Distributed Systems

04. RPC & Web Services: Case Studies

Paul Krzyzanowski
Rutgers University
Fall 2016

Overview of RPC Systems & Web Services

1. Remote Procedure Calls
2. Remote Objects
3. Web Services

ONC (Sun) RPC

ONC (Sun) RPC

- RPC for Unix System V, Linux, BSD, OS X
  - ONC = Open Network Computing
  - Created by Sun
  - RFC 1831 (1995), RFC 5531 (2009)
  - Remains in use mostly because of NFS (Network File System)
- Interfaces defined in an Interface Definition Language (IDL)
- IDL compiler is *rpcgen*

RPC IDL

name.x

```
program GETNAME {
    version GET_VERS {
        long GET_ID(string<50>) = 1;
        string GET_ADDR(long) = 2;
    } = 1;   /* version */
} = 0x31223456;
```

Interface definition: version 1

Why is versioning important?

name.x

```
program GETNAME {
    version GET_VERS {
        long GET_ID(string<50>) = 1;
        string GET_ADDR(long) = 2;
    } = 1;   /* version */
    version GET_VERS2 {
        long GET_ID(string<50>) = 1;
        string GET_ADDR(string<128>) = 2;
    } = 2;   /* version */
} = 0x31223456;
```

Interface definition: version 2

rpcgen

`rpcgen name.x`

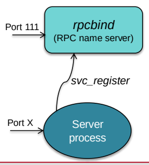
produces:

– name.h	header
– name_svc.c	server stub (skeleton)
– name_clnt.c	client stub
– [name_xdr.c]	optional XDR conversion routines

- Function names derived from IDL function names and version numbers
- Client gets *pointer* to result
  - Allows it to identify failed RPC (null return)
  - Reminder: C doesn't have exceptions!

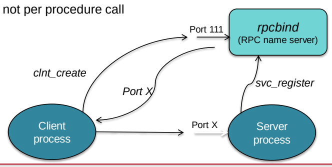
What goes on in the system: server

Start server
- Server stub creates a socket and binds any available local port to it
- Calls a function in the RPC library:
  - *svc_register* to register program#, port #
  - Contacts the port mapper, rpcbind (portmap on older Linux systems):
    - Name server
    - Keeps track of {program #, version #, protocol} → port # bindings
- Server then listens and waits to accept connections



What goes on in the system: client

- Client calls clnt_create with:
  - Name of server
  - Program #
  - Version #
  - Protocol#
- *clnt_create* contacts port mapper on that server to get the port for that interface
  - early binding – done once, not per procedure call
- Communications
  - Marshaling to XDR format (eXternal Data Representation)



Advantages

- Don't worry about getting a unique transport address (port)
  - But with you need a unique program number per server
  - Greater portability
- Transport independent
  - Protocol can be selected at run-time
- Application does not have to deal with maintaining message boundaries, fragmentation, reassembly
- Applications need to know only one transport address
  - Port mapper (portmap process)
- Function call model can be used instead of send/receive
- Versioning support between client & server

DCE RPC

DCE RPC

- DCE: set of components designed by The Open Group (merger of OSF and X/Open) for providing support for distributed applications
  - Distributed file system service, time service, directory service, …
  - Not currently adopted in any popular versions of UNIX/Linux
- There was room for improvement in ONC (Sun) RPC

DCE RPC

- Similar to ONC RPC
- Interfaces written in an Interface Definition Notation (IDN)
  - Definitions look like function prototypes
- Run-time libraries
  - One for TCP/IP and one for UDP/IP
- Authenticated RPC support with DCE security services
- Integration with DCE directory services to locate servers

Unique IDs

ONC RPC required a programmer to pick a "unique" 32-bit number

DCE: get unique ID with uuidgen

- Generates prototype IDN file with a 128-bit Unique Universal ID (UUID)
- 10-byte timestamp multiplexed with version number
- 6-byte node identifier (ethernet address on ethernet systems)

IDN compiler

Similar to rpcgen:

Generates header, client, and server stubs

Service lookup

Sun RPC requires client to know name of server

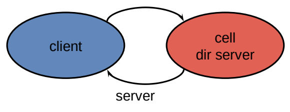
DCE allows several machines to be organized into an administrative entity

cell (collection of machines, files, users)

