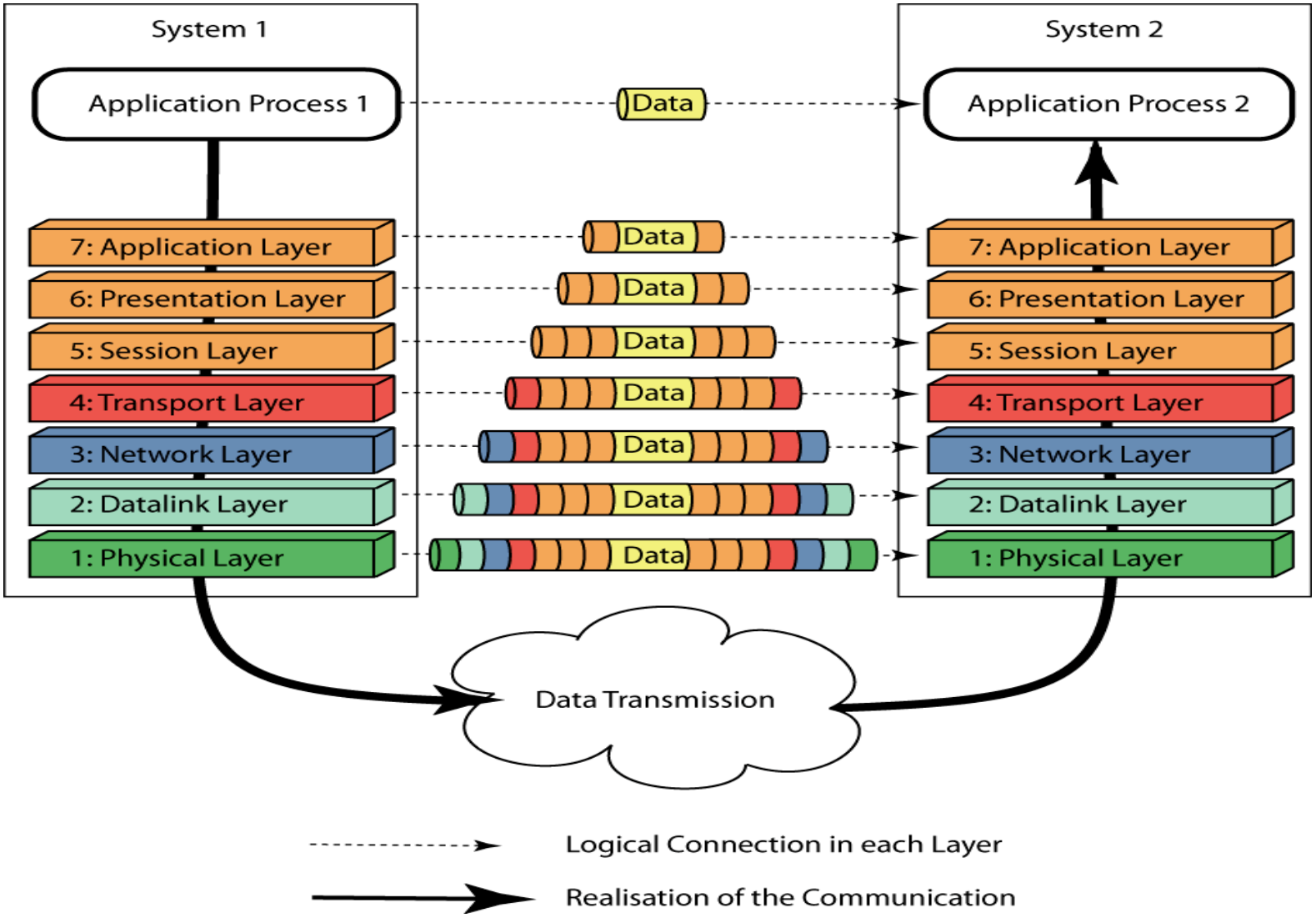
Achieving loose coupling

- **SOA employs two architectural constraints**
  - A small set of simple and ubiquitous interfaces to all participating software agents. Only generic semantics are encoded at the interfaces. The interfaces should be universally available for all providers and consumers
  - Descriptive messages constrained by an extensible schema delivered through the interfaces. No, or only minimal, system behavior is prescribed by messages. A schema limits the vocabulary and structure of messages. An extensible schema allows new versions of services to be introduced without breaking existing services

Implementing SOA

- **Many different possible approaches to SOA**
  - CORBA
  - J(2)EE
    - JMS
- **Two most popular approaches are**
  - WS-*
  - REST
    - Actually an architecture in its own right (not tied to HTTP)
    - Much more solidly defined than SOA
- **Which is the right approach?**
  - Why should there be only one?

