CORBA and object oriented middleware

Introduction
General info

- Web page
  - http://www.dis.uniroma1.it/~beraldi/Elective
- Exam
  - Project (application), plus oral discussion
- 3 credits
Roadmap

• Distributed applications
• Middleware for distributed applications
• OO middleware
• CORBA
• Java-RMI
Definitions

- Distributed application: An **Application composed of two or more components running on different machines connected via a network**

- A distributed (computing) system is: A collection of independent computers that appears to its users as a single coherent system.
Definitions

- Distributed application: *Application composed of two or more components running on different machines connected via a network*

- A distributed (computing) system is: *A collection of independent computers that appears to its users as a single coherent system.*
A distributed system organized as **middleware**.
The middleware layer extends over multiple machines, and offers each application the same interface.
This system supports two ‘local’ applications (A,C) and one distributed application (application B)
Characteristic of distributed applications (and systems)

• Independent failures
  – Components can fail individually
  – There is not a ‘global’ failure (consider Internet)

• Concurrent execution
  – Any component is autonomous
  – There can be, however, the need of actions to be coordinated

• No shared memory
  – Communication via message exchange
  – No global state

• No global time
A sees: Re: Meeting, Meeting, Re: Meeting

There is no need of a physical global time; rather logical time is enough
• **Distributed applications** that deliver message causally ordered can be easily built once a middleware providing the right primitive, e.g., casual-order multicast, exists.
Characteristics of decentralized algorithms

• No machine has complete information about the system state.
• Machines make decisions based only on local information.
• Failure of one machine does not ruin the algorithm.
• There is no implicit assumption that a global clock exists.
Goal of a distributed system (...and then of a middleware)

- Transparency
- Openness
- Scalability
# Transparency

<table>
<thead>
<tr>
<th>Transparency</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>Hide differences in data representation and how a resource is accessed</td>
</tr>
<tr>
<td>Location</td>
<td>Hide where a resource is located</td>
</tr>
<tr>
<td>Migration</td>
<td>Hide that a resource may move to another location</td>
</tr>
<tr>
<td>Relocation</td>
<td>Hide that a resource may be moved to another location while in use</td>
</tr>
<tr>
<td>Replication</td>
<td>Hide that a resource is replicated</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Hide that a resource may be shared by several competitive users</td>
</tr>
<tr>
<td>Failure</td>
<td>Hide the failure and recovery of a resource</td>
</tr>
</tbody>
</table>

Hides...
Enables/allows...

Transparency (Couloris)

• **Access transparency**
  – enables local and remote resources to be accessed using identical operations.

• **Location transparency**
  – enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

• **Concurrency transparency:**
  – enables several processes to operate concurrently using shared resources without interference between them.

• **Replication transparency:**
  – enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
Transparency (Couloris)

• **Failure transparency:**
  – enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

• **Mobility transparency:**
  – allows the movement of resources and clients within a system without affecting the operation of users or programs.

• **Performance transparency:**
  – allows the system to be reconfigured to improve performance as loads vary.

• **Scaling transparency:**
  – allows the system and applications to expand in scale without change to the system structure or the application algorithms.
Other goal of distributed system

- Transparency
- Openness
- Scalability
Openness

- A DS is open if the service it offers are defined using standard rules for
  - syntax (how to use, through interface)
  - semantics (what the service does)

- Syntax (easy)
  - IDL (Interface Definition Language)

- Semantics
  - Natural language
  - Semi formal (UML)
  - Formal (Petri nets, FSM,...)
Interface definition

• Complete
  – Contains all is needed

• Neutral
  – no implementation-dependent

• Interface is a key concept to support
  – Interoperability
  – Portability
  – Extensibility
Information hiding

• It is a well-known and powerful modularization principle
• Separation of what a ‘service’ does from how it is realized (implemented)
  – Interface vs implementation

POWER SUPPLY

12 v, up 1 A
Examples of where it is applied

- Interface
- Class
- Implementation time
- Signature
- Body
- Function/procedure with no side-effect
- Design time
- Application
- API
- Run time
Generalization

Application

Middleware/
Run-timelibrary

Operating System

Hardware

upcall
(i.e., exception raised by the OS)
Openness - Flexibility

- A flexible DS is composed of a small set components that can be combined together
- Separation between mechanism and policy
  - Mechanism: tells what a component can be done
  - Policy: how the component is used
- For example
  - The cache functionality in a web-browser is obtained via a component (cache) plus a policy (when using cache, how often replace the cache, e.g., according to the content, train time table
  - Policy as a component (?)
# Scalability

<table>
<thead>
<tr>
<th>Concept</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized services</td>
<td>A single server for all users</td>
</tr>
<tr>
<td>Centralized data</td>
<td>A single on-line telephone book</td>
</tr>
<tr>
<td>Centralized algorithms</td>
<td>Doing routing based on complete information</td>
</tr>
</tbody>
</table>
Scalability technique

• Asynchronous communication
  – No blocking calls \(\rightarrow\) increase concurrency (e.g., AJAX)

• Distribution
  – Divide the ‘load’ (e.g., DNS)

• Replication
  – Many nodes offers the same service (e.g., CDN)
  – Caching (local replica of data)
    • Consistency is an issue
Asynchronous comm
Distribution
Examples of distributed applications

• Parallel/distributed computation
  – Clusters
  – Grid computing
  – Cloud computing

• Information systems
  – Transactional systems
  – Enterprise Application Integration (EAI)

• Pervasive systems
  – Sensor networks
  – Home networks
Example of distributed computing system
Transaction processing systems

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_TRANSACTION</td>
<td>Mark the start of a transaction</td>
</tr>
<tr>
<td>END_TRANSACTION</td>
<td>Terminate the transaction and try to commit</td>
</tr>
<tr>
<td>ABORT_TRANSACTION</td>
<td>Kill the transaction and restore the old values</td>
</tr>
<tr>
<td>READ</td>
<td>Read data from a file, a table, or otherwise</td>
</tr>
<tr>
<td>WRITE</td>
<td>Write data to a file, a table, or otherwise</td>
</tr>
</tbody>
</table>
TP systems
Enterprise Application Integration

Client application

Communication middleware

Client application

Server-side application

Server-side application

Server-side application
Electronic Health Care Appl.

(a) Tilt sensor
ECG sensor
PDA
Motion sensors
body-area network

(b) Transmitter
GPRS/UMTS
External storage

body-area network
Sensor Networks and application

Operator's site

Sensor network

Sensor data is sent directly to operator
Sensor Networks

Each sensor can process and store data

Sensor network

Operator's site

Query

Sensors send only answers