Cell directory server

Each machine communicates with it for cell services information

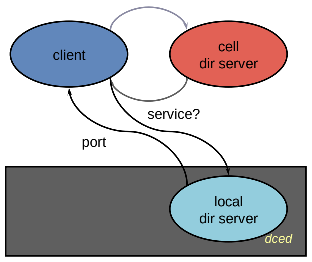
DCE service lookup



Request service lookup from cell directory server

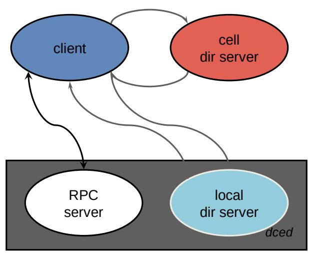
Return server machine name

DCE service lookup



Connect to endpoint mapper service and get port binding from this local name server

DCE service lookup



Connect to service and request remote procedure execution

Marshalling

Standard formats for data
- NDR: Network Data Representation

Goal
- *Multi-canonical* approach to data conversion
  - Fixed set of alternate representations
  - Byte order, character sets, and floating-point representation can assume one of several forms
  - Sender can (hopefully) use native format
  - Receiver may have to convert

What's Cool

- DCE RPC improved Sun RPC
  - Unique Universal ID
  - Multi-canonical marshalling format
  - *Cell* of machines with a cell directory server
    - No need to know which machine provides a service

Sun and DCE RPC deficiencies

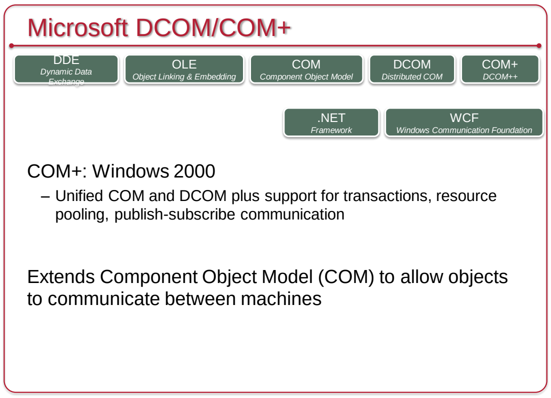
- If server is not running
  - Service cannot be accessed
  - Administrator responsible for starting it
- If a new service is added
  - There is no mechanism for a client to discover this
- Object oriented languages expect polymorphism
  - Service may behave differently based on data types passed to it

The next generation of RPCs

Distributed objects:
support for object oriented languages

DOA: Distributed Object Architecture

Microsoft COM+ (DCOM)



Microsoft DCOM/COM+

DDE
Dynamic Data Exchange

OLE
Object Linking & Embedding

COM
Component Object Model

DCOM
Distributed COM

COM+
DCOM++

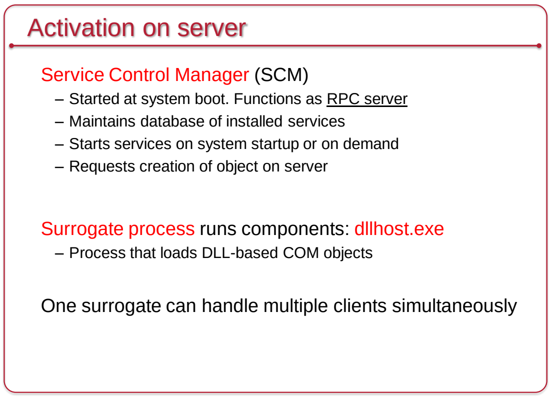
.NET
Framework

WCF
Windows Communication Foundation

COM+: Windows 2000
- Unified COM and DCOM plus support for transactions, resource pooling, publish-subscribe communication

Extends Component Object Model (COM) to allow objects to communicate between machines

