

CS162 Operating Systems and Systems Programming Lecture 23

Remote Procedure Call

April 23, 2014
Anthony D. Joseph
<http://inst.eecs.berkeley.edu/~cs162>

Goals for Today

- Remote Procedure Call
- Examples using RPC
 - Distributed File Systems
 - World-Wide Web
- Modern RPC systems
 - SOAP
 - REST

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne, notes by Joseph and Kubiawicz.

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.2

Distributed Systems – Message Passing

- Distributed systems use a variety of messaging frameworks to communicate:
 - e.g. the protocols for TCP: connecting, flow control, loss...
 - 2PC for transaction processing
 - HTTP GET and POST
 - UDP messages for MS SQL Server (last time)
- Disadvantages of message passing:
 - Complex, stateful protocols, versions, feature creep
 - Need error recovery, data protection, etc.
 - Ad-hoc checks for message integrity
 - Resources consumed on server between messages (DoS risk)
 - Need to program for different OSes, target languages,...
- Want a higher-level abstraction that addresses these issues, but whose effects are application-specific

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.3

Remote Procedure Call

- Another option: Remote Procedure Call (RPC)
 - Looks like a local procedure call on client:

```
file.read(1024);
```
 - Translated automatically into a procedure call on remote machine (server)
- Implementation:
 - Uses request/response message passing “under the covers”
 - Deals with many of the generic challenges of protocols that use message passing – may even be “transactional” - but usually not.
 - Allows the programmer to focus on the message **effects**: as though the procedure were executed on the server.

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.4

RPC Details

- Client and server use “stubs” to glue pieces together
 - Client stub is responsible for “marshalling” arguments and “unmarshalling” the return values
 - Server-side stub is responsible for “unmarshalling” arguments and “marshalling” the return values
- Marshalling** involves (depending on system) converting values to a canonical form, serializing objects, copying arguments passed by reference, etc.
 - Needs to account for cross-language and cross-platform issues
- Technique: compiler generated stubs
 - Input: interface definition language (IDL)
 - Contains, among other things, types of arguments/return
 - Output: stub code in the appropriate source language

4/23/2014

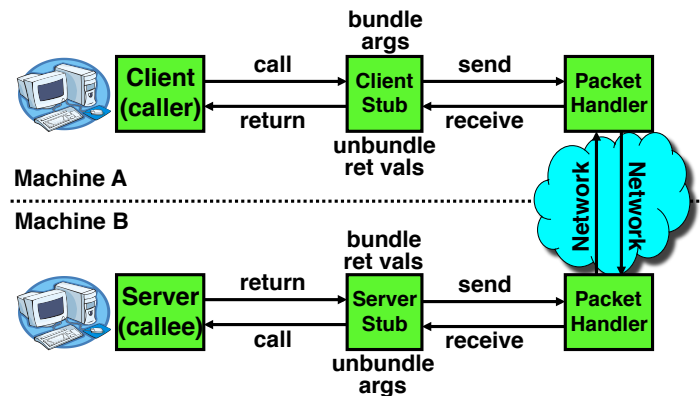
Anthony D. Joseph

CS162

©UCB Spring 2014

23.5

RPC Information Flow



4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.6

RPC Binding

- How does client know which machine to send RPC?
 - Need to translate name of remote service into network endpoint (e.g., host:port)
 - Binding:** the process of converting a user-visible name into a network endpoint
 - This is another word for “naming” at network level
 - Static: fixed at compile time
 - Dynamic: performed at runtime
- Dynamic Binding**
 - Most RPC systems use dynamic binding via name service
 - Why dynamic binding?
 - Access control: check who is permitted to access service
 - Fail-over: If server fails, use a different one
- Object registry (if used)**
 - Contains remote object names and client stub code
 - Allows dynamic loading of remote object stub

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.7

Cross-Domain Communication/Location Transparency

- How do address spaces communicate with one another?
 - Shared Memory with Semaphores, monitors, etc...
 - File System
 - Pipes (1-way communication)
 - “Remote” procedure call (2-way communication)
- RPC’s can be used to communicate between address spaces on different machines or the same machine
 - Services can be run wherever it’s most appropriate
 - Access to local and remote services looks the same
- Examples of modern RPC systems:
 - ONC/RPC (originally SUN RPC) in Linux, Windows,...
 - DCE/RPC (Distributed Computing Environment/RPC)
 - MSRPC: Microsoft version of DCE/RPC
 - RMI (Java Remote Method Invocation)

