Distributed Systems

RPC Case Studies

Paul Krzyzanowski
pxk@cs.rutgers.edu
ds@pk.org

Overview of RPC Systems

Sun RPC

DCE RPC

DCOM

CORBA

Java RMI

XML RPC, SOAP/.NET, AJAX, REST

Sun RPC

Sun RPC

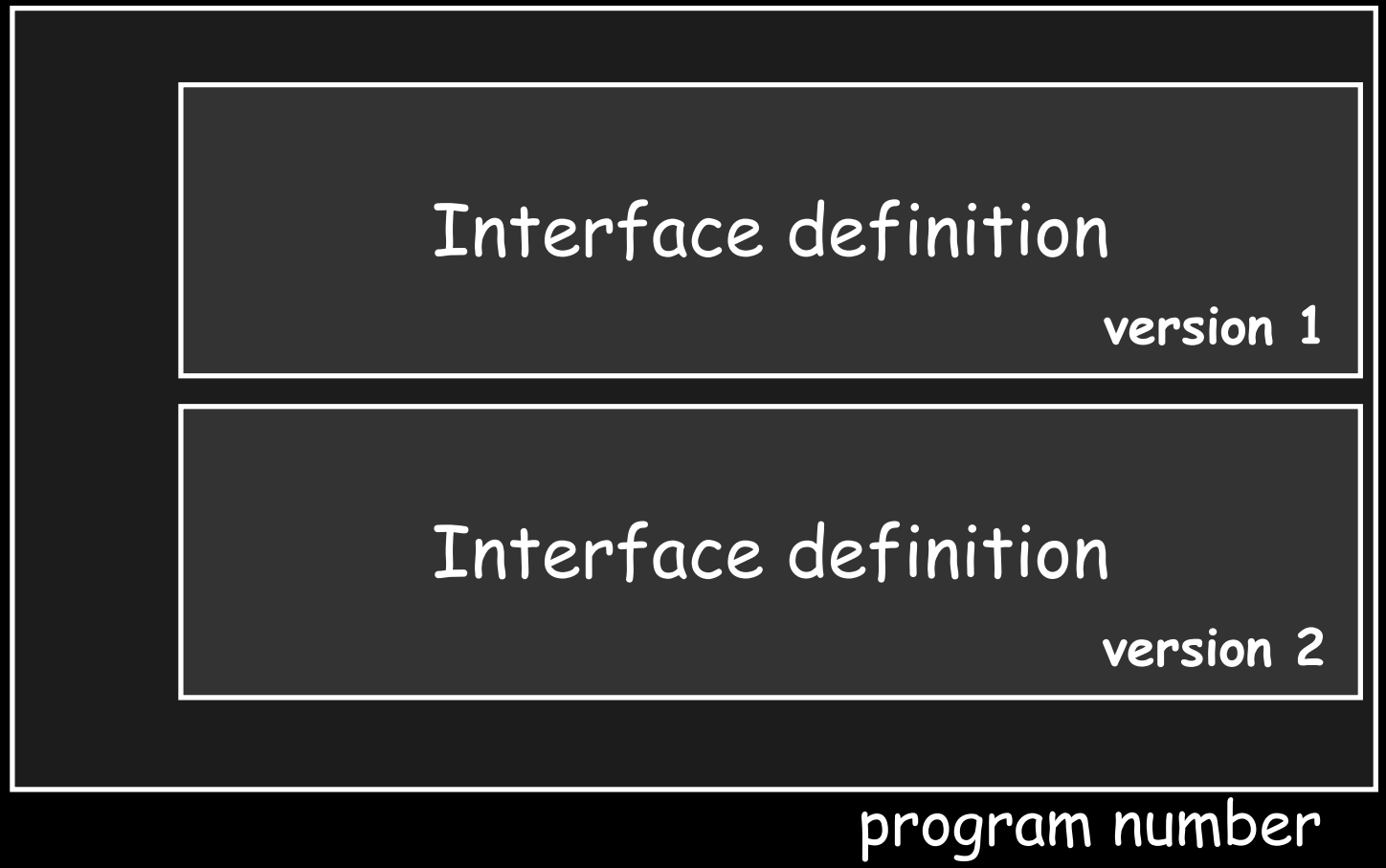
RPC for Unix System V, Linux, BSD, OS X

- Also known as ONC RPC (Open Network Computing)

Interfaces defined in an Interface Definition Language (IDL)

- IDL compiler is **rpcgen**

RPC IDL


Interface definition
version 1
Interface definition
version 2
program number

RPC IDL

name.x

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

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)