September 19, 2016 © 2013-2016 Paul Krzyzanowski 25



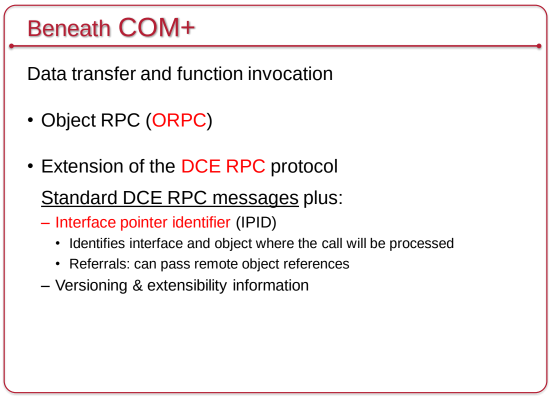
Activation on server

Service Control Manager (SCM)
- Started at system boot. Functions as RPC server
- Maintains database of installed services
- Starts services on system startup or on demand
- Requests creation of object on server

Surrogate process runs components: dllhost.exe
- Process that loads DLL-based COM objects

One surrogate can handle multiple clients simultaneously

September 19, 2016 © 2013-2016 Paul Krzyzanowski 26



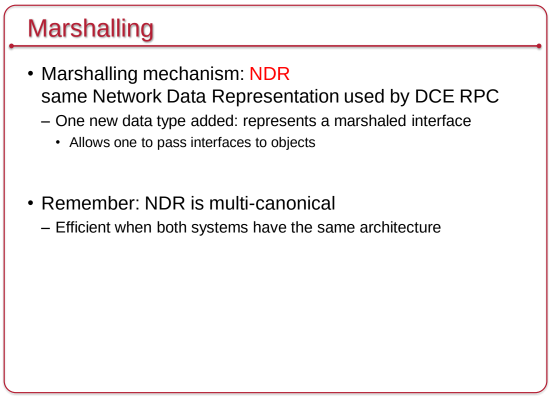
Beneath COM+

Data transfer and function invocation

- Object RPC (ORPC)
- Extension of the DCE RPC protocol

  Standard DCE RPC messages plus:
  - Interface pointer identifier (IPID)
    - Identifies interface and object where the call will be processed
    - Referrals: can pass remote object references
  - Versioning & extensibility information

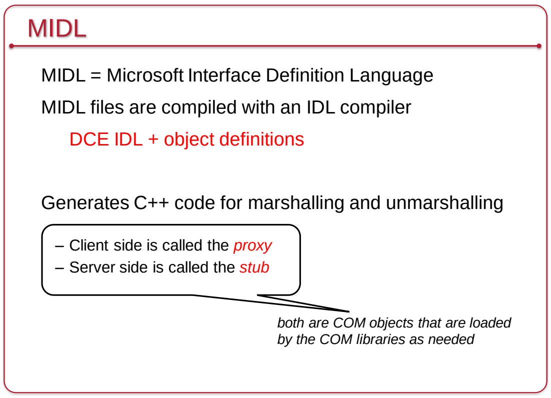
September 19, 2016 © 2013-2016 Paul Krzyzanowski 27



Marshalling

- Marshalling mechanism: NDR
  same Network Data Representation used by DCE RPC
  - One new data type added: represents a marshaled interface
    - Allows one to pass interfaces to objects
- Remember: NDR is multi-canonical
  - Efficient when both systems have the same architecture

September 19, 2016 © 2013-2016 Paul Krzyzanowski 28



MIDL

MIDL = Microsoft Interface Definition Language

MIDL files are compiled with an IDL compiler

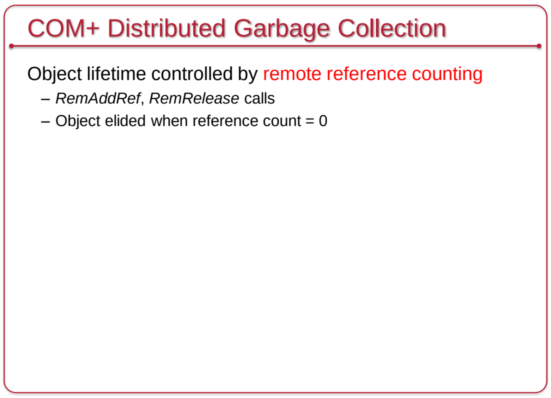
DCE IDL + object definitions

Generates C++ code for marshalling and unmarshalling

- Client side is called the *proxy*
- Server side is called the *stub*

both are COM objects that are loaded by the COM libraries as needed

September 19, 2016 © 2013-2016 Paul Krzyzanowski 29



COM+ Distributed Garbage Collection

Object lifetime controlled by remote reference counting
- *RemAddRef*, *RemRelease* calls
- Object elided when reference count = 0

