

---

## Case Studies

Sun RPC  
DCE RPC  
DCOM  
CORBA  
Java RMI  
XML RPC, SOAP/.NET, AJAX

---

---

## Sun RPC

---

---

### Sun RPC

RPC for Unix System V, Linux, BSD

- Also known as ONC RPC
  - (Open Network Computing)

Interfaces defined in an Interface Definition Language (IDL)

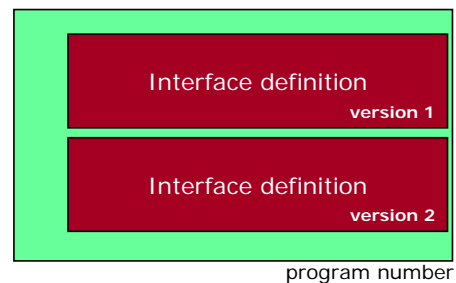
- IDL compiler is **rpcgen**

---

Paul Krzyzanowski • Distributed Systems

---

### RPC IDL



---

Paul Krzyzanowski • Distributed Systems

---

### RPC IDL

name.x

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

---

Paul Krzyzanowski • Distributed Systems

---

### rpcgen

rpcgen name.x

produces:

- name.h header
- name\_svc.c server stub (skeleton)
- name\_clnt.c client stub
- [ name\_xdr.c ] 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)

---

Paul Krzyzanowski • Distributed Systems

## 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 **portmapper (rpcbind** on SVR4):
    - Name server
    - Keeps track of {program#,version#,protocol}→port# bindings
- Server then listens and waits to accept connections

Paul Krzyzanowski • Distributed Systems

## 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

Paul Krzyzanowski • Distributed Systems

## Advantages

- Don't worry about getting a unique transport address (port)
  - But with SUN RPC 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
- Function call model can be used instead of send/receive

Paul Krzyzanowski • Distributed Systems

## 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, ...
- Room for improvement in Sun RPC

Paul Krzyzanowski • Distributed Systems

## DCE RPC

- Similar to Sun's RPC
- Interfaces written in a language called **Interface Definition Notation (IDN)**
  - Definitions look like function prototypes
- Unique ID generator
- 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

Paul Krzyzanowski • Distributed Systems

## Unique IDs

Sun 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)

Paul Krzyzanowski • Distributed Systems

## IDN compiler

Similar to rpcgen:

Generates header, client, and server stubs

Paul Krzyzanowski • Distributed Systems

## 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

Paul Krzyzanowski • Distributed Systems

## Service registration

Sun RPC:

- Server registers program# → port# on local name server (*rpcbind*)

DCE RPC:

Server registers:

**program name (UUID) → port**  
binding using local name server

DCE host daemon: *dced*

and

**program name (UUID) → machine**  
binding and protocols supported with cell directory server

Paul Krzyzanowski • Distributed Systems

## Service discovery

To locate a service:

- Contact cell directory server and locate machine
- Connect to *dced* (endpoint mapper) on machine hosting service and get port binding from local server

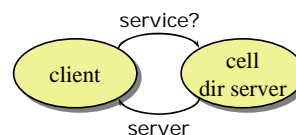
Directory has support for more complex searches that span cells

