Overview of RPC Systems & Web Services

1. Remote Procedure Calls
2. Remote Objects
3. Web Services
4. Just Marshaling

ONC (Sun) RPC

- RPC for Unix System V, Linux, BSD, OS X
  - ONC = Open Network Computing
  - Created by Sun
  - Remains in use mostly because of NFS (Network File System)

- Interfaces defined in an Interface Definition Language (IDL)
- IDL compiler is rpcgen

RPC IDL

Interface definition: version 1

Why is versioning important?

Interface definition: version 2

ONC (Sun) RPC

name.x

program GETNAME
{
  version GET VERS (long) = 1;
  string GET_ADDR(long) = 2;
} = 1; /* version */

name.x

program GETNAME
{
  version GET VERS (long) = 1;
  string GET_ADDR(string<50>) = 1;
  string GET_ADDR(string<128>) = 2;
} = 2; /* version */
**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 `portmap` (rpcbind on SVR4):
    - 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: 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

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+

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

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

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

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

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

COM+ Distributed Garbage Collection

Object lifetime controlled by remote reference counting

- RemAddRef, RemRelease calls
- Object elided when reference count = 0
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

Problems with COM+/DCOM

- Originally designed for object linking and embedding
- Relatively low-level implementation
- Objects had to provide reference counting explicitly
- Languages & libraries provided varying levels of support
  - A lot for VB, less for C++

⇒ .Net Remoting tried to fix all this

Object Activation

- Server Activated Objects
  - Single Call: new instance per call (stateless)
  - Created when the client invokes the first method
  - Singleton: same instance for all requests (shared state)
  - If the instance does not exist, the server creates one and all other requests from clients use it
  - Neither of these require garbage collection

- Client Activated Objects
  - Lifetime is controlled by the client. Created when client calls new
  - Similar to DCOM (COM+) model
  - Supports distributed garbage collection

Microsoft .NET Remoting

- marshalling
- Listener
- channel
- TCP binary
- HTTP/SOAP
- Named pipes

Leasing Distributed Garbage Collector (LDGC)

- Used with Client Activated Objects
  - No more reference counting!
  - No pinging
- Lease Manager manages object leases at a server
  - Server object is in use as long as its lease has not expired
  - If a client wants to be contacted when a lease expires, it needs to provide a sponsor object. The sponsor object can then extend the lease.
- Each time a method is called:
  - renewOnCallTime: amount of time to renew lease after each method call
  - Lease time set to MAX(lease time - expired time, RenewOnCallTime)
  - Requestor has to renew lease when leaseTime elapses
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 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:
    - Information on how to get to contact the remote object
    - (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

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

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

Client

Client contacts rmiregistry to look up name

```java
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):

```java
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 … even with CSS
  - Site scraping is a pain!
- We want
  - Remotely hosted services — that programs can use
  - Machine-to-machine communication

Why Not RPC?
- Problems
  - Web pages are content-focused
  - Traditional RPC solutions usually used a range of ports
    - And we sometimes need more than just RPC interactions
  - Many RPC systems didn’t work well across languages
  - Firewalls restrict ports & may inspect the protocol
  - No support for load balancing

Web Services

General Principles
- HTTP over TCP/IP
  - Use existing infrastructure: web servers, firewalls, load-balancers
- Send documents to a server
  - Get documents back
- Tolerate high latency
- Tolerate large # of clients and client disappearance → stateless design

What’s a document?
- Richer (larger) data structures – self describing
- Textual data: usually XML or JSON
  - technology-neutral
- Loose coupling between clients & servers → stateless
- Asynchronous interactions in mind → high latency
Service Oriented Architecture (SOA)

- Applications will typically invoke multiple remote services
  - 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

Web Services vs. Distributed Objects

Web Services
- Focus on document exchange – designed with high latency in mind
- Interfaces are just ways to pass documents
  - Document design is central

Distributed Objects
- Interfaces are central to the design
- Data structures ("documents") just package data for use by interfaces

Web Services vs. Distributed Objects

- Web services usually lack the capabilities of distributed objects
  - Dynamic object creation
  - Garbage collection
  - Object references
  - State management
- But…
  - Interoperability is usually easier
  - Support various interaction models
    - RPC, async messaging, transactions, message queuing, event notification
  - Fits into existing firewall & load balancing infrastructures

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

<methodCall>
  <methodName>
    sample_sumAndDifference
  </methodName>
  <params>
    <param><value><int>5</int></value></param>
    <param><value><int>3</int></value></param>
  </params>
</methodCall>

XML-RPC example
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 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

From Messaging to Web Services

- Things like SOAP give us a messaging structure
- What else is useful for services?
  - Service definition
    - WSDL – Web Services Description Language
    - create software to create the right SOAP messages
    - Similar to an IDL (interface definition language)
  - Service discovery
    - UDDI – Universal Description Discovery & Integration
    - Never really took off outside companies
  - Message delivery
    - HTTP over TCP/IP

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 about a messaging format
  - No garbage collection or object references
  - Does not define transport
  - Does not define stub generation
What do we do with WSDL?

WSDL is an IDL – but not meant for human consumption

- Language-specific interface definition (e.g., wsdl.exe, Java2WSDL)
- 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

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)

GRUD: 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="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"?>
  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>
  <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.getUser();

Examples of REST services

• Various Amazon APIs
• Yahoo! Search APIs
• Flickr
• Twitter
• Google Glass
• Open Zing Services – Sirius radio
• Tesla Model S

svc://Radio/ChannelList
svc://Radio/ChannelInfo?sid=001-siriushits1&ts=2007091103205
**JSON: JavaScript Object Notation**

- Lightweight (relatively efficient) data interchange format
  - Introduced as the “fat-free alternative to XML”
  - Based on JavaScript
- Human writeable and readable
- Self-describing (explicitly typed)
- Language independent
- Easy to parse
- Currently converters for 50+ languages
- Includes support for RPC invocation via JSON-RPC

**Example (from the Developer Guide)**

```protobuf
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }
  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }
  repeated PhoneNumber phone = 4;
}
```

**Google Protocol Buffers**

- Efficient mechanism for serializing structured data
  - Much simpler, smaller, and faster than XML
- Language independent
- Define messages
  - Each message is a set of names and types
- Compile the messages to generate data access classes for your language
- Used extensively within Google. Currently over 48,000 different message types defined.
  - Used both for RPC and for persistent storage

**Example (from the Developer Guide)**

```protobuf
Person person;
person.set_name("John Doe");
person.set_id(1234);
person.set_email("jdoe@example.com");
fstream output("myfile", ios::out | ios::binary);
person.SerializeToOstream(&output);
```
Efficiency example (from the Developer Guide)

- Binary encoded message: ~28 bytes long, 100-200 ns to parse
- XML version: ≥69 bytes, 5,000-10,000 ns to parse

XML version

```
<person>
  <name>John Doe</name>
  <email>jdoe@example.com</email>
</person>
```

Text (uncompiled) protocol buffer

```
person {
  name: "John Doe"
  email: "jdoe@example.com"
}
```

http://code.google.com/apis/protocolbuffers/docs/overview.html