September 19, 2016 © 2013-2016 Paul Krzyzanowski 30

COM+ Distributed Garbage Collection

Abnormal client termination
- Insufficient *RemRelease* messages sent to server
- Object will not be deleted

In addition to reference counting:

Client Pinging
- Server has *pingPeriod, numPingsToTimeOut*
- Relies on client to ping
  - background process sends ping set – IDs of all remote objects on server
- If ping period expires with no pings received, all references are cleared

Microsoft DCOM/COM+ Contributions

- Fits into Microsoft COM model
- Generic server hosts dynamically loaded objects
  - Requires unloading objects (dealing with dead clients)
  - Reference counting and pinging
- Support for references to instantiated objects
- But… COM+ was a Microsoft-only solution
  - And it did not work well across firewalls because of dynamic ports

Java RMI

Java RMI

- Java language had no mechanism for invoking remote methods
- 1995: Sun added extension
  - Remote Method Invocation (RMI)
  - Allow programmer to create distributed applications where methods of remote objects can be invoked from other JVMs

RMI components

Client
- Invokes method on remote object

Server
- Process that owns the remote object

Object registry
- Name server that relates objects with names

Interoperability

RMI is built for Java only!
- No goal of OS interoperability (as CORBA)
- No language interoperability (goals of SUN, DCE, and CORBA)
- No architecture interoperability

No need for external data representation
- All sides run a JVM

Benefit: simple and clean design

RMI similarities

Similar to local objects

- References to remote objects can be passed as parameters *(not as pointers, of course)*
  - You can execute methods on a remote object
- Objects can be passed as parameters to remote methods
- Object can be cast to any of the set of interfaces supported by the implementation
  - Operations can be invoked on these objects

RMI differences

- Objects (parameters or return data) passed by value
  - Changes will visible only locally
- Remote objects are passed by reference
  - Not by copying remote implementation
  - The “reference” is not a pointer. It’s a data structure:
    { IP address, port, time, object #, interface of remote object }
- RMI generates extra exceptions

Classes to support RMI

- remote class:
  - One whose instances can be used remotely
  - Within its address space: regular object
  - Other address spaces:
    - Remote methods can be referenced via an object handle
- serializable class:
  - Object that can be marshaled
  - If object is passed as parameter or return value of a remote method invocation, the value will be copied from one address space to another
    - If remote object is passed, only the object handle is copied between address spaces

Classes to support RMI

- remote class:
  - One whose instances can be used remotely
  - Within its address space: regular object
  - Other address spaces:
    - Remote methods can be referenced via an object handle

needed for remote objects

- serializable class:
  - Object that can be marshaled
  - If object is passed as parameter or return value of a remote method invocation, the value will be copied from one address space to another
    - If remote object is passed, only the object handle is copied between address spaces

needed for parameters

Stub & Skeleton Generation

- Automatic stub generation since Java 1.5
  - Need stubs and skeletons for the remote interfaces
  - Automatically built from java files
  - Pre 1.5 (still supported) generated by separate compiler: *rmic*
- Auto-generated code:
  - Skeleton
    - Server-side code that calls the actual remote object implementation
  - Stub
    - Client side proxy for the remote object
    - Communicates method invocations on remote objects to the server

Naming service

We need to look an object up by name

Get back a remote object reference to perform remote object invocations

Object registry does this: rmiregistry running on the server

Server

Register object(s) with Object Registry

```
Stuff obj = new Stuff();
Naming.bind("MyStuff", obj);
```

Client

Client contacts *rmiregistry* to look up name

```
MyInterface test = (MyInterface)
    Naming.lookup("rmi://www.pk.org/MyStuff");
```

rmiregistry service returns a remote object reference.