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

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

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

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

DCE RPC

- Similar to Sun's RPC
- Interfaces written in a language called **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

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)

IDN compiler

Similar to rpcgen:

Generates header, client, and server stubs

Service lookup

Sun RPC requires client to know name of server

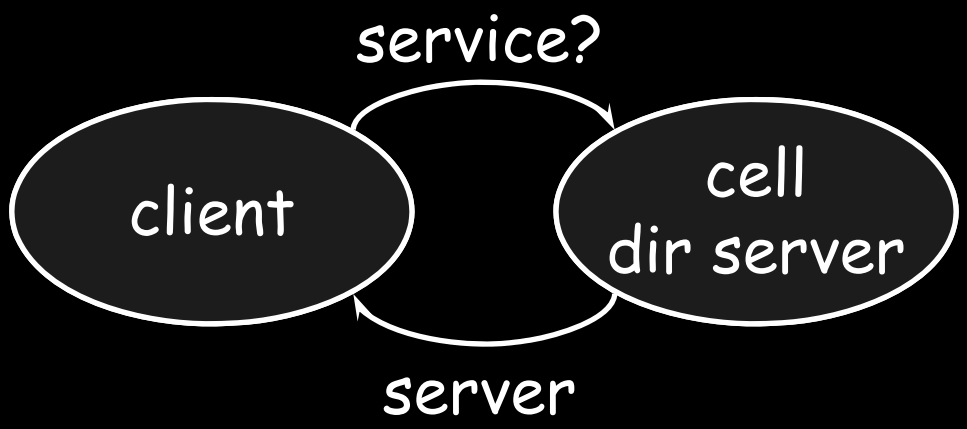
DCE allows several machines to be organized into an administrative entity

- **cell** (collection of machines, files, users)

