DISTRIBUTED COMPUTER SYSTEMS

Communication Fundamental
REMOTE PROCEDURE CALL

Dr. Jack Lange
Computer Science Department
University of Pittsburgh

Fall 2015

Outline

- Communication Architecture Fundamentals
- Middleware Layer
- Types of Communications
  - Transient, Asynchronous and Synchronous
- Remote Procedure Call
  - Basic Operation
  - Parameter Passing
  - Parameter Specification and Stub Generation
  - RPC Case Study: DCE
- Conclusion
Communication Fundamentals

Layering, Middleware and Communication Types

Protocol Architecture – Layering

- Layers, interfaces, and protocols in the OSI model.
Layered Protocols – Encapsulation

- A typical message as it appears on the network.

Middleware Protocols

- A distinction can be made between high-level communication protocols and protocols establishing various middleware services
  - Authentication service to prove a communication channel claimed identity
  - Commit protocols to ensure atomicity
  - Access control protocol through locks, for example
- Above protocols offer specific services, which are highly independent of any specific application
  - Middleware communication protocols support high-level communications
Middleware Adapted Reference Model

- An adapted reference model for networked communication.

Middleware Communication Alternatives

The core of an electronic mail system can be seen as a middleware communication service, involving three major components:

- User agents – Eudora, Outlook, Pine,
- Mail servers for incoming and outgoing messages
- Simple Mail Transfer Protocol: SMTP
Middleware Communication Alternatives

- Viewing middleware as an intermediate, distributed, service in application-level communication.

Middleware Communication Types

- **Persistent Communication** – A transmitted message, is stored the communication middleware until deliver is successfully achieved
  - Neither the sender or receiver application need be running after submitting the message

- **Transient Communication** – Message is stored by communication system only as long as sender and receiver are executing
  - Middleware is unable to deliver a message due to transmission interrupt
    - Transport-level communication service offer transient communication,
    - Traditional store-and-forward, with possibility of packet dropping
Middleware Communication Types

- **Synchronous** – Sender is blocked until its request is known to be delivered
  - Synchronization can occur in three different ways
    - Sender is blocked until it received notification from middleware that it took charge of the message
    - Sender may synchronize until its request is received by intended recipient
    - Sender waits until recipient returns response

- **Asynchronous** – Sender continues execution, immediately after submitting message
  - Message is temporarily stored by middleware upon submission

**REMOTE PROCEDURE CALL**

- Basic Operation, Stubs, Parameter Passing
Remote Procedure Call

- Explicit message passing has been typically used in early distributed systems
  - The paradigm does not achieve access transparency
    - `send()` and `receive()` primitives do not conceal communication from the communicating entities
- Alternative method to message passing – Allow programs to call procedures located in other machines
  - Simple and elegant idea, but subtle problems may exist

Procedure Call – Basic Operation

- Conventional procedure call
  - `count = read(fd, buff, nbytes)`
  - `fd` = file descriptor, handle
  - `buff` = an array of characters
  - `nbyte` = number of bytes to be read
- The execution of a procedure call made from the main program requires the following steps:
  - Caller pushes the parameters onto the stack in order, last one first, including return address
  - The `read()` procedure puts `return value` in a register, removes `return address` and `transfers control` back to caller
Conventional Procedure Call

Parameter passing in a local procedure call

Parameter Passing

- **Call-by-Value** – To the called procedure a value is just an initialized local variable
- **Call-by-Reference** – The reference parameter is a pointer to a variable, rather than the value of the variable
  - The called procedure modifies the variable in the calling procedure
- **Call-by-Copy/Restore** – The variable is copied onto the stack by the caller and copied back after the call, overwriting the caller’s original value
  - Not often used in computer languages

Which Mechanism to Use in RPC?
Remote Procedure Call

- Principle of RPC between a client and server program.

![Diagram of Remote Procedure Call]

RPC Design Issue

- Make RPC look like LPC, as much as possible
  - Transparency – Local procedure should not be aware that the called procedure is executing remotely
  - RPC achieves its transparency in a similar way as LPC by invoking a library function call
    - The execution of the RPC uses the concept of a stub
      - A client stub, at the calling procedure
      - A server stub, at the called procedure
      - Communication between the stubs is achieved through message passing, transparently from the local procedure
RPC Information Flow

Client (Caller)

Pack Parameters

Call

Return

Send

Receive

Packet Handler

Client Mbox

Network

Server Stub

Unpack Results

Pack Results

Call

Unpack Parameters

Server (Callee)

Packet Handler

Server Mbox

Network

RPC COMPILATION

Interface Definition

IDL Compiler

Client Stub

Header File

Server Stub
RPC Steps – Example

- The steps involved in doing a remote computation through RPC.