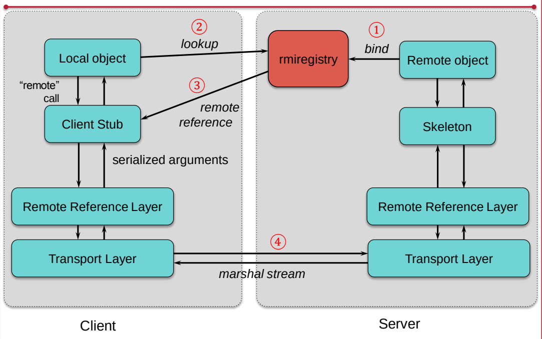
lookup method gives reference to local stub.

The stub now knows where to send requests

Invoke remote method(s):

```
        test.func(1, 2, "hi");
```

Java RMI infrastructure



RMI Distributed Garbage Collection

- Two operations: *dirty* and *clean*
- Local JVM sends a *dirty* call to the server JVM when the object is in use
  - The *dirty* call is refreshed based on the lease time given by the server
- Local JVM sends a *clean* call when there are no more local references to the object
- Unlike DCOM:
  no incrementing/decrementing of references

Remote Services

Web Services

From Web Browsing to Web Services

- Web browser:
  - Dominant model for user interaction on the Internet
- Not good for programmatic access to data or manipulating data
  - UI is a major component of the content
  - *Site scraping* is a pain!

Web Services

- We want
  - Remotely hosted services – that *programs* can use
  - Machine-to-machine communication
- Problems
  - Web pages are content-focused
  - Traditional RPC solutions usually used a range of ports
    - And we need more than just RPC sometimes
  - Many RPC systems didn't work well across languages
  - Firewalls restrict ports & may inspect the protocol
  - No support for load balancing

Web Services

- Set of protocols by which services can be published, discovered, and used in a technology neutral form
  - Language & architecture independent
- Applications will typically invoke multiple remote services
  - Service Oriented Architecture
- General principles
  - Payloads are text (XML or JSON)
    - Technology-neutral
  - HTTP used for transport
    - Use existing infrastructure: web servers, firewalls, load-balancers

Service Oriented Architecture (SOA)

- SOA = Programming model
- App is integration of network-accessible services (components)
- Each service has a well-defined interface
- Components are unassociated & loosely coupled

Neither service depends on the other: all are mutually independent

Neither service needs to know about the internal structure of the others

XML RPC

Origins

- Born: early 1998
- Data marshaled into XML messages
  - All request and responses are human-readable XML
- Explicit typing
- Transport over HTTP protocol
  - Solves firewall issues
- No IDL compiler support for most languages
  - Lots of support libraries for other languages
  - Great support in some languages – those that support introspection (Python, Perl)
- Example: WordPress uses XML-RPC

XML-RPC example

```
<methodCall>
    <methodName>
        sample.sumAndDifference
    </methodName>
    <params>
        <param><value><int> 5 </int></value></param>
        <param><value><int> 3 </int></value></param>
    </params>
</methodCall>
```

XML-RPC data types

- int
- string
- boolean
- double
- dateTime.iso8601
- base64
- array
- struct

Assessment

- Simple (spec about 7 pages)
- Humble goals
- Good language support
  - Little/no function call transparency for some languages
- No garbage collection, remote object references, etc.
  - Focus is on data messaging over HTTP transport
- Little industry support (Apple, Microsoft, Oracle, …)
  - Mostly grassroots and open source

SOAP

SOAP origins

(Simple) (Object) Access Protocol

- Since 1998 (latest: v1.2 April 2007)
- Started with strong Microsoft & IBM support
- Specifies XML format for messaging
  - Not necessarily RPC
- Continues where XML-RPC left off:
  - XML-RPC is a 1998 simplified subset of SOAP
  - user defined data types
  - ability to specify the recipient
  - message specific processing control
  - and more …
- Usually XML over HTTP

SOAP

- Stateless messaging model
- Basic facility is used to build other interaction models
  - Request-response
  - Request-multiple response
- Objects marshaled and unmarshaled to SOAP-format XML
- Like XML-RPC, SOAP is a messaging format
  - No garbage collection or object references
  - Does not define transport
  - Does not define stub generation

