COMMUNICATION MIDDLEWARE

INTRODUCTION TO COMMUNICATION MIDDLEWARE AND WEB SERVICE CONCEPTS
Contents
1. What is Middleware?
2. Basic (common) concepts of (distributed) middleware
3. Classification of middleware
4. Comparison of middleware technologies
5. Fallacies of distributed computing
Introduction to Middleware and Web Services

What is middleware?

*Wikipedia:* “Middleware is computer software that connects software components or applications.”

Middleware (MW) is the software between platform / network and the application. The term “middleware“ is very fuzzy, so almost everything is middleware. In this presentation, the focus is on distributed communication.
Basic (common) concepts of (distributed) middleware (1/6)
Middleware can be characterized according to the following criteria:

1. Serialization / marshalling
2. Data presentation
3. Distributed garbage collection
4. Location and discovery
5. Interaction model
6. Wire protocol / encapsulation (transport protocol)
7. Service description
8. Target domain
9. Platform independence
Introduction to Middleware and Web Services

Basic (common) concepts of (distributed) middleware (2/6)

1. Serialization / marshalling:
Serialization / marshalling converts data (objects, procedures, parameters) into a byte stream for transmission over the network.
Often serialization and marshalling are used synonymously, but there is a (subtle) difference:

a. **Serialization:**
Convert objects into a byte stream for transport over network or persistent storage.

b. **Marshalling:**
Bundle up parameters for a remote method call (serialization of parameters).
Basic (common) concepts of (distributed) middleware (3/6)

2. Data presentation:
Different middleware technologies present data in different ways to the application:
- Sockets → Plain byte stream (TCP) or byte packet (UDP, SCTP)
- RPC → Parameters of a procedure / method
- DAM → SQL statements, tables, keys
- Dist. tuples → Objects
- DOT → Objects
- MOM → Messages with „opaque“ body (message = data container)
- Web service → XML fragment, JSON

3. Distributed garbage collection (GC):
Local objects are garbage collected by the local GC (or the appl. if there is no GC as in C++).
Remote objects may have multiple client objects that access them. Thus remote objects may only be garbage collected if there are no more references to these objects.
Usually remote garbage collectors use some kind of a reference counter for the remote objects.
4. Localization & discovery:
Localization & discovery is the process of finding a suitable (concrete) instance of a remote service, server or object. Usually this is done through some kind of registry or directory service.

5. Interaction model (request/reply, publish/subscribe):
The interaction model defines the way how the local and remote parties interact. There are 2 main models:

a. Synchronous request / reply

b. Asynchronous messaging

The receiver receives messages independently from the sender.
Introduction to Middleware and Web Services

Basic (common) concepts of (distributed) middleware (5/6)

6. Wire protocol / encapsulation (transport protocol):
The wire protocol defines the encapsulation of the data for the transport over the network. Serialization / marshalling converts the data into a format conforming to the wire protocol.
Example wire protocols:
- CORBA \(\rightarrow\) IIOP
- Web services \(\rightarrow\) SOAP / XML over HTTP
- RPC \(\rightarrow\) XDR

7. Service description:
Remote services can be described formally with a description language. Often such a service or interface description is used to create code (local and remote objects).
Example service descriptions:
- CORBA \(\rightarrow\) IDL (Interface Description Language)
- Web service \(\rightarrow\) WSDL (Web Service Description Language)
- RPC \(\rightarrow\) XDR (External Data Representation)
Basic (common) concepts of (distributed) middleware (6/6)

8. Target domain:
Even though middlewares typically use TCP/IP as network protocol(s), not all are suited for use over the Internet due to different reasons:
¬ Some middlewares are very „chatty“ (a lot of messages going back and forth).
¬ Middlewares use different port ranges, thus there are potential problems with firewalls.

Target domains for middleware:
a. Internet (WAN)
b. Intranet (local network, LAN)
c. Host
d. Inter-process communication (IPC) between applications
e. Embedded devices (small footprint required, usually in C++)

9. Platform dependence:
Some middleware(s) are only available on a specific platform like Java, other middleware(s) were designed to be platform independent.