**Advantage:** services can be relocated

**Disadvantage:** greater overhead

Paul Krzyzanowski • Distributed Systems

## DCE service lookup

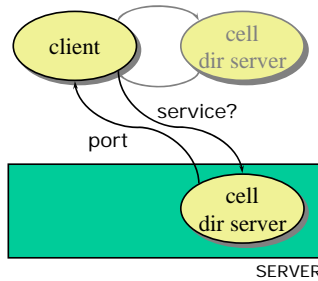


Request service lookup from cell directory server

Return server machine name

Paul Krzyzanowski • Distributed Systems

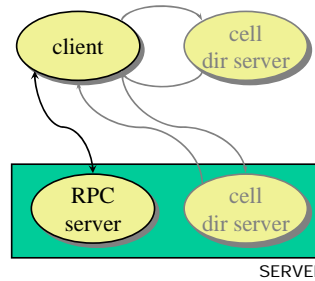
## DCE service lookup



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

Paul Krzyzanowski • Distributed Systems

## DCE service lookup



Connect to service and request remote procedure execution

Paul Krzyzanowski • Distributed Systems

## Marshaling

Standard formats for data

- NDR: Network Data Representation

Goal

- Sender can (hopefully) use native format
- Receiver may have to convert

Paul Krzyzanowski • Distributed Systems

## 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

Paul Krzyzanowski • Distributed Systems

## The next generation of RPCs

Support for object oriented languages

## Microsoft DCOM

## Microsoft DCOM

---

OLE/COM →

DCOM: Windows NT 4.0, fall 1996

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

---

Paul Krzyzanowski • Distributed Systems

## (D)COM activation mechanisms

---

- Establish connections to existing components
- Create new instances of components
- Several methods exist to create a COM object
  - e.g., *CoGetInstanceFromFile*
- Loading a COM library:
  - Look up appropriate binary code (DLL or executable) in system registry
  - Create object (load & dynamic link)
  - Return interface pointer to caller
- COM objects are named with *globally unique identifiers* (GUIDs)
- Classes of objects are identified with *class IDs*

---

Paul Krzyzanowski • Distributed Systems

## DCOM activation mechanisms

---

DCOM needs:

- Network name of server
- Class ID

Network name

- Fixed in registry (or DCOM class store)
  - Application unaware that this is a remote object
- or explicit parameter

---

Paul Krzyzanowski • Distributed Systems

## Activation on server

---

**Service Control Manager**  
(SCM, part of COM library)

- Connects to server SCM
- Requests creation of object on server

**Surrogate process** runs components

- Loads components and runs them

Can handle multiple clients simultaneously

---

Paul Krzyzanowski • Distributed Systems

## Beneath DCOM

---

Data transfer and function invocation

- Object RPC (**ORPC**)
- Extension of the **DCE RPC** protocol
- Standard DCE RPC packets plus:
  - **Interface pointer identifier** (IPID)
    - Identifies interface and object where the call will be processed
    - Referrals: can pass remote object references
  - Versioning information
  - Extensibility information

---

Paul Krzyzanowski • Distributed Systems

## Marshaling

---

- Marshaling mechanism: **NDR** (Network Data Representation) of DCE RPC
  - Byte order and floating point representation preserved across machine architectures
  - One new data type: represents a marshaled interface
- For marshaling, COM needs to know methods, parameters, structures
  - Obtained from Interface Definition Language (**MIDL**)
    - DCE IDL + object definitions

---

Paul Krzyzanowski • Distributed Systems

## MIDL

MIDL files are compiled with an IDL compiler

Generates C++ code for marshaling and unmarshaling

- 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*

Paul Krzyzanowski • Distributed Systems

## Remote reference lifetime

Object lifetime controlled by **remote reference counting**

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

Paul Krzyzanowski • Distributed Systems

## Cleanup

Abnormal client termination

- No message to decrement reference count on server

### 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

Paul Krzyzanowski • Distributed Systems

## Microsoft DCOM improvements

- Fits into Microsoft COM
- Generic server hosts dynamically loaded objects
  - Requires unloading objects (dealing with dead clients)
  - Reference counting and pinging
- Support for references to instantiated objects
- But... DCOM is a Microsoft-only solution
  - Doesn't work across firewalls

Paul Krzyzanowski • Distributed Systems

## CORBA

## CORBA

### Common Object Request Architecture

- Evolving since 1989

Standard architecture for distributing objects

Defined by OMG (Object Management Group)

- Consortium of > 700 companies

Goal: provide support for distributed, heterogeneous object-oriented applications

- Specification is independent of any language, OS, network

Paul Krzyzanowski • Distributed Systems

## CORBA

### Basic paradigm:

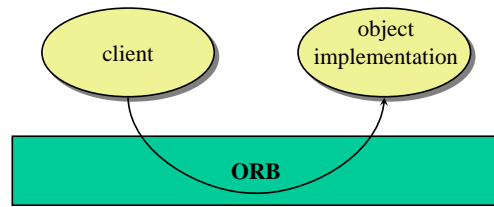
- Request services of a distributed object
- Interfaces are defined in an IDL
- Distributed objects are identified by object reference