4/23/2014

Anthony D. Joseph

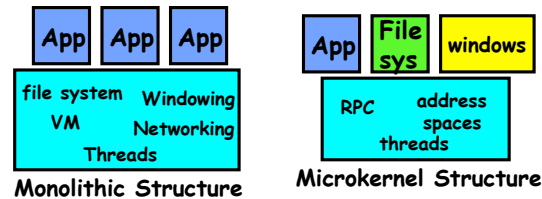
CS162

©UCB Spring 2014

23.8

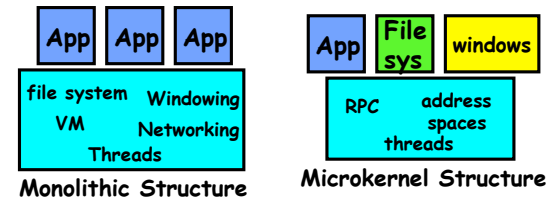
Microkernel Operating Systems

- Example: split kernel into application-level servers using RPC
 - File system looks remote, even though on same machine



4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.9

Microkernel Operating Systems



- Why split the OS into separate domains?
 - Fault isolation: bugs are more isolated (build a firewall)
 - Enforces modularity: allows incremental upgrades of pieces of software (client or server)
 - Location transparent: service can be local or remote
 - » For example in the X windowing system: Each X client can be on a separate machine from X server; Neither has to run on the machine with the frame buffer

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.10

Problems with RPC

- Handling failures
 - Different failure modes in distributed system than on a single machine
 - Without RPC a failure within a procedure call usually meant whole application would crash/die
 - With RPC a failure within a procedure call means remote machine crashed, but local one could continue working
 - Answer? Distributed transactions can help
- Performance
 - Cost of Procedure call « same-machine RPC « network RPC
 - Means programmers must be aware they are using RPC (so much for transparency!)
 - » Caching can help, but may make failure handling even more complex

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.11

Administrivia

- Project 4 design due date **changed**
 - Tuesday 4/29 by 11:59PM
- Midterm II is April 28th 4-5:30pm in **245 Li Ka Shing and 100 GPB**
 - Covers Lectures #13-23, projects, handouts, readings
 - Closed book and notes, no calculators
 - One double-sided handwritten page of notes allowed
 - Review session: Fri Apr 25th 4-6pm in 245 Li Ka Shing
 - Three years of finals and 2nd midterms exams online:
 - » Fall 2013, Spring 2013, Spring 2012, Fall 2011, Spring 2011

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.12

2min Break

4/23/2014

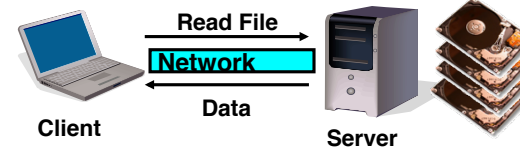
Anthony D. Joseph

CS162

©UCB Spring 2014

23.13

Distributed File Systems



- Distributed File System:
 - **Transparent** access to files stored on a remote disk
 - **Transparent concurrency**: All clients have the same view of the state of the file system.
 - **Failure transparency** The client and client programs should operate correctly after a server failure.
 - **Replication transparency** To support scalability, we may wish to replicate files across multiple servers. Clients should be unaware of this.
 - **Migration transparency** Files should be able to move around without the client's knowledge.

4/23/2014

Anthony D. Joseph

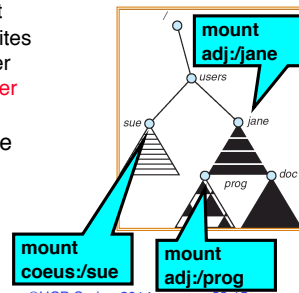
CS162

©UCB Spring 2014

23.14

Distributed File Systems

- Naming choices (always an issue):
 - *Hostname.localname*: Name files explicitly
 - » No location or migration transparency
 - *Mounting* of remote file systems
 - » System manager mounts remote file system by giving name and local mount point
 - » Transparent to user: all reads and writes look like local reads and writes to user
e.g. `/users/sue/foo` → `/sue/foo` on server
 - *A single, global name space*: every file in the world has unique name
 - » Location Transparency: servers can change and files can move without involving user



