Communication

Users View

Applications do not care about networking details.

**Middleware achieves transparency.**

Middleware services have witnessed explosive growth.
Naming

Names should be location independent, unique
Naming service should be scalable.

How does Domain Name Service (DNS) work?

A central table will contain billions of names that will be concurrently accessed by millions of people. Different parts of the table are massively replicated and distributed at numerous locations.
Example of name resolution for ghosh@hp.uiowa.edu

Each server has authority over a small part of the naming hierarchy.

Each server has pointers pointing to tables at the next lower and higher levels.

These tables are replicated. It helps with load balancing. Resolved names are put in a local cache that speeds up response.
Naming Mobile Clients

- Alice, where are you? ……))))))))))))
  (broadcast)
- I am here!!

Such address resolution works on Ethernet-based LANs for translating IP-addresses to MAC addresses. This is not efficient on WANs.

A home agent keeps track of the client. Mobile IP is comparable to call forwarding.
Remote Procedure Call (RPC)

Clients and servers are on different machines. RPC fakes remote calls by making it look like local procedure calls.

Transparency needs parameter marshaling and un-marshaling

<table>
<thead>
<tr>
<th><strong>Client stub</strong></th>
<th><strong>Server stub</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pack parameters into a message;</td>
<td>while no message do skip;</td>
</tr>
<tr>
<td>Send message to remote machine</td>
<td>Unpack the call parameters;</td>
</tr>
<tr>
<td><strong>while no result do skip;</strong></td>
<td>call the server procedure;</td>
</tr>
<tr>
<td>Receive result and unpack it;</td>
<td>Pack result into a message;</td>
</tr>
<tr>
<td><strong>Return</strong> to the client program;</td>
<td>Send it to the client</td>
</tr>
</tbody>
</table>

Concurrent calls are serialized.
Impact of failures on RPC

What happens if messages are lost, or the server crashes? Various semantics are possible:

- At least once
- At most once
- Exactly once

What is the relevance of these Call semantics?
More on RPC and failures

Consider the **serialization** of concurrent calls in the client-server environment. Who will handle the blocking?

If the server handles blocking and the current client crashes, then all clients will be blocked forever.

If clients handle blocking, then how will a client know if another client has already initiated an RPC? *Requires additional coordination among clients.*
**Event Notification**

Events are comparable to *interrupts* or *exceptions*. They are asynchronous in nature.

*Notify me when the airfare from Cedar Rapids to Paris will be less than $300.*

Extensively used in *Publish-Subscribe*. Communication is possible via brokering. Commercial product: *Jini* by SUN.

Microsoft’s *smart home* project relies on event notification. These include device discovery, and disaster notification to homeowners when they are away from home.
**Mobile Agents**

Communicate via *messengers* instead of *messages*.

**Example.** Compute the *smallest id* in a ring of processes, and inform it to every process.

![](image)

**Question 1.** How will you do it using messages? How many messages will be needed in the worst case?

**Question 2.** How will you solve it using a mobile agent? What is the worst-case message complexity now?
Solution using Mobile Agents

Compute the lowest price of an item sold in n different stores.

{Let best be the lowest price}

initially, best = price(home)

do current ≠ home

    if price(i) < best
        best := price(i)
    fi;

    visit next; {determined by traversal algorithm}

od

Ants mimic primitive biological entities and have been successfully used in network management (shortest path, congestion control etc).
Models of Communication

Why models?
In general, there are many dimensions of variability in distributed systems. Examples: interprocess communication mechanisms, failure classes, security mechanisms etc.

Models are simple abstractions that help understand the variability -- abstractions that preserve the essential features, but hide the implementation details from observers who view the system at a higher level.
Example 1. A Message-Passing Model

System is a graph $G = (V, E)$.

$V =$ set of nodes (sequential processes)

$E =$ set of edges (links or channels) (bi- or unidirectional?)

**Four** types of actions by a process:

- *Internal* action
- *Communication* action
- *Input* action (from the environment)
- *Output* action (to the environment)
Axioms of a Reliable FIFO Channel

**Axiom 1.** Message $m$ sent $\rightarrow$ message $m$ received

**Axiom 2.** Message propagation delay is arbitrary but finite.

**Axiom 3.** $m_1$ sent before $m_2$ $\rightarrow$ $m_1$ received before $m_2$.

**Channel capacity.** Finite or infinite?

*Output action* $\equiv$ append to the channel

*Input action* $\equiv$ read and delete from the channel

Can you evaluate the predicate $\text{empty}(c)$?
Life of a process

Some initiator sends a message. Thereafter every process repeatedly executes the following step until termination.

if a message $m$ is received
then
begin
evaluate a predicate with $m$ and internal variables;
if predicate = true then
    update the internal variables;
    send zero or more messages;
else skip {do nothing}
end
endif
## Synchrony and Asynchrony

<table>
<thead>
<tr>
<th>Synchronous clocks</th>
<th>Local physical clocks synchronized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous processes</td>
<td>Lock-step synchrony</td>
</tr>
<tr>
<td>Synchronous channels</td>
<td>Bounded delay</td>
</tr>
<tr>
<td>Synchronous message-order</td>
<td>First-in first-out channels</td>
</tr>
<tr>
<td>Synchronous communication</td>
<td>Communication via handshaking</td>
</tr>
</tbody>
</table>

Send & receive operation can be **blocking** or **non-blocking**

Example of asynchronous communication:

*Postal communication*

Example of synchronous communication:

*Telephone communication*

Review the kind of communication in (1) RPC, (2) RMI, (3) Email (4) Ada Rendezvous
The shared memory model

Programmers like the shared-memory models (Linda, DSM).

Exercise. Implement a channel of capacity max-1

shared var p, q : integer {initially p = q}
buffer: array [0..max-1] of message

{sender process P}
var i : integer {initially 0}
repeat
if p \neq q-1 \mod max then
    begin
        buffer[p] := m[i];
        i := i+1;
p := p+1 \mod max
    end
forever

{receiver process Q}
var j : integer {initially 0}
repeat
    if q \neq p then
        begin
            m[j] := buffer[q]
j := j+1;
q := q+1 \mod max
        end
forever
Some variations of shared memory models

(a) State reading model. Each process can read the states of all of its neighbors

(b) Link register model. Each process can read from and write into the link registers shared between neighbors.