### Object Request Broker (ORB)

- delivers request to the object and returns results to the client

Paul Krzyzanowski • Distributed Systems

## CORBA logical view



Paul Krzyzanowski • Distributed Systems

## IDL (Interface Definition Language)

- Indicates operations an object supports
  - Not *how* they are implemented
- Programming language neutral
  - Currently standardized language bindings for C, C++, Java, Ada, COBOL, Smalltalk, Objective C, LISP, Python
- IDL data types
  - Basic types: long, short, string, float, ...
  - Constructed types: struct, union, enum, sequence
  - Typed object references
  - The **any** type: a dynamically typed value

Paul Krzyzanowski • Distributed Systems

## IDL example

```
Module StudentObject {
    struct StudentInfo {
        string name;
        int id;
        float gpa;
    };
    exception Unknown {};
    interface Student {
        StudentInfo getinfo(in string name)
            raises(Unknown);
        void putinfo(in StudentInfo data);
    };
};
```

Paul Krzyzanowski • Distributed Systems

## CORBA IDL

- Compiled with IDL compiler
- Converted to target language
  - Generates stub functions

Paul Krzyzanowski • Distributed Systems

## Object Request Broker (ORB)

### Distributed service that implements the request to the remote object

- Locates the remote object on the network
- Communicates request to the object
- Waits for results
- Communicates results back to the client

### Responsible for providing location transparency

- Same request mechanism used by client & CORBA object regardless of object location

### Client request may be written in a different programming language than the implementation

Paul Krzyzanowski • Distributed Systems

## ORB functions

- Look up and instantiate objects on remote machines
- Marshal parameters
- Deal with security issues
- Publish data on objects for other ORBs to use
- Invoke methods on remote objects
  - Static or dynamic execution
- Automatically instantiate objects that aren't running
- Route callback methods
- Communicate with other ORBs

Paul Krzyzanowski • Distributed Systems

## CORBA Programming

Programming is done via object references and requests

- Clients issue a request on a CORBA object using the **object reference**

```
Student st = ...
try {
    StudentInfo sinfo = st.getinfo("Amy Smith");
} catch (Throwable e) {
    . . .
}
```

Paul Krzyzanowski • Distributed Systems

## Objects

- Object references persist
  - They can be saved as a string
  - ... and be recreated from a string
- Client
  - Performs requests by having an **object reference** for object & desired operation
  - Client initiates request by
    - calling **stub** routines specific to an object
    - Or constructing request dynamically (**DII** interface)
- Server (object implementation)
  - Provides semantics of objects
  - Defines data for instance, code for methods

Paul Krzyzanowski • Distributed Systems

## Interoperability

- CORBA clients are portable
  - They conform to the API
- Object implementations (servers)
  - generally need some rework to move from one vendor's CORBA product to another
- 1996: CORBA 2.0 added **interoperability** as a goal in the specification
  - Define network protocol called **IIOP**
    - Inter-ORB Protocol
  - IIOP works across any TCP/IP implementations

Paul Krzyzanowski • Distributed Systems

## IIOP

IIOP can be used in systems that do not even provide a CORBA API

- Used as transport for version of Java RMI (RMI over IIOP)
- Various application servers use IIOP but do not expose the CORBA API
- Programs written to different APIs can interoperate with each other and with programs written to the CORBA API

Paul Krzyzanowski • Distributed Systems

## Internet services

"Netscape and its partners want to make CORBA and the IIOP as ubiquitous as HTML and HTTP, making Internet-based services as widely available as Internet content is today".