Cell directory server

- Each machine communicates with it for cell services information

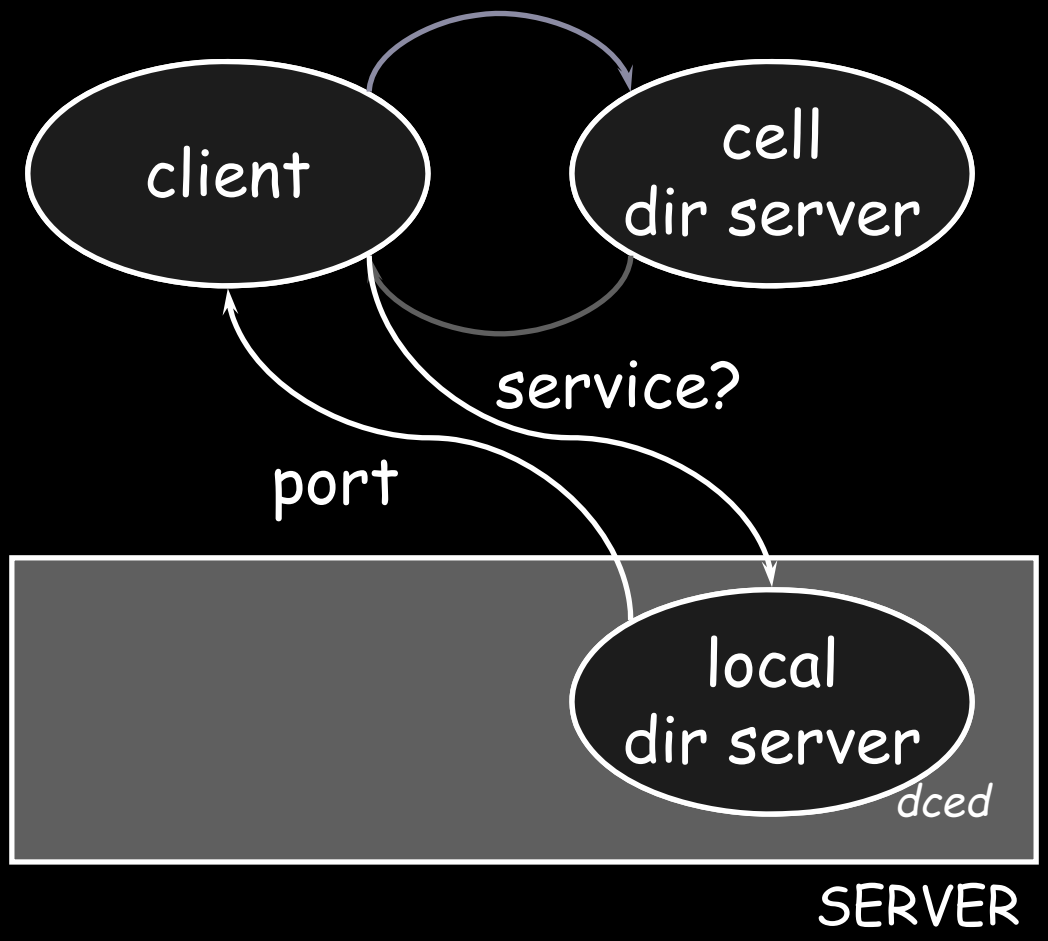
DCE service lookup



Request service lookup from cell directory server

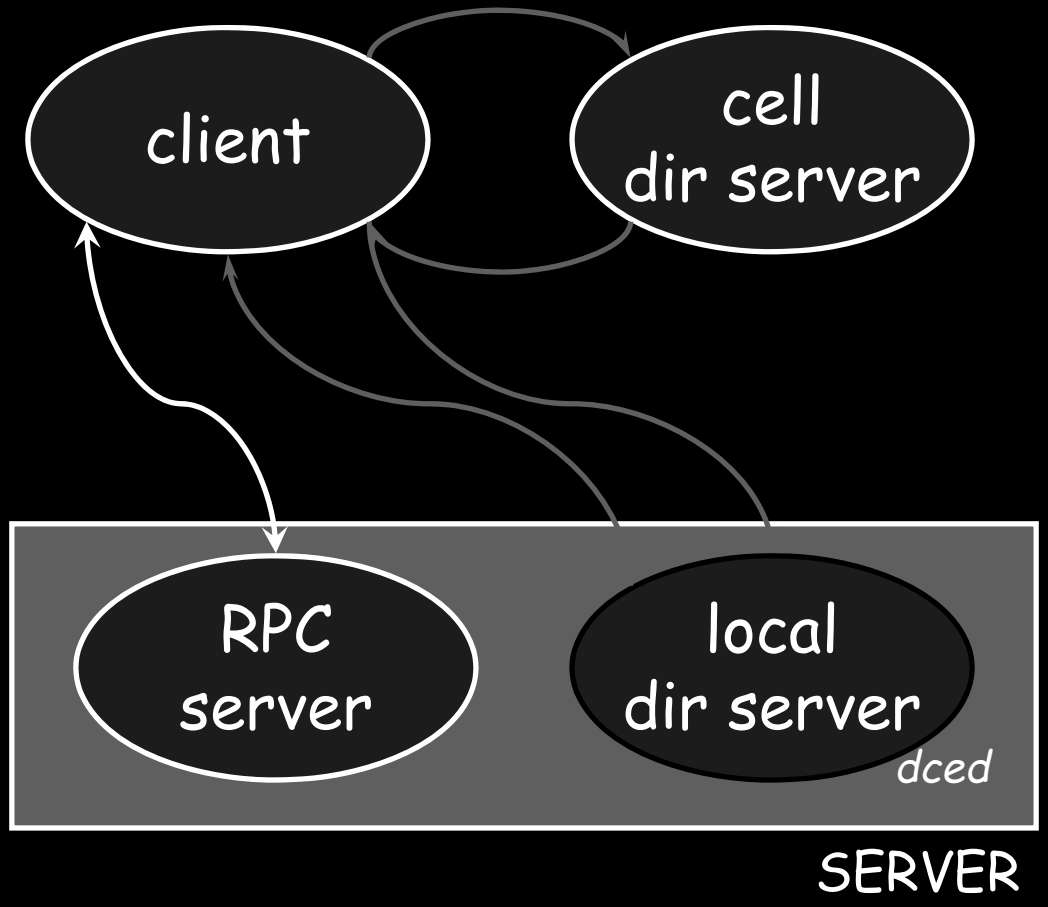
Return server machine name

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

Marshaling

Standard formats for data

- NDR: Network Data Representation

Goal

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

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

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

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

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 & extensibility information

Marshaling

- Marshaling mechanism: **NDR** (Network Data Representation) of DCE RPC
  - One new data type: represents a marshaled interface

MIDL

MIDL files are compiled with an IDL compiler

DCE IDL + object definitions

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

Remote reference lifetime

Object lifetime controlled by **remote reference counting**

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

Cleanup

Abnormal client termination

- No message to decrement reference count set to 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

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 well across firewalls

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

CORBA

Basic paradigm:

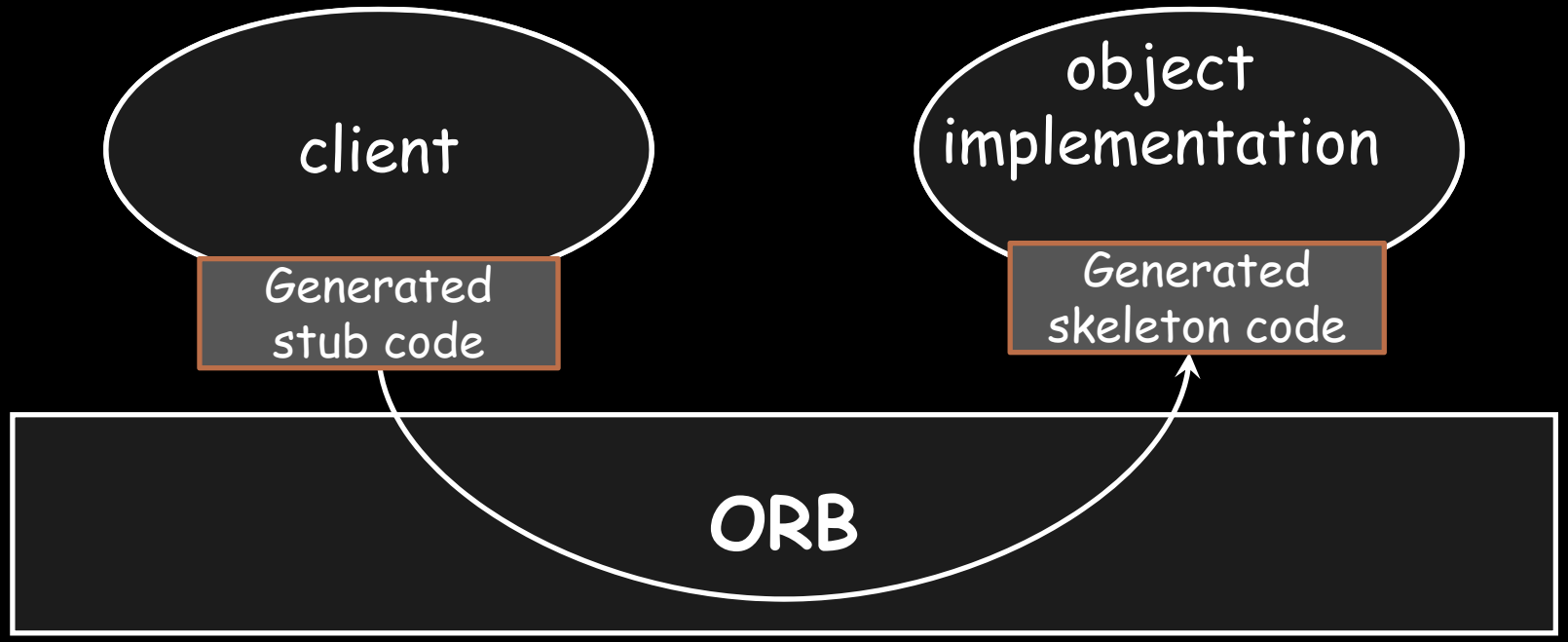
- Request services of a distributed object

- 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
- = set of code that implements RPC

CORBA logical view



IDL (Interface Definition Language)

- Indicates operations an object supports
  - Not *how* they are implemented

- Programming language neutral
  - Currrently 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

