ENTERPRISE SOA CONFERENCE

24 October 2006, De Montil, Affligem

www.bejug.org
REST - the Better Web Services Model

Stefan Tilkov
Founder & Principal Consultant
innoQ Deutschland GmbH
Overall presentation goal

Understand the value of REST and its use for building scalable systems
Speaker’s Qualifications

- Architecting mission-critical systems since 1995
- Background with C/DCE, C++/CORBA, Java/J2EE
- SOA Community editor at InfoQ (www.infoq.com)
- Numerous articles and presentations on SOA, Web services, and architecture

See: http://www.innoq.com/blog/st/
If your goal is to create a loosely-coupled, evolvable, distributed system architecture based on standards and accessible to as many people as possible, you might think **Web services** are the best option.
REST as an Alternative to Web Services

If your goal is to create a loosely-coupled, evolvable, distributed system architecture based on standards and accessible to as many people as possible, you might think Web services are the best option.

You would be wrong.
Agenda

- Introducing REST
- REST vs. Web services
- Advanced use cases
- Summary
Agenda

- Introducing REST
- REST vs. Web services
- Advanced use cases
- Summary
What is REST?

- **RE**presentational **S**tate **T**ransfer
- Described by Roy Fielding in his dissertation
- One of a number of “architectural styles”
- Architectural principles underlying HTTP, defined *a posteriori*

REST and HTTP

- REST is an abstraction that could be implemented with any technology
- Best-known implementation of REST is HTTP
- Bear this in mind when HTTP is used to illustrate REST in the rest of this talk

http://savas.parastatidis.name/2005/03/12/505b74f7-d5d3-4b94-95d4-65129ce2bf2b.aspx
REST Key Principles

- Identifiable resources
- Uniform interface
- Stateless communication
- Resource representations
- Hypermedia
Identifiable resources

- A resource represents a real or virtual entity - a customer, vehicle fleet, shopping cart ...
- On the Web, resources are identified by URIs
- Each URI adds value to the Net as a whole
- Imagine Amazon.com without URIs!
Uniform interface

Once you know a resource's ID, you can interact with it using a single standard interface

Limited set of operations (verbs) in HTTP: GET, PUT, POST, DELETE (+ some more)

Pre-defined semantics allow for optimization (e.g. caching)
Resource representations

- Resources are always accessed through a representation
- There can be more than one
  - e.g. HTML, PDF, XML
- HTTP provides content types and content negotiation
- If possible, resources should be represented using well-known (ideally: standardized) content types
Stateless communication

- A server does not need to maintain state for each client
- Massive advantages in terms of scalability
- Enforces loose coupling (no shared session knowledge)
Hypermedia

- Possible (client) state transitions are made explicit through links
- Enable following of part-whole, detail, belongs-to and arbitrary other connections
- Links are (ideally) always provided by the server, not created by the client
- Enables seamless evolution and distribution
REST Approach

- A single *generic* (uniform) interface for everything
- Generic verbs mapped to resource semantics
- A standard application protocol (e.g. HTTP)
Contribution to the Net’s Value

- 1 URL *for each resource* - possibly millions
  - every customer
  - every order
- 4-7 supported methods per resource
  - GET, PUT, POST, DELETE
  - TRACE, OPTIONS, HEAD
- Cacheable, addressable, linkable, ...
Designing a RESTful application

- Identify resources & design URIs
- Select formats (or create new ones)
- Identify method semantics
- Select response codes

See: http://bitworking.org/news/How_to_create_a_REST_Protocol
RESTful HTTP Advantages

- Universal support (programming languages, operating systems, servers, ...)
- Proven scalability
- Support for redirect, caching, different representations, resource identification, ...
- "Real" web integration for machine-2-machine communication
- Support for XML, but also other formats
Agenda

- Introducing REST
- REST vs. Web services
- Advanced use cases
- Summary
Distributed Objects Approach

- A separate interface (façade) for each purpose
- As known CORBA, DCOM, RMI/EJB
- Often used for SOA ("CORBA w/ angle brackets")
- Application-specific protocol
Web Services and the Web

