The Service-Oriented Web

by Mark Baker, Coactus
presented by Makoto Murata

Who am I (Mark)

- Distributed systems specialist
- Former CORBA developer
- Former RMI developer
- Former DCE developer
- Former homegrown-RPC-over-TCP developer

- Achieved “Web nirvana”, 1998
  - Realized how the Web related to those other systems
- Now a consultant (Coactus)
  - Specializing in machine-to-machine integration on the Web

Mark's CORBA experience

- 1995; CORBA team lead for telecom project
  - Large project; 70 developers
  - Defined some common APIs used by other teams
- One common need was data transfer
  - Getting data from point A to point B
  - I defined getData() operation
- Defined it only for UI-facing objects
  - Not all CORBA objects implemented it
- But it was my first data transfer protocol!
- As I learned 2 1/2 years later ...

  HTTP was **FAR** superior to that protocol

Today's Talk

- A visit to 1998
- REST
- REST and SOA compared
- How to “Webize”
- Conclusion

A visit to 1998

- No SOAP
- No WSDL
- No UDDI
- No WS-*

- Just HTTP, URIs, XML

What *can't* we do?

Without SOAP, can we not ...

... order a pizza?

```
POST http://pizza.example.com/order HTTP/1.1
Content-Type: application/pizza+xml
<pizza>
   <size>X-Large</size>
   <type>thin crust</type>
   <topping>anchovies</topping>
   <topping>olives</topping>
</pizza>
```

Without SOAP, can we not ...

... turn on a light bulb?

```
PUT http://example.com/kitchen/main HTTP/1.1
Content-Type: application/lightbulb+xml
<lightbulb>
   <state>on</state>
</lightbulb>
```

Without WSDL, can we not ...

... discover a service's operations?

OPTIONS http://pizza.example.com/order HTTP/1.1

--->

HTTP/1.1 200 Ok
Allow: GET, POST

Without UDDI, can we not ...

... publish service metadata?

GET http://pizza.example.com/order HTTP/1.1

--->

```
<pizza-order-processor>
  <business-name>Joe's Pizza</business-name>
  <hours>11:00-02:00</hours>
  <minutes-or-free>30</minutes-or-free>
</pizza-order-processor>
```

Conclusion?

The **Web** *already* does services

Corollary ...

“**Web services**” are *unnecessary*

What now?

How do we build services in this Web-friendly way?

We need a *guide*

Today's Talk

- A visit to 1998
- **REST**
- REST and SOA compared
- How to “Webize”
- Conclusion

REST

- "REpresentational State Transfer"
  - "representational state"; a document
  - "transfer"; exchange between parties
  - Therefore, REST is for document exchange
- An architectural style
  - Abstract description of an architecture
  - Like "client/server", "pipe and filter", ...
- Independent of any particular technology
  - Though "REST for the Web" has technology *use* implications
- Used (by Roy Fielding) to craft HTTP, URI specs
- As with all arch styles, defined by its constraints ...

Key REST Constraints

- Uniform interface
- Resource identification
- Self-descriptive messages
- Hypermedia as engine of application state

Uniform Interface

- Operations must be meaningful to all resources
  - The “java.lang.Object” of network interfaces

  getRealTimeStockQuote: **not** uniform ...
  getStockQuote: **not** uniform ...
  getQuote: **not** uniform ...

  GET: ***uniform***

- Uniform implies *general*

Resource identification

- Identify your resources with a *standard syntax*
  - URIs for the Web
- “Resource”
  - Anything identifiable
  - e.g. Pizza order processor, your pizza order, the pizza store, toppings, ...