4/23/2014

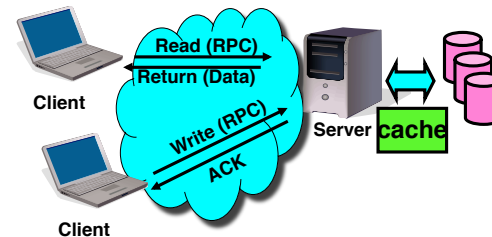
Anthony D. Joseph

CS162

©UCB Spring 2014

23.15

Simple Distributed File System



- EVERY read and write gets forwarded to server
- Advantage: Server provides completely consistent view of file system to multiple clients
- Problems? Performance!
 - Going over network is slower than going to local memory
 - Server can be a bottleneck

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.16

Failures



- What if server crashes? Can client wait until server comes back up and continue as before?
 - Any data in server memory but not on disk can be lost
 - Shared state across RPC: What if server crashes after seek? Then, when client does “read”, it will fail
 - Message retries: suppose server crashes after it does UNIX “rm foo”, but before acknowledgment?
 - » Message system will retry: send it again
 - » How does it know not to delete it again? (could solve with two-phase commit protocol, but NFS takes a more ad hoc approach)

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.17

Stateless Protocol



- **Stateless protocol:** A protocol in which all information required to process a request is passed with request
 - Server keeps no state about client, except as hints to help improve performance (e.g. a cache)
 - Thus, if server crashes and restarted, requests can continue where left off (in many cases)
- What if client crashes?
 - Might lose modified data in client cache
- Examples:
 - HTTP
 - REST (Representational State Transfer)
- Stateful
 - SOAP (Simple Object Access Protocol) - usually over HTTP!

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.18

Network File System (NFS)

- Three Layers for NFS system
 - **UNIX file-system interface:** open, read, write, close calls + file descriptors
 - **VFS layer:** distinguishes local from remote files
 - » Calls the NFS protocol procedures for remote requests
 - **NFS service layer:** bottom layer of the architecture
 - » Implements the NFS protocol

4/23/2014

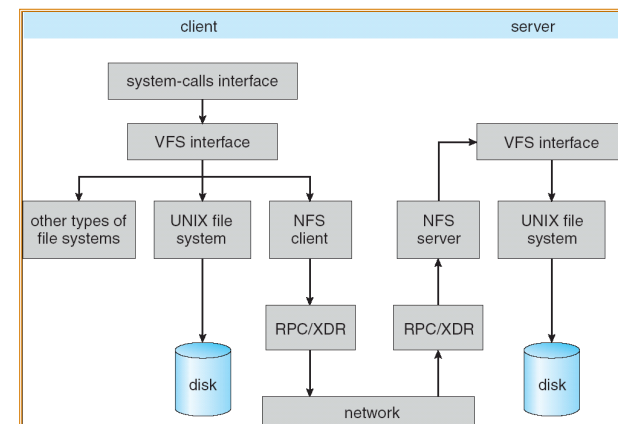
Anthony D. Joseph

CS162

©UCB Spring 2014

23.19

Schematic View of NFS Architecture



4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.20

Network File System (NFS)

- NFS Protocol: RPC for file operations on server
 - Reading/searching a directory
 - Manipulating links and directories
 - Accessing file attributes/reading and writing files
- **Write-through caching**: Modified data committed to server's disk before results are returned to the client
 - Lose some of the advantages of caching
 - Time to perform write() can be long
 - Need some mechanism for readers to eventually notice changes! (more on this later)

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.21

NFS Continued

- NFS servers are **stateless**; each request provides all arguments require for execution
 - E.g. reads include information for entire operation, such as `ReadAt(inumber, position)`, not `Read(openfile)`
 - No need to perform network open() or close() on file – each operation stands on its own
- **Idempotent**: Performing requests multiple times has same effect as performing it exactly once
 - Example: Server crashes between disk I/O and message send, client resend read, server does operation again
 - Example: Read and write file blocks: just re-read or re-write file block – no side effects
 - Example: What about “remove”? NFS does operation twice and second time returns an advisory error

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.22

NFS Continued

- Failure Model: Transparent to client system
 - Is this a good idea? What if you are in the middle of reading a file and server crashes?
 - Options (NFS Provides both):
 - » Hang until server comes back up (next week?)
 - » Return an error. (Of course, most applications don't know they are talking over network)

4/23/2014

Anthony D. Joseph

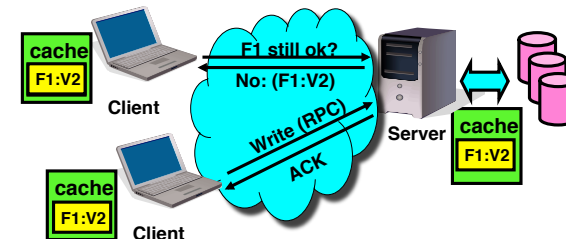
CS162

©UCB Spring 2014

23.23

NFS Cache consistency

- NFS protocol: weak consistency
 - Client polls server periodically to check for changes
 - » Polls server if data hasn't been checked in last 3-30 seconds (exact timeout is tunable parameter).
 - » Thus, when file is changed on one client, server is notified, but other clients use old version of file until timeout.