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);
    };
};
```

CORBA IDL

Compiled with IDL compiler

- Converted to target language
- Generates stub functions

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

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

Objects

- Object references persist
  - They can be saved as as 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

Interoperability

- CORBA clients are portable
  - They conform to the API … but may need recompilation
- 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

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

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

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

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

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

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

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 really)
- Objects can be passed as parameters to remote methods (but not as a reference)
- Object can be cast to any of the set of interfaces supported by the implementation
  - Operations can be invoked on these objects

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

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

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

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

Stubs

Generated by separate compiler

rmic

- Produces Stubs and skeletons for the remote interfaces are generated (class files)

Naming service

Need a remote object reference to perform remote object invocations

Object registry does this: **rmiregistry**

Server

Register object(s) with Object Registry

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

Client

Contact *rmiregistry* to lookup name

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

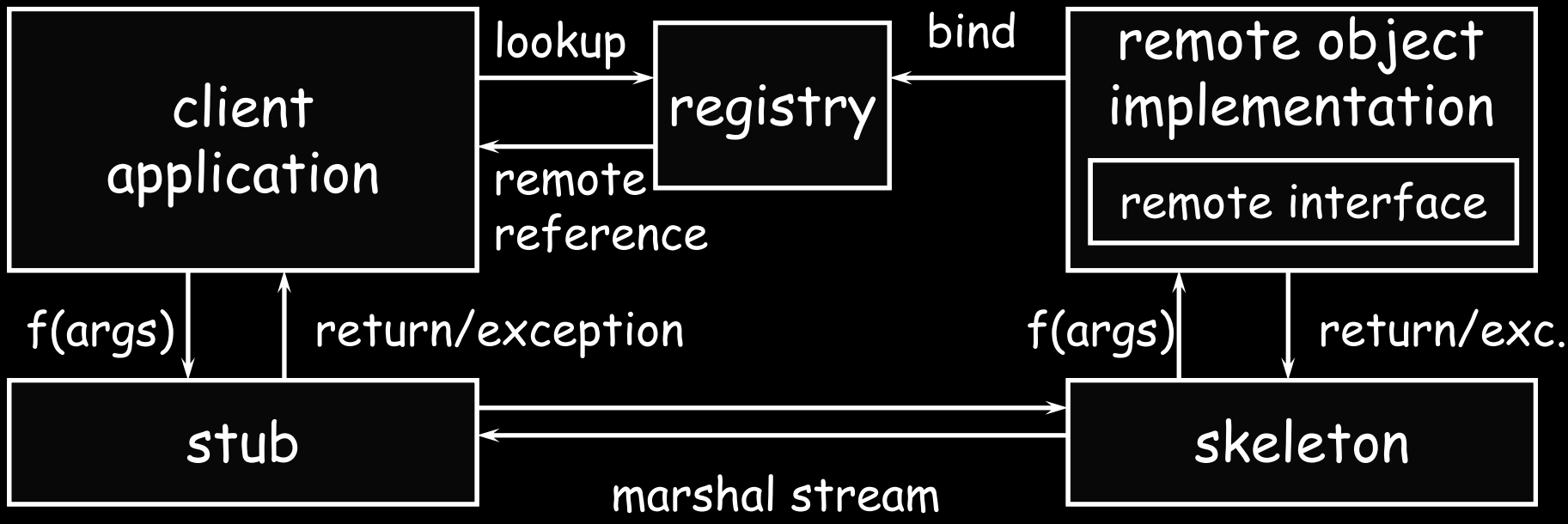
rmiregistry returns a remote object reference.

lookup gives reference to local stub.

Invoke remote method(s):

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

Java RMI infrastructure



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

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

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

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
- Little/no industry support
  - Mostly grassroots

SOAP

SOAP origins

(Simple) Object Access Protocol

- 1998 and evolving (v1.2 Jan 2003)
- 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 …
- XML (usually) over HTTP

Web Services and WSDL

Web Services Description Language

- Analogous to an IDL

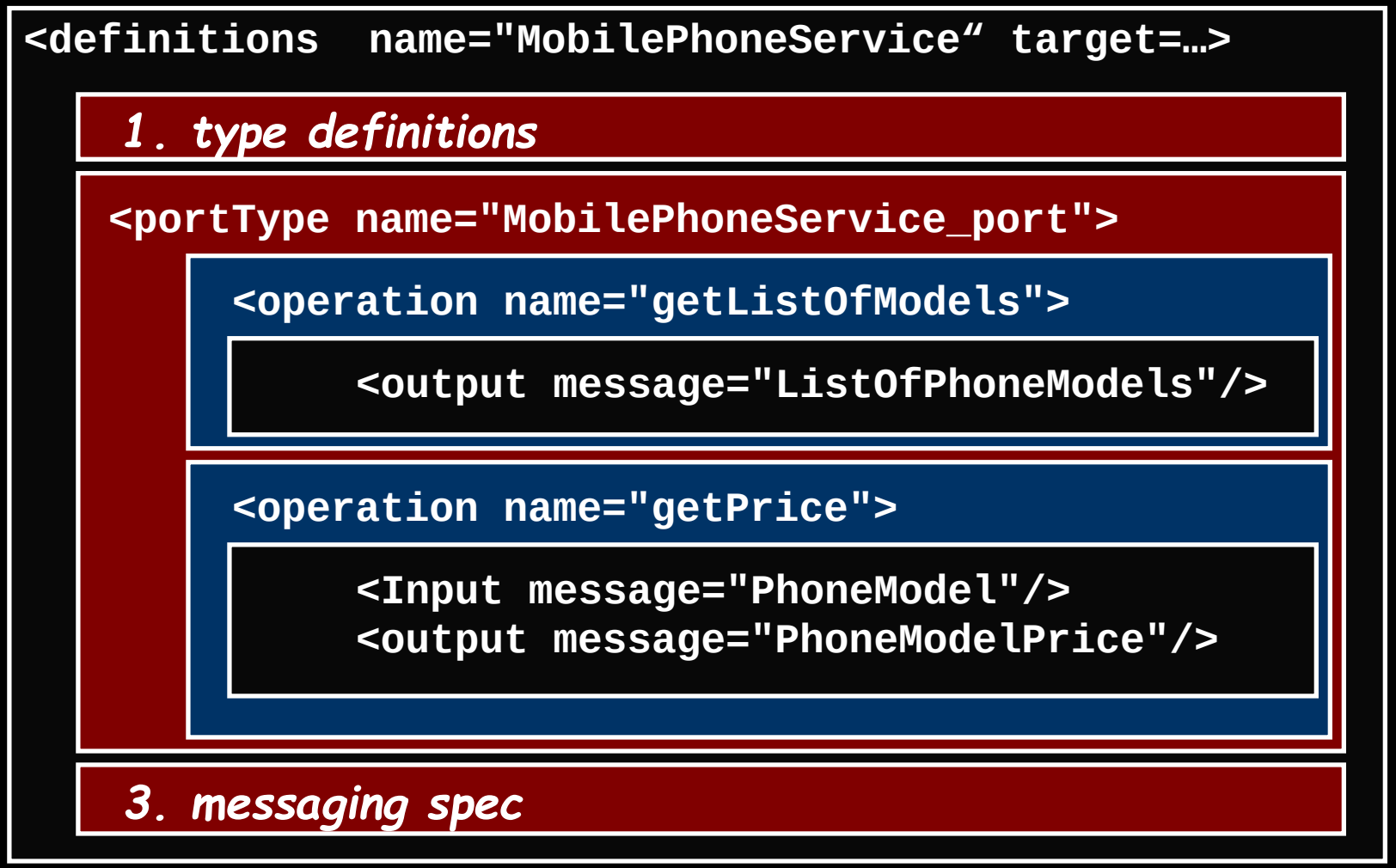
Describe an organization's web services

- Businesses will exchange WSDL documents

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



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>
```