**RPC FAILURE HANDLING**

Failures can be of three types
- Lost message
- Server crash
- Client crash
RPC Details

- Equivalence with regular procedure call
  - Parameters – Request Message
  - Result – Reply message
  - Name of Procedure – Passed in request message
  - Return Address – client return mail box

- Where do stubs come from? – Compiler generates stubs
  - Input: interface definitions in an “interface definition language (IDL)”
    - Contains, among other things, types of arguments/return
  - Output: stub code in the appropriate source language
    - Client code to pack and send message, wait for result, unpack result and return to caller
    - Server Code to unpack message, call procedure, pack and send back results

RPC Implementation Issues

- Cross-platform issues
  - What if client/server machines are different architectures or in different languages?
  - How does client know where to send call request?
    - Biding process – Need to translate name of remote service into network endpoint: Server network address, port number, possibly other information

- How to pass complex structures, pointers, big arrays?
  - Handling these structure could be very costly, and perhaps impractical to pass as arguments
  - Should there be limitation on size and types of RPC arguments?
Passing Parameters – Original Message

- Original message on the sending machine

<table>
<thead>
<tr>
<th>Byte Address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>L</td>
<td>I</td>
<td>J</td>
</tr>
</tbody>
</table>

Passing Parameters – Upon Arrival

- The original message upon arrival on the receiving machine.

<table>
<thead>
<tr>
<th>Byte Address</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>J</td>
<td>I</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>
Passing Parameters – Upon Conversion

- The message after being inverted at receiving machine

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>I</td>
<td>J</td>
<td></td>
</tr>
</tbody>
</table>

Parameter Specification and Stub Generation

```
foobar( char x; float y; int z[5] )
{
    ....
}
```

<table>
<thead>
<tr>
<th>foobar's local variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>z[0]</td>
</tr>
<tr>
<td>z[1]</td>
</tr>
<tr>
<td>z[2]</td>
</tr>
<tr>
<td>z[3]</td>
</tr>
<tr>
<td>z[4]</td>
</tr>
</tbody>
</table>

Procedure Description

Corresponding Message
Asynchronous RPC (2)

- The interaction using asynchronous RPC.

Asynchronous RPC

- Interaction between client and server in a traditional RPC
Asynchronous RPC – Two RPCs

- A client and server interacting through two asynchronous RPCs.

Overcoming Lost Packets
Overcoming Lost Packets

Client Sends Request

Server

Timeout!

Retransmit

Ack For Request

Reply

Ack

For Request

Timeout!
Costs in fault-tolerant version?

- Acknowledgments are expensive – Must be avoided, when possible
  - For example, if the reply can be sent quickly suppress the initial acknowledgement
- Retransmission is costly – Try and tune the delay to be “optimal”
- For big messages, send packets in bursts and acknowledge a burst at a time, not burst by burst

Big Packets

Client Sends Request as a Burst

Server Ack Entire Burst
Reply
Ack For Reply
RPC “Semantics”

- **At most once:** Request is processed 0 or 1 times
- **Exactly once:** Request is always processed 1 time
- **At least once:** Request is processed 1 or more times
- ... but exactly once is **impossible** because we can’t distinguish packet loss from true failures!
  - In both cases, RPC protocol simply times out.

REMOTE PROCEDURE CALL

DCE – Case Study
Writing a Client and a Server

The steps in writing a client and a server in DCE RPC:

- Uuidgen
  - Interface definition file
  - IDL compiler

- Writes three files:
  - A header file (e.g., interface.h, in C terms).
  - The client stub.
  - The server stub.
Binding a Client to a Server

- Registration of a server makes it possible for a client to locate the server and bind to it.

- Server location is done in two steps:
  1. Locate the server’s machine.
  2. Locate the server on that machine.

Binding a Client to a Server

- Client-to-server binding in DCE
Conclusion

- Communication Architecture Fundamentals
- Middleware Layer
- Types of Communications
  - Transient, Asynchronous and Synchronous
- Remote Procedure Call
  - Basic Operation
  - Parameter Passing
  - Parameter Specification and Stub Generation
  - RPC Case Study: DCE