- What if multiple clients write to same file?
 - » In NFS, can get either version (or parts of both)
 - » Completely arbitrary! (You can try this at home ☺)

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.24

NFS Pros and Cons

- NFS Pros:
 - Simple, Highly portable
- NFS Cons:
 - Sometimes inconsistent!
 - Doesn't scale to large # clients
 - » Must keep checking to see if caches out of date
 - » Server becomes bottleneck due to polling traffic

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.25

Andrew File System

- Andrew File System (AFS, late 80's) → DCE DFS (commercial product)
- **Callbacks:** Server records who has copy of file (**stateful**)
 - On changes, server immediately tells all with old copy
 - No polling bandwidth (continuous checking) needed
- Write through on close
 - Changes not propagated to server until close()
 - Session semantics: updates visible to other clients only after the file is closed
 - » As a result, do not get partial writes: all or nothing!
 - » Although, for processes on local machine, updates visible immediately to other programs who have file open
- In AFS, everyone who has file open sees old version
 - Don't get newer versions until reopen file

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.26

Andrew File System (con't)

- Data cached on local disk of client as well as memory
 - On open with a cache miss (file not on local disk):
 - » Get file from server, set up callback with server
 - On write followed by close:
 - » Send copy to server; tells all clients with copies to fetch new version from server on next open (using callbacks)
- What if server crashes? Lose all callback state!
 - Reconstruct callback information from client: go ask everyone "who has which files cached?"

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.27

Andrew File System (con't)

- AFS Pro: Relative to NFS, less server load:
 - Disk as cache ⇒ more files can be cached locally
 - Callbacks ⇒ server not involved if file is read-only
- For both AFS and NFS: central server is bottleneck!
 - Performance: all writes→server, cache misses→server
 - Availability: Server is single point of failure
 - Cost: server machine's high cost relative to workstation

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.28

Quiz 23.1: RPC and NFS

- Q1: True _ False _ RPC requires special networking support and functionality
- Q2: True _ False _ The client and server for RPC must use the same hardware architecture (e.g., little endian)
- Q3: True _ False _ Local procedure call << same-machine RPC << remote machine RPC
- Q4: True _ False _ NFS provides weak client-server data consistency

2min Break

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.29

Quiz 23.1: RPC and NFS

- Q1: True _ False **X** RPC requires special networking support and functionality
- Q2: True _ False **X** The client and server for RPC must use the same hardware architecture (e.g., little endian)
- Q3: True **X** False _ Local procedure call << same-machine RPC << remote machine RPC
- Q4: True **X** False _ NFS provides weak client-server data consistency

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.30

Distributed Object-Oriented Systems

Distributed systems, like any complex software benefit from careful software architecture, especially object-oriented programming.

Major efforts were devoted to OOP distributed systems architectures:

- CORBA (Common Object Request Broker Architecture)
- DCOM (Distributed Component Object Model) from MS, which drew heavily from the open system DCE/DFS

These systems use remote methods, and add object proxying and even garbage collection.

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.31

Internet-Scale Distributed Computing

- CORBA and DCOM were robust, powerful RPC-based distributed object systems. They were supposed to become the substrate for internet-scale distributed computing. What happened? (they didn't)
- From last time:
 - Morris worm
 - Code Red
 - Slammer
 - which led to...
- Ubiquitous firewalls, packet filters etc., across the internet.
- HTTP (port 80) was the only reliable route to a remote host

4/23/2014 Anthony D. Joseph CS162 ©UCB Spring 2014 23.32

Internet-Scale Distributed Computing

- One approach is to tunnel other types of payload (other than HTTP) through port 80, and demultiplex at the server
 - Usually, but not always, this works
- Instead many systems have used HTTP directly as a high-level transport for RPC
- A cluster of technologies have developed around data messaging, RPC and distributed objects over HTTP:
 - Simple Object Access Protocol (SOAP),
 - Web Services Description Language (WSDL),
 - REpresentation State Transfer (REST),
 - and enabled by XML and JSON (JavaScript Object Notation)