– *developer.netscape.com*  
Fall 1996

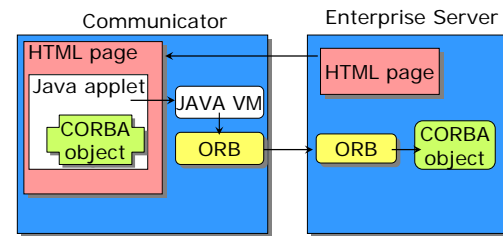
Paul Krzyzanowski • Distributed Systems

## Netscape – an IOP adopter

- Netscape Communicator ships with the Netscape Internet Service Broker (ISB) for Java
  - Possible for clients to use services of CORBA
- Netscape Enterprise Server 3.x includes
  - IDL compilers, network agents, other utilities
- Netscape's Visual JavaScript
  - Can integrate CORBA-compliant objects into an HTML page. Users can work with these objects interactively just as they do with any other Java Bean components

Paul Krzyzanowski • Distributed Systems

## Netscape – an IOP adopter



Paul Krzyzanowski • Distributed Systems

## CORBA Services (COS)

Set of distributed services to support the integration and interoperation of distributed objects

- Defined on top of ORB
  - Standard CORBA objects with IDL interfaces

Paul Krzyzanowski • Distributed Systems

## Popular services

- Object life cycle
  - Defines how CORBA objects are created, moved, removed, copied
- Naming
  - Defines how objects can have friendly symbolic names
- Events
  - Asynchronous communication
- Externalization
  - Coordinates the transformation of objects to/from external media

Paul Krzyzanowski • Distributed Systems

## Popular services

- Transactions
  - Provides atomic access to objects
- Concurrency control
  - Locking service for serializable access
- Property
  - Manage name-value pair namespace
- Trader
  - Find objects based on properties and describing service offered by object
- Query
  - Queries on objects

Paul Krzyzanowski • Distributed Systems

## CORBA vendors

Lots of vendors

- ORBit
  - Bindings for C, Perl, C++, Lisp, Pascal, Python, Ruby, and TCL (designed for GNOME)
- Java ORB
  - Part of Java SDK
- VisiBroker for Java
  - From Imprise; embedded in Netscape Communicator
- OrbixWeb
  - From Iona Technologies
- Websphere
  - From IBM
- Many, many others

Paul Krzyzanowski • Distributed Systems

## Assessment

---

- Reliable, comprehensive support for managing services
- Standardized
- Complex
  - Steep learning curve
  - Integration with languages not always straightforward
- Pools of adoption
- Late to ride the Internet bandwagon

Paul Krzyzanowski • Distributed Systems

## Java RMI

---

## Java RMI

---

- Java language had no mechanism for invoking remote methods
  - At inception, Java supported downloading of code to a remote site – applets
  - Only support for distributed communication was sockets
- 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

Paul Krzyzanowski • Distributed Systems

## RMI components

---

### Client

- Invokes method on remote object

### Server

- Process that owns the remote object

### Object registry

- Name server that relates objects with names
- Objects are registered with object registry
- Once registered, registry can be used to access a remote object using its name

Paul Krzyzanowski • Distributed Systems

## 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**

Paul Krzyzanowski • Distributed Systems

## Design goals

---

- Fit the language
- Easy to use
- Seamless invocation of objects
- Support callbacks from servers to objects
- Preserve safety of Java environment
- Support distributed garbage collection
- Support multiple transports

Paul Krzyzanowski • Distributed Systems

## RMI similarities

Similar to local objects

- References to objects can be passed as parameters (not really)
- Object can be cast to any of the set of interfaces supported by the implementation

Paul Krzyzanowski • Distributed Systems

## RMI differences

- Non-remote arguments/results passed to/from a remote method by copy
- Remote object passed by reference, not by copying remote implementation
- Extra exceptions

Paul Krzyzanowski • Distributed Systems

## New classes

### • remote class:

- One whose instances can be used remotely
- Within its address space: regular object
- Other address spaces: can be referenced with an **object handle**

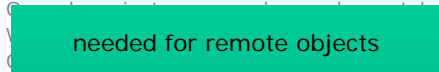

### • serializable class:

- 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

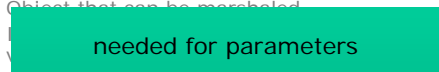

Paul Krzyzanowski • Distributed Systems

## New classes

### • remote class:

-  needed for remote objects
-  with an **object handle**

### • serializable class:

-  needed for parameters
-  to another
  - If remote object is passed, only the object handle is copied between address spaces