- WSDL/SOAP-based Web services are "protocol independent"

Examples:

- Microsoft supports SOAP over TCP
- Many Java implementations allow for SOAP with proprietary JMS implementations
- HTTP is just one more "transport protocol"
Contribution to the Net’s Value

- 2 URLs
  - http://example.com/customerservice
  - http://example.com/orderservice

- 1 method
  - POST
$ 64,000 Question:

- So how can one use HTTP’s features with SOAP Web services?
Some HTTP features

- Verbs (in order of popularity):
  - GET, POST
  - PUT, DELETE
  - HEAD, OPTIONS, TRACE
- Standardized (& meaningful) response codes
- Content negotiation
- Redirection
- Caching (incl. validation/expiry)
- Compression
- Chunking
$ 64,000 Answer
$ 64,000 Answer

- So how can one use HTTP’s features with SOAP Web services?
$ 64,000 Answer

- So how can one use HTTP’s features with SOAP Web services?
  - Not at all. At least not without breaking protocol independence.
$ 64,000 Answer

- So how can one use HTTP’s features with SOAP Web services?
  - Not at all. At least not without breaking protocol independence.

- HTTP is not a “dumb” transport protocol, but an application protocol
Web Services Issues
Web Services Issues

Web Services Standards

www.bejug.org
Web Services Issues

see http://www.innoq.com soa/ws-standards/poster
Web Services Issues
Web Services Issues
Web Services Issues

See http://www.loudthinking.com/arc/000585.html

www.bejug.org
Web Services Issues

“No matter how hard I try, I still think the WS-* stack is bloated, opaque, and insanely complex. I think it is going to be hard to understand, hard to implement, hard to interoperate, and hard to secure.”
Web Services Issues

“*No matter how hard I try, I still think the WS-* stack is bloated, opaque, and insanely complex. I think it is going to be hard to understand, hard to implement, hard to interoperate, and hard to secure.*”

Tim Bray, co-inventor of XML
“No matter how hard I try, I still think the WS-* stack is bloated, opaque, and insanely complex. I think it is going to be hard to understand, hard to implement, hard to interoperate, and hard to secure.”

Tim Bray, co-inventor of XML
http://www.tbray.org/ongoing/When/200x/2004/09/18/WS-Oppo
Web Services Issues

“Show me the interoperable, full and free implementations of WS-* in Python, Perl, Ruby and PHP. You won’t see them, because there’s no intrinsic value in WS-* unless you’re trying to suck money out of your customers. Its complexity serves as a barrier to entry at the same time that it creates “value” that can be sold.”
Web Services Issues

“Show me the interoperable, full and free implementations of WS-* in Python, Perl, Ruby and PHP. You won’t see them, because there’s no intrinsic value in WS-* unless you’re trying to suck money out of your customers. Its complexity serves as a barrier to entry at the same time that it creates “value” that can be sold.”

Mark Nottingham, formerly BEA,
Web Services Issues

“Show me the interoperable, full and free implementations of WS-* in Python, Perl, Ruby and PHP. You won’t see them, because there’s no intrinsic value in WS-* unless you’re trying to suck money out of your customers. Its complexity serves as a barrier to entry at the same time that it creates “value” that can be sold.”

Mark Nottingham, formerly BEA, former chair of the WS-Addressing WG
“Show me the interoperable, full and free implementations of WS-* in Python, Perl, Ruby and PHP. You won’t see them, because there’s no intrinsic value in WS-* unless you’re trying to suck money out of your customers. Its complexity serves as a barrier to entry at the same time that it creates “value” that can be sold.”

Mark Nottingham, formerly BEA, former chair of the WS-Addressing WG

http://www.mnot.net/blog/2006/05/10/vendors

www.bejug.org
Web Services Issues

- Web services and WS-* stack are supposed to create a new ubiquitous protocol stack on top of another ubiquitous protocol stack
- WS-* tends to ignore the web
- Abstractions “leak”, anyway
- Protocol independence is a bug, not a feature
Web Services Issues (contd.)