Self-descriptive Messages

- Stateless interactions
  - All data needed to understand the message, is in the message
    - No shared context on server
  - Cookies are not RESTful
- Standardized operations (uniform interface)
- Standardized media types

Hypermedia

- Clients make progress by following links
  - Not just with GET, but all operations
  - e.g. POST a form to a provided URI, get a doc back with more URIs
- No implicit links
  - e.g. Shouldn't specify “Append '/toppings' to all pizza order URIs to find the toppings for that order”
    - Search engines can't find toppings that way (without a software upgrade)

Today's Talk

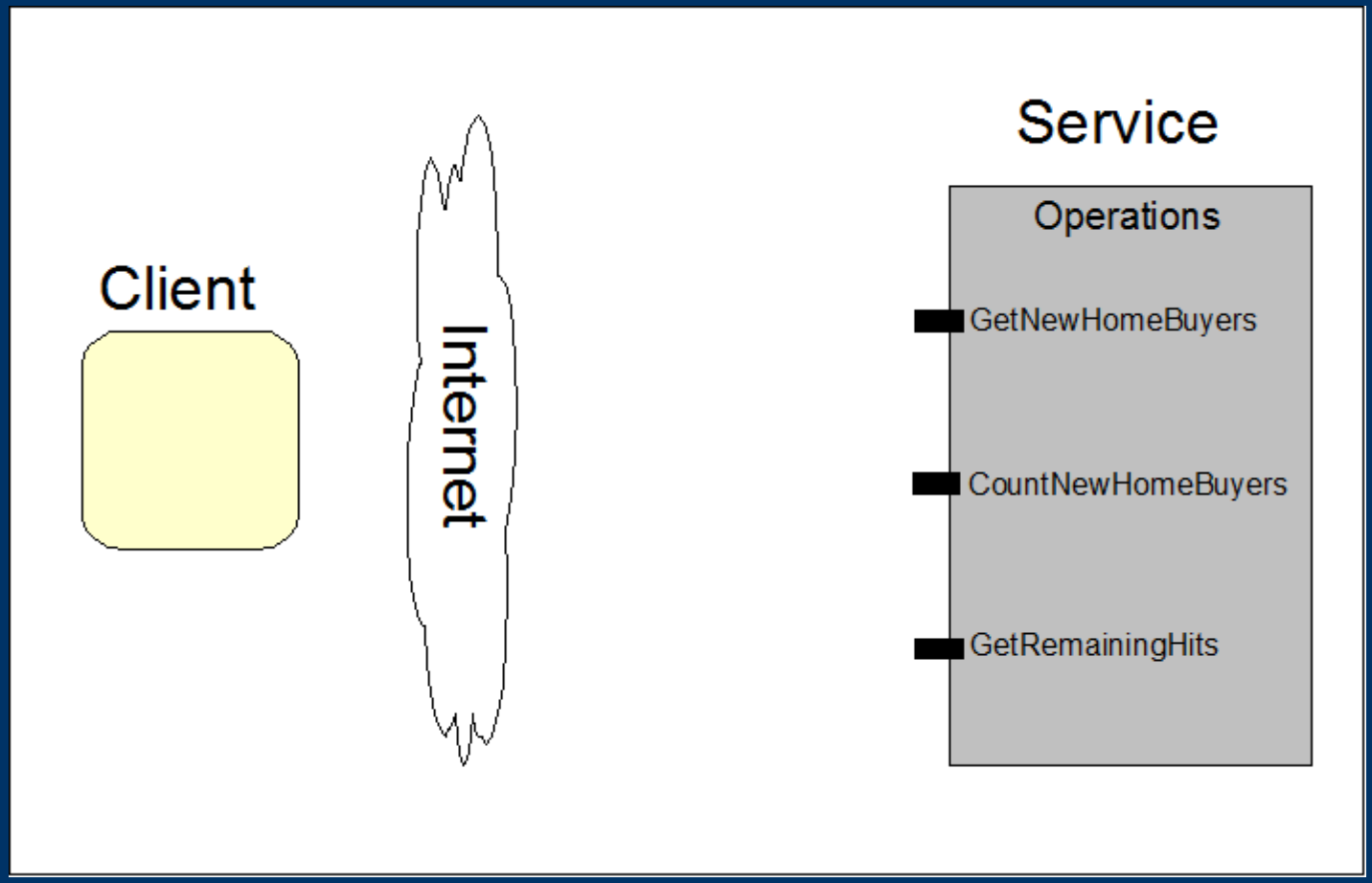
- A visit to 1998
- REST
- **REST and SOA compared**
- How to “Webize”
- Conclusion