From Messaging to Web Services

- Things like SOAP give us a messaging structure
- What else is useful for services?
  - Service definition: create software to create the right SOAP messages
  - Service discovery
  - Message delivery

Web Services and WSDL

- Web Services Description Language
  - Analogous to an IDL
- A WSDL document describes a set of services
  - Name, operations, parameters, where to send requests
  - Goal is that organizations will exchange WSDL documents
    - If you get WSDL document, you can feed it to a program that will generate software to send and receive SOAP messages

WSDL Document Contents

Service
Name & address

Port 1
Methods for accessing the service
Example ports: SOAP, HTTP GET, HTTP POST

Binding
Describes the data format & protocol for a port
E.g., RPC style

PortType 1
Operations that the service can implement
Messages: definition of the input and output message for each operation

Port 2 …

WSDL Structure

```
<definitions>
  <types>
    data type used by web service: defined via XML Schema syntax
  </types>
  <message>
    describes data elements of operations: parameters
  </message>
  <portType>
    describes service: operations, and messages involved
  </portType>
  <binding>
    defines message format & protocol details for each port
  </binding>
</definitions>
```

WSDL structure: port types

```
<definitions  name="MobilePhoneService" target=…>
  1. type definitions
  <portType name="MobilePhoneService_port">
      <operation name="getListOfModels">
          <output message="ListOfPhoneModels"/>
      <operation name="getPrice">
          <Input message="PhoneModel"/>
          <output message="PhoneModelPrice"/>
  3. messaging spec
```

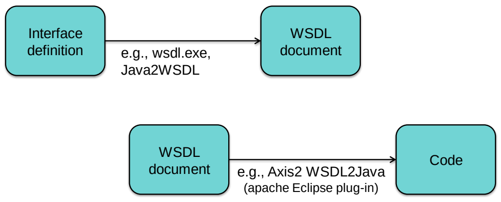
2. service definition

WSDL part 3: messaging spec

```
<binding name="MobilePhoneService_Binding"
         type="MobilePhoneService_port">
  <soap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="getPrice">
    <soap:operation soapAction="urn:MobilePhoneService"/>
    <input>
      <soap:body encodingStyle=
           "http://schemas.xmlsoap.org/soap/encoding/"
           namespace="urn:MobilePhoneService" use="encoded"/>
    </input>
    <output>
      <soap:body encodingStyle=
           "http://schemas.xmlsoap.org/soap/encoding/"
           namespace="urn:MobilePhoneService" use="encoded" />
    </output>
  </operation>
</binding>
```

What do we do with WSDL?

It's an IDL – not meant for human consumption


Interface definition
e.g., wsdl.exe, Java2WSDL
WSDL document
WSDL document
e.g., Axis2 WSDL2Java (apache Eclipse plug-in)
Code

Java Web Services

JAX-WS: Java API for XML Web Services

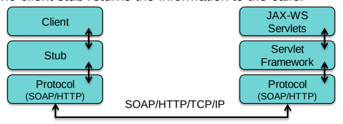
- Lots of them! We'll look at one
- JAX-WS (evolved from earlier JAX-RPC)
  - Java API for XML-based Web-Service messaging & RPCs
  - Invoke a Java-based web service using Java RMI
  - Interoperability is a goal
    - Use SOAP & WSDL
    - Java not required on the other side (client or server)
- Service
  - Defined to clients via a WSDL document

JAX-WS: Creating an RPC Endpoint

- Server
  - Define an interface (Java interface)
  - Implement the service
  - Create a publisher
    - Creates an instance of the service and publishes it with a name
- Client
  - Create a proxy (client-side stub)
    - *wsimport* command takes a WSDL document and creates a stub
  - Write a client that creates an instance of the service and invokes methods on it (calling the proxy)

JAX-RPC Execution Steps

1. Java client calls a method on a stub
2. Stub calls the appropriate web service
3. Server gets the call and directs it to the framework
4. Framework calls the implementation
5. The implementation returns results to the framework
6. The framework returns the results to the server
7. The server sends the results to the client stub
8. The client stub returns the information to the caller