Microsoft .NET Remoting

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++

Microsoft .NET

Microsoft's Internet strategy

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

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

New object runtime environment

Common Language Runtime (CLR)

- Services compile to Intermediate Language (IL)
- Language neutral
  - C++, C#, VB, Jscript + 3rd party support
- Common class libraries
  - ADO.NET, ASP.NET, Windows Forms

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

.NET Remoting

- Object interaction across application domains
- Invoke remote objects
  - Object derived from MarshalByRefObject
  - Proxy created when object is activated
  - CLR intercepts calls
  - The CLR is told which classes are remote so it can do the right thing when the client requests a *new* object
- Passing objects as parameters
  - Objects implement ISerializable interface

.NET Remoting

- Communication over channels
  - HTTP channel - XML/SOAP encoding
  - TCP channel - binary encoding
  - SMTP – XML/SOAP encoding

Object Lifetime

Single Call: new instance per call (stateless)

Singleton: same instance for all requests

Client Activated Objects:

Similar to DCOM (COM+)

Each time a method is called:

- Lease time set to max of current LeaseTime and RenewOnCallTime
- Requestor has to renew lease when LeaseTime elapses
- No more reference counting!

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

.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

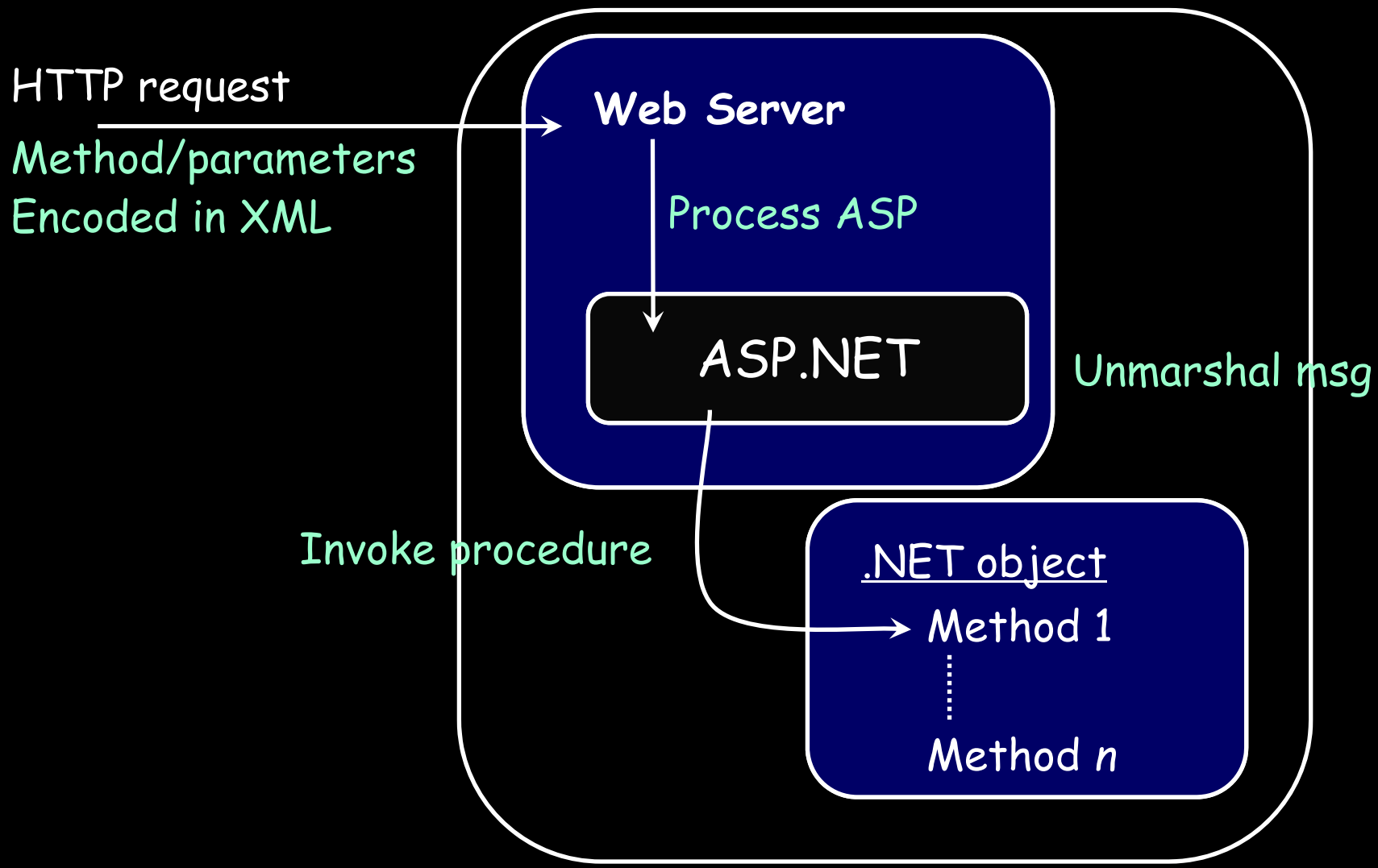
WSDL – format for defining services
Web Services Definition Language

UDDI – protocol for discovering services
Universal Description, Discovery, & Integration