Paul Krzyzanowski • Distributed Systems

## Serializable

To make an object serializable

- Implement `java.io.Serializable` interface

Paul Krzyzanowski • Distributed Systems

## Remote

To make an object remote

- Remote interface must
  - Be public
  - Extend `java.rmi.Remote`
  - Every method in the interface must declare that it throws `java.rmi.RemoteException`
- Remote class must:
  - Implement a remote Interface
  - Extend the `java.rmi.server.UnicastRemoteObject` class
  - Can have methods that are not in the remote interface
    - These can only be invoked locally

Paul Krzyzanowski • Distributed Systems

## Stubs

Generated by separate compiler

### rmic

- Remote interfaces and classes compiled with javac
- Stubs and skeletons for the remote interfaces are generated (class files) with rmic stub compiler

Paul Krzyzanowski • Distributed Systems

## Naming service

Need a remote object reference to perform remote object invocations

- Object registry does this: **rmiregistry**

Name of a remote object includes:

- Internet name of machine running Object Registry with which the object is registered
  - Default: localhost
- Port on which object registry is listening
  - Default: 1099
- Name of the object
  - Java uses a URL naming scheme

Paul Krzyzanowski • Distributed Systems

## Server

Create and install a security manager

```
System.setSecurityManager(  
    new RMISecurityManager());
```

Register object(s) with Object Registry

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

Paul Krzyzanowski • Distributed Systems

## Client

Contact *rmiregistry* to lookup name

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

**rmiregistry** returns a remote object reference.

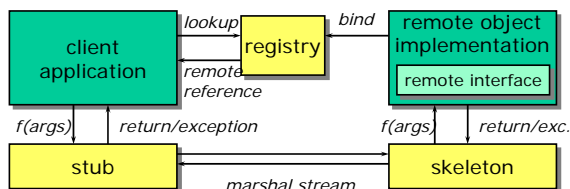
**lookup** replaces object reference to reference to local stub.

Invoke remote method(s):

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

Paul Krzyzanowski • Distributed Systems

## Java RMI infrastructure



Paul Krzyzanowski • Distributed Systems

## RMI Distributed Garbage Collection

- Two operations: *dirty* and *free*

- 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

Paul Krzyzanowski • Distributed Systems

---

## The third generation of RPCs

Web services  
and  
Riding the XML Bandwagon



---

### We began to want

Remotely hosted services

### Problem

Firewalls:  
Restrict ports  
Inspect protocol

### Solution

Proxy procedure calls over HTTP

---

Paul Krzyzanowski • Distributed Systems

---

## XML RPC

---

---

### Origins

- 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 true IDL compiler support (yet)
    - Lots of support libraries
- 

Paul Krzyzanowski • Distributed Systems

---

### 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>
```

---

Paul Krzyzanowski • Distributed Systems

---

### XML-RPC data types

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

Paul Krzyzanowski • Distributed Systems

## Assessment

- Simple (spec about 7 pages)
- Humble goals
- Good language support
  - Little/no function call transparency
- Little/no industry support
  - Mostly grassroots

Paul Krzyzanowski • Distributed Systems

# SOAP

## SOAP origins

### (Simple) Object Access Protocol

- 1998 and evolving
- 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
  - other features
- XML (usually) over HTTP

Paul Krzyzanowski • Distributed Systems

## Web Services and WSDL

Web Services Description Language

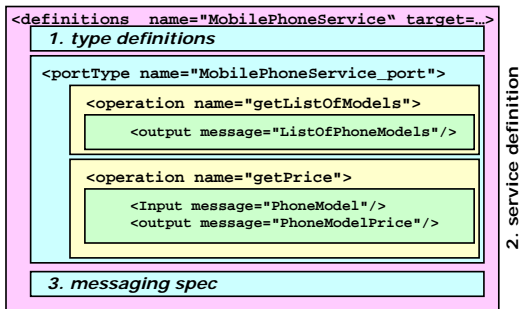
- Analogous to an IDL

Describe an organization's web services

- Businesses will exchange WSDL documents

Paul Krzyzanowski • Distributed Systems

## WSDL structure



Paul Krzyzanowski • Distributed Systems

## 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>
```

Paul Krzyzanowski • Distributed Systems

---

## Microsoft .NET Web Services

---

### 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++
- 

Paul Krzyzanowski • Distributed Systems

### Microsoft .NET

---

Microsoft's Internet strategy

- Not an OS
  - Delivers software as web services
  - Framework for universally accessible services
  - Server-centric computing model
- 