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.33

WWW- SOAP RPC

Stateful protocol covering the following four main areas:

- A **message format** for one-way communication describing how a message can be packed into an XML document
- A **description** of how a SOAP message should be transported using HTTP (for Web-based interaction) or SMTP (for e-mail-based interaction). Also, TCP, UDP, ...
- A **set of rules** that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message
- A **set of conventions** on how to turn an RPC call into a SOAP message and back

4/23/2014

Anthony D. Joseph

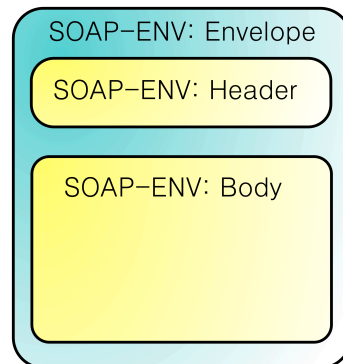
CS162

©UCB Spring 2014

23.34

SOAP Message

Typically an XML element containing header and body elements



4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.35

SOAP Message

Typically an XML element containing header and body elements

```
<SOAP:Envelope xmlns:SOAP=
  "http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header>
    <!-- content of header goes here -->
  </SOAP:Header>
  <SOAP:Body>
    <!-- content of body goes here -->
  </SOAP:Body>
</SOAP:Envelope>
```

4/23/2014

Anthony D. Joseph

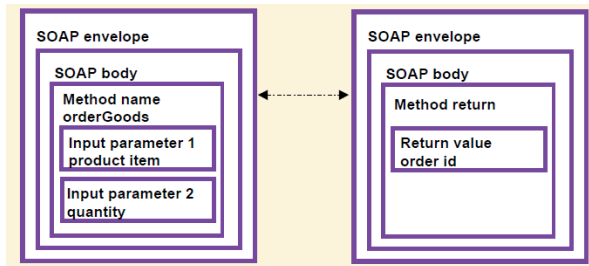
CS162

©UCB Spring 2014

23.36

SOAP RPC

SOAP RPC messages typically encode arguments that are presented to the calling program as parameters and return values:



4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.37

SOAP RPC

```
POST /travelservice
SOAPAction: "http://www.acme-travel.com/flightinfo"
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP=
  "http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <m:GetFlightInfo
      xmlns:m="http://www.acme-travel.com/flightinfo"
      SOAP:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="
        "http://www.w3.org/2001/XMLSchema-instance">
      <airlineName xsi:type="xsd:string">UI
      </airlineName>
      <flightNumber xsi:type="xsd:int">506
      </flightNumber>
    </m:GetFlightInfo>
  </SOAP:Body>
</SOAP:Envelope>
```

Method: GetFlightInfo

Arg #1: airlineName

Arg #1: flightNumber

4/23/2014

23.38

SOAP Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP=
  "http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body>
    <m:GetFlightInfoResponse
      xmlns:m="http://www.acme-travel.com/flightinfo"
      SOAP:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="
        "http://www.w3.org/2001/XMLSchema-instance">
      <flightInfo>
        <gate xsi:type="xsd:int">10</gate>
        <status xsi:type="xsd:string">ON TIME</status>
      </flightInfo>
    </m:GetFlightInfoResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

Method: GetFlightInfo

Return value #1: Gate

Return value #2: Status

4/23/2014

23.39

WSDL

Is “Web Services Description Language” an XML format for specifying metadata about a SOAP protocol

WSDL is used to describe precisely

- **what** a service does, i.e., the operations the service provides,
- **where** it resides, i.e., details of the protocol specific address, e.g., a URL, and
- **how** to invoke it, i.e., details of the data formats and protocols