- The Web and the Internet architecture is based on standard protocols
- Not only HTTP, but also SMTP, FTP, DNS, ...
- If *web* services are supposed to work on Internet scale, they should be inspired by the Web, not by Distributed Objects
Agenda

- Introducing REST
- REST vs. Web services
- Advanced use cases
- Summary
Asynchronous Communication

- HTTP is always synchronous request/response
- For lengthy interactions, the server should return 202 Accepted and a URI for the result
- Poll or pass a URI to be notified
- WS-Addressing is just URIs used badly
Reliable Messaging w/ HTTP

- First question: What does “reliable” mean?
- Often solved at the application level
- Existing proposals:
  - Bill de hÓra’s HTTPLR
  - Yaron Goland’s SOA-Reliability (“SOA-Rity”)
2-Phase-Commit Transactions

- The Holy Grail of Applied Computer Science
- In practice, not used as often as you think
  “It hurts when I do that, Doctor.” – ”Then don’t do that!”
- 2PC and loose coupling don’t work together very well
- Compensating transactions are business logic anyway
- A light-weight protocol could be created, but no one has cared so far
Resource Access
Resource Access

WS-Transfer

This specification defines a mechanism for acquiring XML-based representations of entities using the Web service infrastructure. [...] Specifically, it defines two operations for sending and receiving the representation of a given resource and two operations for creating and deleting a resource and its corresponding representation.
Resource Access

- WS-Transfer

  This specification defines a mechanism for acquiring XML-based representations of entities using the Web service infrastructure. [...] Specifically, it defines two operations for sending and receiving the representation of a given resource and two operations for creating and deleting a resource and its corresponding representation.

- Sounds familiar?
Resource Access

- **WS-Transfer**

  *This specification defines a mechanism for acquiring XML-based representations of entities using the Web service infrastructure. [...] Specifically, it defines two operations for sending and receiving the representation of a given resource and two operations for creating and deleting a resource and its corresponding representation.*

- Sounds familiar?

- HTTP-over-SOAP-over-HTTP
Layers, we’ve got Layers
UDDI

- 420-page specification
- Finding and maintaining (meta-)model objects

**Inquiry**
- find_binding
- find_business
- find_relatedBusinesses
- find_service
- find_tModel
- get_bindingDetail
- get_businessDetail
- get_operationalInfo
- get_serviceDetail
- get_tModelDetail

**Publication**
- save_binding
- save_business
- save_service
- save_tModel
- delete_binding
- delete_business
- delete_publisherAssertions
- delete_service
- delete_tModel
- add_publisherAssertions
- set_publisherAssertions
- get_assertionStatusReport
- get_publisherAssertions
- get_registeredInfo
UDDI (contd.)

- UDDI could be greatly simplified by using plain HTTP
- It would no longer be protocol-independent
  - Who cares?
- Atom (Syndication Format & Protocol) would be a great match

See: http://www.xml.com/pub/a/ws/2002/02/06/rest.html?page=2
Binary Attachments

- In the WS-* world:
  - SOAP with Attachments (MIME)
  - DIME (supported by Microsoft)
  - XOP/MTOM

- In a RESTful HTTP application:
  - Use a link or
  - provide an alternative representation
Security

- Web services security is message-based
- HTTP relies on
  - transport level security (SSL/TLS)
  - basic and digest authentication
  - access control based on resources and methods
- WSS concepts would be a great value-add for HTTP-based systems
Agenda

- Introducing REST
- REST vs. Web services
- Advanced use cases
- Summary
Summary

- REST is the architecture of the world’s most successful distributed system
- Web Services don’t use the Web, they abuse it
- HTTP is not a transport protocol
- Very often, “Just use HTTP” is the best advice
- Understanding REST will help you build better Web-based systems
If You Only Remember One Thing…

HTTP is Good Enough.
Q&A
Thank you for your attention!

stefan.tilkov@innoq.com
http://www.innoq.com/blog/st/