Paul Krzyzanowski • Distributed Systems

### Components

---

- New object runtime environment
  - Prefabricated web functionality
    - Web services
  - Windows Forms
  - Visual Studio .NET
    - Make it easy to program .NET-compliant programs and build web services
- 

Paul Krzyzanowski • Distributed Systems

### New object runtime environment

---

#### Common Language Runtime (CLR)

- Services compile to IL bytecodes
  - Language neutral
    - C++, C#, VB, Jscript + 3<sup>rd</sup> party support
  - Common class libraries
    - ADO.NET, ASP.NET, Windows Forms
  - CLR implements common features
    - Lifetime management
    - Security
    - Garbage collection
    - versioning
- 

Paul Krzyzanowski • Distributed Systems

### Web functionality

---

#### ASP.NET

- Evolution of existing ASP product
- New smart controls in web pages

#### .NET Web Services

- Function-based way to expose software functionality to clients
- 

Paul Krzyzanowski • Distributed Systems

## .NET Web Services

Based on:

- HTTP** – communications protocol  
*HyperText Transfer Protocol*
- XML** – data format  
*eXtended Markup Language*
- SOAP** – format for requesting services  
*Simple Object Access Protocol*
- WSDL** – format for defining services  
*Web Services Definition Language*
- UDDI** – protocol for discovering services  
*Universal Description, Discovery, & Integration*

Paul Krzyzanowski • Distributed Systems

## .NET web services vs. SOAP

- SOAP is lower-level protocol
- Web Services provides higher level of abstraction
- Write .NET object as if it were accessed by local clients
- Mark it with attribute that it should be available to Web clients
- ASP.NET does the rest
  - Hooks up an infrastructure that accepts HTTP requests and maps them to object calls
- Service description in WSDL
  - Automatically generated by examining metadata in .NET object

Paul Krzyzanowski • Distributed Systems

## .NET operating environment

Services compile to **Intermediate Language (IL)** bytecodes

Compiled into executable code by **Common Language Runtime (CLR)**

Paul Krzyzanowski • Distributed Systems

## Common Language Runtime

Implementation of common features:

- lifetime management
- garbage collection
- security
- versioning

When first loaded (prior to running):

- CLR runs just-in-time compiler to generate native code
- Never interpreted

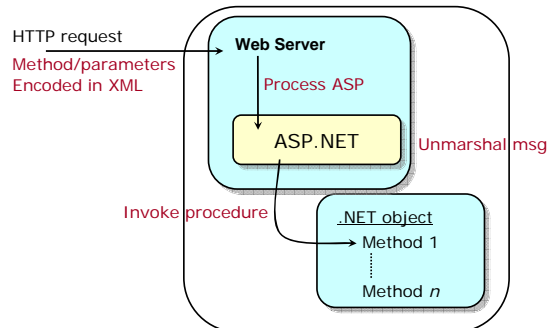
Paul Krzyzanowski • Distributed Systems

## .NET framework

- .NET framework available to any language that wants to produce IL
- .NET objects can register themselves to be used as COM objects by COM clients
- .NET promises:
  - Function-based way to expose software functionality on one machine to programs running on another (.NET Web Services)

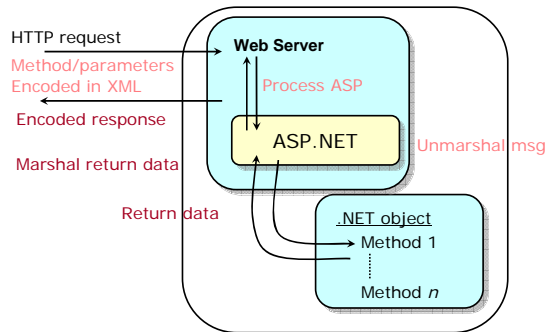
Paul Krzyzanowski • Distributed Systems

## Web Service invocation



Paul Krzyzanowski • Distributed Systems

## Web Service invocation



Paul Krzyzanowski • Distributed Systems

## Some .NET services

- data access
  - built on NTFS, SQL server, Exchange
- authentication
  - with Microsoft Passport and Windows Authentication
- Internet calendar
  - based on Outlook and Hotmail
- Messaging
  - Hotmail, Exchange, Instant Messenger
- Directory services