The protocol stack



Distributed Systems

- **Same fundamental laws**
  - Develop “entity”
    - Define the UOW it supports
  - Search for “entity”
    - Agree “entity” offers the desired capability or UOW
  - Request “entity” to perform UOW
    - Create a network-transferable message
    - Send the message
  - Maybe try to make the remote interaction appear local
  - Maybe do some “enterprise” work as well
    - Security, transactions, replication etc.
      - Contextualization of messages

The most popular (by demand)

- Client and server technologies based on RPC
  - **Hide distribution**
  - **Make remote service invocation look the same as local component invocation**
  - **In use since the 1970's**
- Unfortunately this leads to *tightly coupled* applications
  - **Changes to the IDL require re-generation of stubs**
    - **And dissemination of new code**
    - **Or errors will occur during interactions**
  - **Such applications can be brittle**
    - Hard to control the infrastructure as needed
    - No quiescent period

Degrees of coupling

- **At the one extreme**
  - Defining specific service interfaces, akin to IDL
    - Easier to reason about the service
    - Limits the amount of freedom in changing the implementation
- **At the other extreme**
  - Single operation (e.g., doWork)
    - More flexibility is changing the implementation
      - Well, almost …
    - More difficult to determine service functionality a priori
      - Need more service metadata
- **There are degrees of coupling and you should choose the level that is right for you**
  - Not specific to distributed systems

Distributed Systems 101

- **The same requirements are present throughout the stack**
  - Split differently between the infrastructure and the “application”
- **Uniform interface allows for generic infrastructural support**
  - Caching, extremely loose coupling
  - Can push more requirements on to the “developer”
  - Requires more from external contract meta-data
- **Specific interface allows for more limited generic support**
  - Targeted caching, application semantics
  - Impacts less on the “developer” but may cost in terms of flexibility

So what do we need for SOA?

- **An architectural approach that supports**
  - Loose coupling
  - Enterprise capabilities
    - Security, reliable messaging, fault tolerance
  - Different invocation mechanisms
    - UDP, TCP, HTTP, JMS, IIOP
  - Integration with back-end “legacy” systems
  - Standards based
- **Things that would be nice (not architecture)**
  - Easy to use and administer
  - Good tooling

REST

- **Defined by Roy Fielding in his PhD**
  - One of a range of approaches
- **Core set of architectural principles**
  - Identify all resources/entities
  - Link resource together
    - Hypermedia as the engine of application state
  - Use standard methods for interacting with resources
  - Multiple resource representation
  - Stateless communication
- **Not tied to HTTP**

REST/HTTP in a nutshell

- **RESTful HTTP**
  - Just because you use HTTP does not mean you are using REST
- **Original HTTP specification talked about adding new commands**
  - GET only in 0.9
  - GET, HEAD, POST, extension-method in 1.0
  - Now we have up to 8 different verbs
    - Changes to “interface” occur but users aren’t affected
- **Many good distributed systems characteristics**
  - OPTIONS
  - Try-before-you-buy
  - Caching
  - Scalability
- **HTTP is NOT a transport**

Human driven?

- **Often stated that REST is only suitable for hypertext**
  - "Because that's the way the majority of the Web works"
- **Weak argument**
  - It's still a distributed system, just based on resources
  - OK, humans fill in the "gaps" in contract definition
  - Extra infrastructure support could help
    - Anyone remember URN name servers (1994/1995)?
    - But now we have Google!
- **Not enough "application" standards**
  - Good point
- **Lack of good tooling**
  - Well … JAX-RS, WCF, …

Standards

- **The Web is a series of standards**
  - URIs
  - HTTP
  - HTML
- **Universal adoption has to count for something!**
- **REST/HTTP is ubiquitous**
  - Communication interoperability, which is a good start
    - Also why WS-* standardized on HTTP
  - But application semantic interoperability is not there (yet)
    - Take a minimum of 5 years to do

Enterprise ready?

