Topics

1. Middleware
2. Technology classification
3. Distributed objects
4. Message-oriented middleware
5. Class exercise: Discussions & presentations
## Lecture plan:

1. Reading material provided 1 week in advance
2. Student to read papers before the lecture
3. Class exercises in groups (4) selected groups give presentations
Software architecture is (definition 1):
the fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

So, software architecture:
(1) defines structure
(2) specifies component communication
(3) addresses non-functional requirements
(4) is an abstraction
1. Middleware (1)

**What is Middleware?**

- Layer between OS and distributed applications
- Hides complexity and heterogeneity of distributed system
- Bridges gap between low-level OS communications and programming language abstractions
- Provides common programming abstraction and infrastructure for distributed applications

Distributed Applications

Middleware

Operating System Comms

Network

(remote calls, object invocation, messages, ...)

(sockets, IP, TCP, UDP, ...)

(packets, bytes, bits...)
1. Middleware (2)

What is middleware?

Middleware:
- provides proven ways to connect the various components in an application so that they can exchange information using relatively easy-to-use mechanisms.
- can be used:
  - in a wide range of different application domains
  - to ‘wire’ together numerous components in useful well understood topologies:
    - one-to-one, one-to-many, many-to-many
- is completely hidden from the application user.
1. Middleware (3)

Why middleware?
Middleware has evolved over the last 25 years in order to:

- simplify the building of complex, distributed, concurrent applications.
- institutionalise proven design practices by supporting them in off-the-shelf middleware technologies.

For the software architect important skills are:

- how to select, mix and match architectures & technologies in a way that meets the application’s requirements & constraints.
- =>deep knowledge of technologies
## 2. Technology classification (1)

<table>
<thead>
<tr>
<th>Business Process Orchestrators</th>
<th>e.g. BizTalk, StaffWare, ActiveBPEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Brokers</td>
<td>e.g. BizTalk, WebSphere Message Broker</td>
</tr>
<tr>
<td>Application Servers</td>
<td>e.g. J2EE, CCM (CORBA COMPONENT MODEL), .NET</td>
</tr>
<tr>
<td>Transport</td>
<td>e.g. Distributed Objects systems (CORBA), Message-Oriented Middleware</td>
</tr>
</tbody>
</table>

- **Transport layer**: provides the basic services for sending requests & moving data between components.
- **Application servers:**
  - typically **built on top of the basic transport services**.
  - **Provide additional capabilities** e.g. transactions, security, directory services.
  - **support a programming model** for building multi-threaded server-based applications that exploit the additional services.
## 2. Technology classification (2)

| Business Process Orchestrators | e.g. BizTalk, StaffWare, ActiveBPEL |
| Message Brokers | e.g. BizTalk, WebSphere Message Broker |
| Application Servers | e.g. J2EE, CCM, .NET |
| Transport | e.g. Distributed Objects systems (CORBA), Message-Oriented Middleware |

- **Message brokers:**
  - use either a basic transport service or application servers or both & add a message processing engine.
  - engine provides high-level programming features for defining how to:
    - *exchange, manipulate, route messages* between application components
## 2. Technology classification (3)

<table>
<thead>
<tr>
<th>Business Process Orchestrators</th>
<th>e.g. BizTalk, StaffWare, ActiveBPEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Brokers</td>
<td>e.g. BizTalk, WebSphere Message Broker</td>
</tr>
<tr>
<td>Application Servers</td>
<td>e.g. J2EE, CCM, .NET</td>
</tr>
<tr>
<td>Transport</td>
<td>e.g. Distributed Objects systems (CORBA), Message-Oriented Middleware</td>
</tr>
</tbody>
</table>

- **Business process orchestrators (BPOs):**
  - augment message broker features to support *workflow-style* applications
  - for *workflow-style* applications (e.g. ERP systems) business processes may take hours or days to complete due to the need for people to perform certain tasks.
  - BPOs provide tools to:
    - describe processes
    - execute the processes & manage intermediate steps
Distributed objects technology (e.g. CORBA) dates back to the 1990s.