REST and SOA Compared

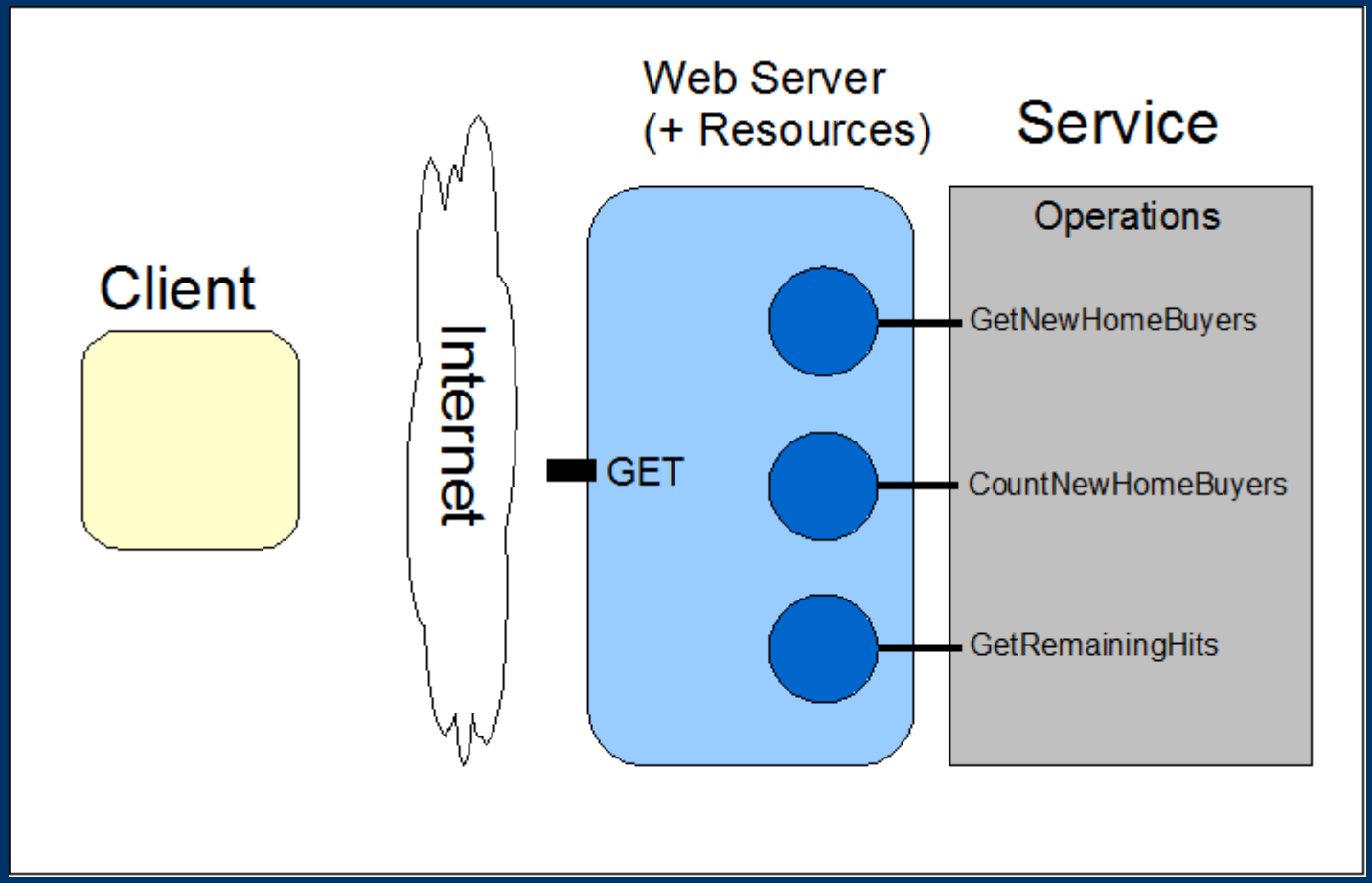
- Contrast typical Web service & Web approaches
- SOA example taken from;