- **D'Oh!**
  - Take a look at the world!
- **Enterprise capabilities**
  - Reliable messaging
  - Some fault tolerance
    - No transactions
    - Workflow
    - 404 rules!
    - Stale links
- **Components are there, just not necessarily used**
- **But … it's not that easy!**
  - Human interaction style can sometimes confuse

Transactions

- **REST transactions by HP in 2000**
  - Yes, customers want to coordinate business interactions across multiple sites
    - Even if only atomically
  - Machine driven
- **http://<machine>/transaction-coordinator**
  Performing a GET returns a list of all transactions know to the coordinator (active and recovery)
- **http://<machine>/transaction-coordinator/begin?<ClientID>**
  Performing a PUT will start a new transaction and return a URL /transaction-coordinator/<id>

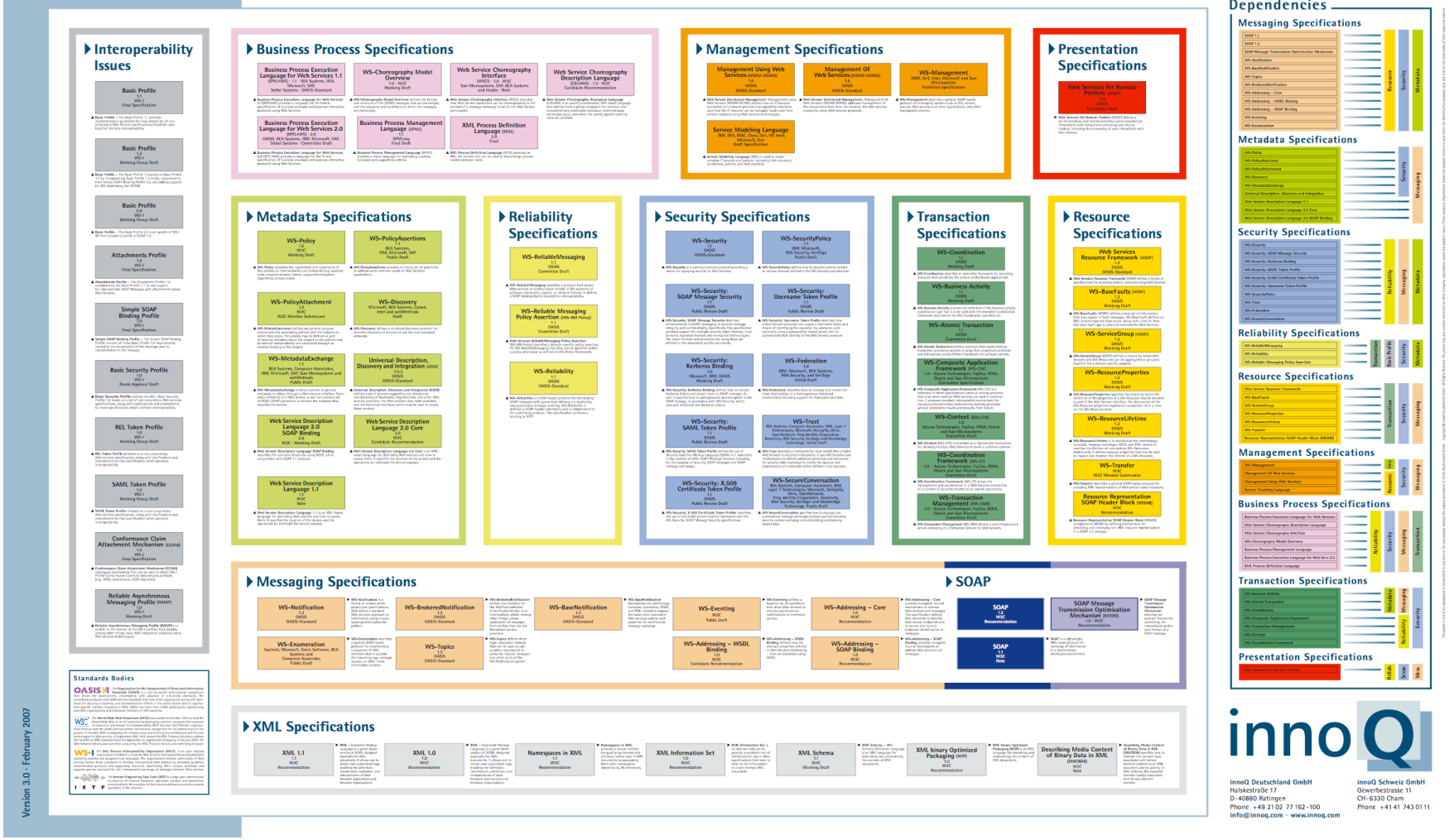
What about Web Services?