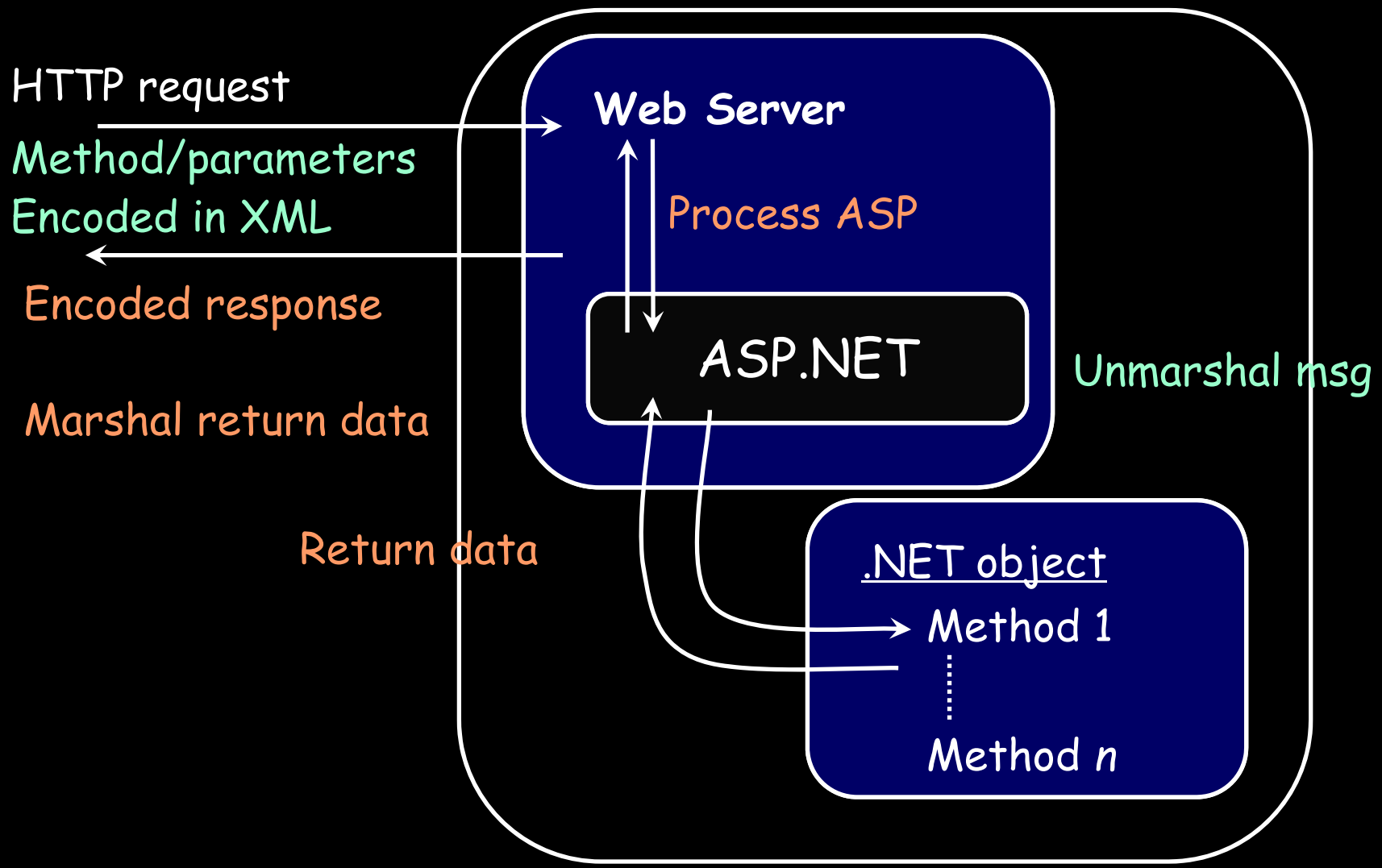
.NET Web Services vs. SOAP

- SOAP is lower-level messaging 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

Web Service invocation



Web Service invocation



Away from RPC…

More Web Services

Until 2006…

Google Web APIs Developer Kit - SOAP

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.

The future of SOAP?

- SOAP
  - Dropped by Google in 2006
  - Alternatives exist: AJAX, XML-RPC, REST, …
  - Allegedly complex because "we want our tools to read it, not people"
    - unnamed Microsoft employee
- Microsoft
  - SOAP APIs for Microsoft Live
  - http://search.live.com/developer

AJAX

- Asynchronous JavaScript And XML
- 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)
  - IE: `new ActiveXObject("msxml3.XMLHTTP")`
  - Mozilla/Opera/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

AJAX on the Web

- Google Maps, Amazon Zuggest, Del.icio.us Director, Writely, …
- Microsoft ASP.NET AJAX 1.0
  - January 2007
  - Integrate client script libraries with ASP.NET server-based code
- Google recommends use of their AJAX Search API instead of SOAP Search API

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.

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, flikr, del.icio.us, …

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>
```

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>
```

REST vs. RPC

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();
```

REST-based Systems

- Yahoo! Search APIs
- Ruby on Rails 1.2
- Open Zing Services – Sirius radio

svc://Radio/ChannelList
svc://Radio/ChannelInfo?sid=001-siriushits1&ts=2007091103205

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

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 distributed garbage collection
  - Clients pings server to keep references valid

Java RMI

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

CORBA

- Cross-platform: language/OS independent
  - Widespread support
- Support for object-oriented languages
- Dynamic discovery and invocation
- Object life-cycle management
  - Persistence
  - Transactions
  - Metering
  - Load balancing
  - Starting services

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 Remoting & Web Services introduces
  - Language support for deploying web services (you don't have to deal with SOAP)
  - Library support, including predefined services

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 the method-like abstractons of SOAP or RMI

The end.