Web Clients: AJAX

Asynchronous JavaScript And XML
- Bring web services to web clients (JavaScript)

- Asynchronous
  - Client not blocked while waiting for result
- JavaScript
  - Request can be invoked from JavaScript (using HTTPRequest)
  - JavaScript may also modify the Document Object Model (CSS) – how the page looks
- XML
  - Data sent & received as XML

AJAX & XMLHTTP

- Allow Javascript to make HTTP requests and process results (change page without refresh)

```
xmlhttp = new XMLHttpRequest();
xmlhttp.open("POST", "demo.html", true);
xmlhttp.send();
```

- Tell object:
  - Type of request you're making
  - URL to request
  - Function to call when request is made
  - Info to send along in body of request

AJAX on the Web

- AJAX ushered in Web 2.0
- Early high-profile AJAX sites:
  - Google Maps, Gmail, Kayak, Amazon Zuggest, Del.icio.us Director, Writely, …

The future of SOAP?

- SOAP
  - Dropped by Google in 2006
  - Alternatives: AJAX, XML-RPC, REST, …
  - Allegedly complex because "we want our tools to read it, not people"
    - – unnamed Microsoft employee
- Microsoft
  - Provides a mix of REST, JSON, and SOAP APIs
  - http://www.bing.com/developers/
- Still lots of support

REST

REpresentational State Transfer

- Stay with the principles of the web
  - Four HTTP commands let you operate on data (a resource):
    - PUT (create)
    - GET (read)
    - POST (update)
    - DELETE (delete)

    CRUD: *Create, Read, Update, Delete*
- Messages contain representation of data

Resource-oriented services

- Blog example
  - Get a snapshot of a user's blogroll:
    - HTTP GET //rpc.bloglines.com/listsubs
    - HTTP authentication handles user identification
  - To get info about a specific subscription:
    - HTTP GET http://rpc.bloglines.com/getitems?s={subid}

Resource-oriented services

- Get parts info
  HTTP GET //www.parts-depot.com/parts
- Returns a document containing a list of parts

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com"
      xmlns:xlink="http://www.w3.org/1999/xlink">
    <Part id="00345" xlink:href="http://www.parts-depot.com/parts/00345"/>
    <Part id="00346" xlink:href="http://www.parts-depot.com/parts/00346"/>
    <Part id="00347" xlink:href="http://www.parts-depot.com/parts/00347"/>
    <Part id="00348" xlink:href="http://www.parts-depot.com/parts/00348"/>
</p:Parts>
```

Resource-oriented services

- Get detailed parts info:
  HTTP GET //www.parts-depot.com/parts/00345
- Returns a document with information about a specific part

```
?xml version="1.0"?>
<p:Part xmlns:p="http://www.parts-depot.com" xmlns:xlink="http://www.w3.org/1999/xlink">
    <Part-ID>00345</Part-ID>
    <Name>Widget-A</Name>
    <Description>This part is used within the frap assembly</Description>
    <Specification xlink:href="http://www.parts-depot.com/parts/00345/specification"/>
    <UnitCost currency="USD">0.10</UnitCost>
    <Quantity>10</Quantity>
</p:Part>
```

REST vs. RPC Interface Paradigms

Example from wikipedia:

RPC

getUser(), addUser(), removeUser(), updateUser(),
getLocation(), AddLocation(), removeLocation()

```
exampleObject = new ExampleApp("example.com:1234");
   exampleObject.getUser();
```

REST

http://example.com/users
http://example.com/users/{user}
http://example.com/locations

```
userResource =
   new Resource("http://example.com/users/001");
   userResource.get();
```

Examples of REST services

- Various Amazon APIs
- Yahoo! Search APIs
- Flickr
- Twitter
- Google Glass
- Open Zing Services – Sirius radio
  svc://Radio/ChannelList
  svc://Radio/ChannelInfo?sid=001-siriushits1&ts=2007091103205
- Tesla Model S & Model X
  https://owner-api.teslamotors.com/api/1/vehicles/vehicle_id/command/set_valet_mode
  https://owner-api.teslamotors.com/api/1/vehicles/vehicle_id/command/flash_lights

The End