  http://www.strikeiron.com/ProductDetail.aspx?p=168
  - "New home buyer" data

SOA/WS Example



REST/Web Equivalent



What are the differences?

- SOA client sees; one service, three operations
- REST client sees; three resources, one operation
- Is one better than the other?
  - One operation means substitutability
    - All HTTP clients can get data from all servers/resources
  - Other benefits of REST
    - “Ilities”; scalability, visibility, modifiability, evolvability, ...
    - See Roy Fielding's dissertation for the details
  - Less formally, REST approach is more ...

Loosely Coupled

Loose Coupling in REST

- Interface and implementation are *truly* separated
- The more specific the interface, the less the implementation can change
  - “getRealTimeStockQuote” can't be used for a delayed stock quote service
  - ... but “getStockQuote” can
  - (as can “GET”, of course)
- A very general (uniform) interface can support nearly arbitrary implementation changes

Today's Talk

- A visit to 1998
- REST
- REST and SOA compared
- **How to “Webize”**
- Conclusion

"Webize"

- Name from Tim Berners-Lee
  - http://www.w3.org/DesignIssues/Webize
- Requires no changes to existing system
- Just "wrap it" in a Web server
  - Web server as "Facade" (GoF design pattern)
    - Clients see only Web server, not application-specific interfaces
  - URIs point to resources *within* existing system

Webizing 101

- Give URIs to sources of distinct chunks of data
  - "GetNewHomeBuyers" one source of data
  - "CountNewHomeBuyers" another source of data
- But use nouns, not verbs, in URIs;

  BAD: http://example.org/GetNewHomeBuyers
    - Because PUT on that URI makes no sense

  GOOD: http://example.org/newHomeBuyers

  - The "GetNewHomeBuyers" operation is *completely opaque* to clients and humans
- PUT use is implicit
  - GET data, change it, PUT data back

Webizing 101; granularity

- Fine grained get* operations might be better off as one resource
  - e.g for getFirstName(), getLastName(), ... consider one "Person" resource which answers GET with;

```
<Person>
  <FirstName>Fred</FirstName>
  <LastName>Jones</LastName>
</Person>
```

Webizing 101; data sinks

- Webizing supports not just data *sources*, but also data *sinks*
  - Accept data submitted via POST
  - Much like an email inbox
- Give URIs to distinct sinks
  - e.g. Pizza order processor

Webizing 101; hypermedia & forms

- Include URIs in response documents
  - But sometimes URIs are not enough
  - How do you know what data to POST to a URI?
  - ... or how to parameterize a GET (like a “query URI”)?
- Need to provide this information along with URI
  - Known as “forms language”, e.g. HTML Forms, XForms, RDF Forms
  - Can be integrated into application specific format ...

Webizing 101; example “pizza form”

<pizza-order-processor
href=”http://pizza.example.org/order”>

<accept>application/pizza+xml</accept>

</pizza-order-processor>

Conclusion; the bad news

- “Web based services” very different than “Web services”
  - Tough learning curve for SOA/WS developers
- Not much of WS-* salvageable
- Stuck supporting SOAP services

Conclusion; the GOOD news

- Relative simplicity in Web based approach
- Pervasive, mature tooling
- Pervasive, mature infrastructure
  - Firewalls, caches, accelerators, ...
- A (hard) lesson learned
  - The end of the “plumbing wars”
- Reuse skills of millions of Web developers

Hug a Web developer today!

Thanks!