- **CORBA** (Common Object Request Broker Architecture) is an exemplar of this technology.
- **CORBA (from OMG) is based on an open standard**
- Currently used in environments where open standards are required (e.g. telecoms)
- but in business environment: **legacy**
3. Distributed objects (2)

ORBs (Object Request Brokers) provide communication between components (e.g. a client & a server)

- **servant objects** support interfaces specified using CORBA’s IDL
- **IDL** (Interface Description Language) defines the methods (plus pmts & return types) that a server object supports.
- programmer must write the code to implement each server method in a native language, e.g. C++, Java
3. Distributed objects (3)

CORBA IDL defines the methods (plus pmts & return types) a server object supports, e.g.

```
module ServerExample {
    interface myObject {
        string isAlive();
    }
}
```

The programmer must write the code to implement server method in e.g. C++, Java

```
class MyServant extends _MyObjectImplBase {
    public String isAlive() {
        return "Yes it is!\n";
    }
}
```
3. Distributed objects (4)

Client

Object Reference request

client ORB

Server

Servant reply

server ORB

Network

programmer must write the code to implement server method in e.g. C++, Java

class MyServant extends _MyObjectImplBase {
    public String isAlive( ) {
        return "Yes it is!\n";
    }
}

Client-side code to initialise client ORB & get a ref to server-side servant:

ORB orb = ORB.init( args, null )
//access CORBA naming service directory
MyServant servantRef = lookup("MyServant");
String reply = servantRef.isAlive( );
For software architect, some **essential design concerns** that must be addressed in applications:

- **Requests to servants are remote calls** (traverse ORB & network) $\Rightarrow$ slow
  - interface design should minimise remote calls
- **Servers may intermittently or permanently be unavailable** due to network or process or machine failure
  - $\Rightarrow$ applications need strategies to cope with failures
4. Message-oriented middleware MOM (1)

- is a key technology for building large-scale enterprise systems
- is a glue that binds together independent & autonomous applications & turns them into a single integrated system

**MOM basics:**
- uses an intermediate message queue
- a loosely-coupled asynchronous technology
- => sender & receiver of message are not tightly coupled
MOM basics cntnd:
- often implemented as a server that can handle messages from multiple concurrent clients
- can also be implemented in point-to-point fashion with ‘send’ and ‘receive’ queues on each communicating system.
- each queue has a name which senders and receivers specify for send & receive operations.
MOM basics cntnd

Basic responsibilities for a **MOM server**:

- **accept a message** from a sending application & **send** acknowledgement of receipt
- **place message at the back of queue** specified by sender
- **hold messages in queues** for possibly extended periods of time
- **dispatch queued messages** to receivers in **FIFO** style
4. Message-oriented middleware MOM (4)

MOM basics cntnd

MOM’s asynchronous de-coupled approach solves many common application design problems, e.g.:

- If sender doesn’t need a reply from receiver (*send -and-forget*)
- If sender doesn’t need an immediate reply from receiver (*no busy-waiting*)
- Receiver and/or network connection may not operate continuously (*messages will be delivered whenever they are available*)
4. Message-oriented middleware  MOM (5)

**MOM advanced features**
- message delivery
  - best effort, persistent, transactional, transactions, clustering, two-way messaging
- publish-subscribe

**MOM standards** (no single standard)
- Some standards:
  - Advanced message Queuing Protocol (AMQP)
  - OMG’s Data Distribution Service (DDS)

**READING FOR THE STUDENT**
Summary for Middleware

Middleware provides support for (some of):
- Naming, Location, Service discovery, Replication
- Protocol handling, Communication faults, QoS
- Synchronisation, Concurrency, Transactions, Storage
- Access control, Authentication

Middleware dimensions:
- Request/Reply vs. Asynchronous Messaging
- Language-specific vs. Language-independent
- Proprietary vs. Standards-based
- Small-scale vs. Large-scale
- Tightly-coupled vs. Loosely-coupled components
5. Class activities (20 minutes)

1. What are the benefits of using middleware technologies that are based on open standards?
References