Paul Krzyzanowski • Distributed Systems

## More Web Integration

## AJAX

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

Paul Krzyzanowski • Distributed Systems

## AJAX & XMLHttpRequest

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

```
– IE:    new ActiveXObject("msxml2.XMLHTTP")
– Mozilla/Safari: new XMLHttpRequest()
    xmlhttp.open("HEAD", "index.html", true)
```

- 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

Paul Krzyzanowski • Distributed Systems

## Sample web services

### Google Web APIs Developer Kit

[www.google.com/apis/download.html](http://www.google.com/apis/download.html)

- A WSDL file you can use with any development platform that supports web services.
- A Java library that provides a wrapper around the Google Web APIs SOAP interface.
- An example .NET program which invokes the Google Web APIs service.
- Documentation that describes the SOAP API and the Java library.

Paul Krzyzanowski • Distributed Systems

## Not-really-a-web-service

REST: REpresentational State Transfer

- Stay with the principles of the web
  - Four HTTP commands let you operate on data (a resource):
    - PUT (insert)
    - GET (select)
    - POST (update)
    - DELETE (delete)
- In contrast to invoking operations on an activity.
- Message includes representation of data.

Paul Krzyzanowski • Distributed Systems

## 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}
- Makes sense for resource-oriented services
  - Bloglines, Amazon, flickr, del.icio.us, ...

Paul Krzyzanowski • Distributed Systems

## Resource-oriented services

- Get parts info
  - HTTP GET //www.parts-depot.com/parts
- Returns a document containing a list of parts (implementation transparent to clients)

```
<?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>
```

Paul Krzyzanowski • Distributed Systems

## Resource-oriented services

- Get detailed parts info:
  - HTTP GET //www.parts-depot.com/parts/00345
- Returns a document containing a list of parts (implementation transparent to clients)

```
?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>
```

Paul Krzyzanowski • Distributed Systems

## Summary

## ONC RPC, DCE

### RPC/DCE

- Language/OS independent (mostly UNIX, some Windows)
- No polymorphism
- No dynamic invocation

### DCE RPC added:

- UUID
- layer of abstraction: a cell of machines

Paul Krzyzanowski • Distributed Systems

## Microsoft DCOM/ORPC

---

- ORPC: slight extension of DCE RPC
- Single server with dynamic loading of objects (surrogate process)
- Platform dependent – generally a Microsoft-only solution
- Support for moderate distributed garbage collection
  - Clients pings server to keep references valid

---

Paul Krzyzanowski • Distributed Systems

## Java RMI

---

- Language dependent
  - Java
- Architecture dependent
  - JVM
- Generalized (and programmable) support for object serialization
- No dynamic invocation
- No support for dynamic object/interface discovery

---

Paul Krzyzanowski • Distributed Systems

## CORBA

---

- Language/OS independent
  - Widespread support
- Support for object-oriented languages
- Dynamic discovery and invocation
- Object life-cycle management
  - Persistence
  - Transactions
  - Metering
  - Starting services

---

Paul Krzyzanowski • Distributed Systems

## XML-RPC/SOAP/.NET

---

- XML over HTTP transport
  - Relatively easy to support even if language does not have a compiler (or precompiler)
  - WSDL – service description
  - Proxy over HTTP/port 80
    - Bypass firewalls
  - SOAP has gotten bloated; large messages
- .NET Web Services introduces
  - Language support for deploying web services (you don't have to deal with SOAP)
  - Library support, including predefined services

---

Paul Krzyzanowski • Distributed Systems

## AJAX, REST

---

- AJAX
  - Designed for web client-server interaction
  - Simple JavaScript calling structure using XMLHttpRequest class
  - You can encapsulate SOAP requests or whatever...
- REST
  - Sticks to basic principles of HTTP.
  - Posits that you don't need additional communication streams or method-like abstractions of SOAP or RMI

---

Paul Krzyzanowski • Distributed Systems

---

**The end.**

---