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.40

WSDL

Service Implementation section →
Service Interface section ↓

```
<message name="GetFlightInfoInput">
  <part name="airlineName" type="xsd:string"/>
  <part name="flightNumber" type="xsd:int"/>
</message>

<message name="GetFlightInfoOutput">
  <part name="flightInfo" type="xsd:FlightInfoType"/>
</message>

<message name="CheckInInput">
  <part name="body" element="ticket:Ticket"/>
</message>

<portType name="AirportServicePortType">
  <operation name="GetFlightInfo">
    <input message="tns:GetFlightInfoInput"/>
    <output message="tns:GetFlightInfoOutput"/>
  </operation>
  <operation name="CheckIn">
    <input message="tns:CheckInInput"/>
  </operation>
</portType>
```

```
<binding name="AirportServiceSoapBinding"
  type="tns:AirportServicePortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetFlightInfo">
    <soap:operation style="rpc">
      <soapAction="http://acme-travel.com/flightinfo"/>
    </soap:operation>
    <input>
      <soap:body use="encoded"
        namespace="http://acme-travel.com/flightinfo"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded"
        namespace="http://acme-travel.com/flightinfo"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
  <operation name="CheckIn">
    <soap:operation style="document">
      <soapAction="http://acme-travel.com/checkin"/>
    </soap:operation>
    <input>
      <soap:body use="literal"/>
    </input>
  </operation>
</binding>

<service name="travelservice">
  <port name="travelservicePort"
    binding="tns:AirportServiceSoapBinding"
    soap:address location="http://acmetravel.com/travelservice"/>
</service>
```

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.41

REpresentation State Transfer (REST) – “post RPC”

- Lightweight, **Stateless**, Client/Server Protocol – Used by Netflix, Facebook, Salesforce, Google Translate, PayPal, and many other sites
- Key principles:
 - Each message contains all info needed by receiver to understand and/or process it (*keep things simple!*)
 - All resources are uniquely addressable via Uniform Resource Identifiers (*everything is a resource!*)
 - Well-defined POST, GET, PUT, DELETE operations can be applied to all resources (*similar to DB's Create, Read, Update, Delete ops*)
 - Use of hypermedia both for application information *and* state transitions (*like a user browsing and clicking on links!*)

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.42

REST

Idempotent: repeated application of the operation does not change the state of the target

Method	Meaning	Idempotent?
GET	Retrieve a COPY of a Resource	YES
DELETE	Remove a Resource	YES
POST	Update a Resource	NO
PUT	Create a Resource	YES

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.43

REST example

```
<user>
  <name>Jane</name>
  <gender>female</gender>
  <location href="http://www.example.org/us/ny/new_york">
    New York City, NY, USA</location>
</user>
```

This documentation is a representation used for the User resource

It might live at <http://www.example.org/users/jane/>

- If an app needs information about Jane, it GET's this resource
- If they need to modify it, they GET it, modify it, and PUT it back
- The href to the Location resource allows smart clients to gain access to its information with another simple GET request

Key implication: Clients cannot be “thin”; need to understand resource formats

4/23/2014

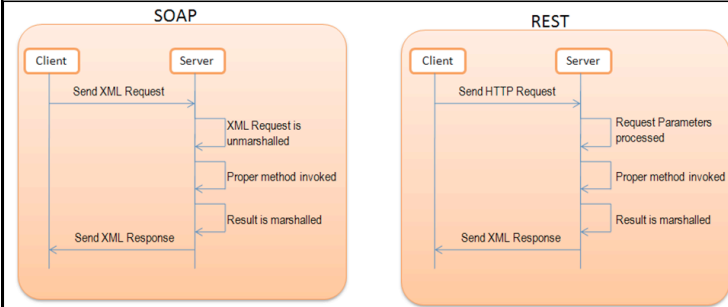
Anthony D. Joseph

CS162

©UCB Spring 2014

23.44

REST vs. SOAP



- REST is stateless
- REST avoids the need to generate and parse XML strings

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.45

REST vs. RPC

In RPC systems, the design emphasis is on **verbs**

- What operations can I invoke on a system?
- `getUser()`, `addUser()`, `removeUser()`, `updateUser()`, `getLocation()`, `updateLocation()`, `listUsers()`, `listLocations()`, etc.

In REST systems, the design emphasis is on **nouns**

- User, Location
- In REST, you would define XML representations for these resources and then apply the standard (POST, GET, PUT, DELETE) methods to them

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.46

Conclusion

- **Remote Procedure Call (RPC):** Call procedure on remote machine
 - Provides same interface as procedure
 - Automatic packing and unpacking of arguments without user programming (in stub)
- **Distributed File System:**
 - Transparent access to files stored on a remote disk
 - » NFS and AFS use caching for performance
- **SOAP:**
 - An RPC protocol and an RPC description format
- **REST:**
 - Simplicity of RPC without any state and without “verbs”

4/23/2014

Anthony D. Joseph

CS162

©UCB Spring 2014

23.47