- **Popular integration approach**
  - XML
  - HTTP
    - Other transport bindings are possible
- **Developed with machine-to-machine interactions in mind**
- **Not specific to SOA**
  - Web Services began life as CORBA-over-HTTP
  - XML-RPC
  - WS-RF and WS-Addressing
- **Web Services+SOA gives benefits**
  - Loose coupling
  - Interoperability
  - Enterprise capabilities, e.g., security and transactions

Enterprise realities

- **Customers want interoperability of heterogeneous systems**
- **They want guaranteed delivery of messages**
  - Even in the presence of failures such as network partitions
- **They want transactions**
  - Not just ACID transactions!
- **They need audit trails**
  - Sarbanes-Oxley anyone?
- **They need bullet-proof security**
  - Sarbanes-Oxley
- **They need machine-readable contracts with SLAs**

WS-* Architecture



Fault tolerance

- **WS-* has driven protocol interoperability**
  - More so than CORBA, DCE, Java
- **Native data and protocol bridging**
- **Clearly defined semantics for transactions, security, reliable messaging**
  - Not specific to HTTP

WS-* backlash?

- **“It’s too complex”**
  - Complicated problems often require complex solutions
- **“It’s using HTTP as a transport!”**
  - Get over it! Mistakes happen.
- **“It doesn’t offer anything better than REST/HTTP”**
  - Short-sightedness works in both directions
- **“It doesn’t leverage the Web”**
  - Valid point for Web deployments
  - Not so valid for every other type of deployment

SOA: REST or WS-*?

- **SOAP/HTTP**
  - WS-* vendors have spent a lot of time ensuring it integrates with legacy systems
    - How many people remember that WS-* is supposed to also be about Internet scale computing?
  - That's important for out-of-the-box and interoperable deployments
  - But REST as a general architectural approach has merits
- **REST/HTTP has "simplicity" and (relative) ease of use**
  - No vendor-lockin at the infrastructure level
  - Is precisely for Internet scale computing as well
  - Remember that a lot of cool Web applications aren't RESTful

What can REST learn from WS-*?

- **Uniform interface isn't enough for complex application requirements**
  - Standardize on the application protocol semantics
  - Ad hoc does not scale and leads to interoperability nightmares
- **Use outside of the browser**
  - Yes, there are examples and implementations, but they are the exception to the rule
- **Just because the Web "works" is not sufficient reason to assume it's right for everything**
  - HTTP versus IIOP?

Industrial realities

- **WS-* captured the mind-set**
- **REST is gaining momentum**
- **But …**
  - Large investment in WS-* and application-level standards are important
  - Probably too much effort to assume the same will happen in REST/HTTP
    - At least not with a big bang approach
  - More customers persuaded by WS-*
    - It's familiar (JEE, .NET, DCOM, DCE)
    - But they want future proof
- **Combinations of WS-* and REST are beginning to evolve**
  - May lead to standards

What can WS-* learn from REST?

- **Don't abuse transports, they don't like it!**
- **Adopt SOA principles**
  - No more WS-RF please!
  - WS-Context in favour of WS-Addressing "extensions"
- **Late binding is good**
  - But extremely late binding can be a burden
- **Occams Razor**
  - Simple but no simpler
- **Infrastructure support for common services/resources simplifies development**
- **The Web uses HTTP**

What should you use?

- **If it needs to be on the Web**
  - REST/HTTP
- **If it needs to be interoperable with arbitrary vendors**
  - WS-* for standards
  - Or persuade vendors to work on standards for REST
- **If you must use HTTP**
  - Consider REST before WS-*
    - But understand all of the implications
- **If integrating with back-end systems out-of-the-box**
  - WS-* has a lot of out-of-the-box solutions

The combination

- **WS-* used within the firewall**
  - REST principles could still help
  - SOAP is not fast and neither is HTTP
- **REST/HTTP for between firewalls**
  - Improved application protocol standards
- **Bridge between WS-* and REST/HTTP**
  - Leverage all of HTTP where possible
  - Definitely not easy to do, but …
- **WS-* between firewalls?**
  - Unlikely to see massive adoption

Conclusions

- **What have we learnt from the last 40 years?**
  - One size does not fit all!
  - Use the right tool for the right job
    - If all you've got is a hammer then of course everything looks like a nail!
- **Just because you are using HTTP does not mean you are using REST**
- **Just because you are using WS-* does not mean you are developing with SOA**