Example platform dependent MW: JMS, RMI (both use the Java platform)
Example platform independent MW: CORBA, web services (.Net and Java web service client and server interoperate)
Introduction to Middleware and Web Services

Classification of middleware (1/5)
The following is a simple classification scheme for middleware technologies.

1. Plain old sockets:
Sockets are the basis of all other middleware technologies.

2. RPC – Remote Procedure Call:
RPC technologies provide a simple means of distributing application logic on different hosts.
Classification of middleware (2/5)

3. TPM - Transaction Processing Monitors:
TPMs are a specialty MW targeted at distributed transactions.

4. DAM - Database Access Middleware:
Databases can be used to share and communicate data between distributed applications.
Introduction to Middleware and Web Services

Classification of middleware (3/5)

5. Distributed Tuple:
Distributed tuple spaces are implementations of a distributed shared memory space.

6. DOT (Distributed Object Technology) / OOM (Object Oriented Middleware):
DOT extends the object-oriented paradigm to distributed applications.
Classification of middleware (4/5)

7. MOM (Message Oriented Middleware):
In message oriented middleware, messages are exchanged asynchronously between distributed applications (senders and receivers).

8. Web services:
Web services expose services (functionality) on a defined interface, typically accessible through the web protocol HTTP.
**Classification of middleware (5/5)**

9. **Peer-to-peer middleware:**
In peer-to-peer middleware, there is no notion of clients and servers. Communication partners are peers with equal roles in the communication pattern.

10. **Grid middleware:**
Grid middleware provide computation power services (registration, allocation, de-allocation) to consumers.
# Introduction to Middleware and Web Services

## Comparison of middleware technologies (1/2)

Comparison of some concepts of middleware technologies

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Sockets</th>
<th>Sun-RPC</th>
<th>XML-RPC</th>
<th>RMI</th>
<th>DCOM</th>
<th>.Net remoting</th>
<th>CORBA</th>
<th>EJB</th>
<th>JMS</th>
<th>MSMQ</th>
<th>Web serv.</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serialization / marshalling</td>
<td>N/a</td>
<td>Yes</td>
<td>N/a</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Data presentation</td>
<td>N/a</td>
<td>XDR</td>
<td>XML</td>
<td>Java interface</td>
<td>.Net interface</td>
<td>Remote object</td>
<td>Remote object</td>
<td>Byte chunk</td>
<td>Byte chunk</td>
<td>SOAP / XML</td>
<td>XML</td>
<td></td>
</tr>
<tr>
<td>Distributed garbage collection</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
<td>N/a</td>
</tr>
<tr>
<td>Location and discovery</td>
<td>N/a</td>
<td>RPCBIND</td>
<td>N/a</td>
<td>JNDI</td>
<td>Windows registry</td>
<td>N/A</td>
<td>ORB</td>
<td>JNDI</td>
<td>JNDI</td>
<td>Active Directory</td>
<td>UDDI</td>
<td>N/a</td>
</tr>
<tr>
<td>Wire protocol (encapsulation)</td>
<td>TCP, UDP or SCTP</td>
<td>XDR</td>
<td>XML + HTTP</td>
<td>IIOP</td>
<td>MSRPC (DCE/RPC)</td>
<td>TCP SOAP+HTTP</td>
<td>XML (channel)</td>
<td>IIOP</td>
<td>IIOP</td>
<td>JMS specific</td>
<td>Proprietary</td>
<td>SOAP + HTTP</td>
</tr>
<tr>
<td>Service description</td>
<td>N/a</td>
<td>XDR</td>
<td>N/a</td>
<td>Java interface</td>
<td>MS IDL</td>
<td>.Net interface (C#, VB)</td>
<td>IDL</td>
<td>Java interface</td>
<td>N/a</td>
<td>N/a</td>
<td>WSDL</td>
<td>WSDL WADL</td>
</tr>
<tr>
<td>Target domain</td>
<td>Host, IPC, LAN, WAN</td>
<td>Host, LAN</td>
<td>Host, IPC, LAN, WAN</td>
<td>Host, LAN, IPC</td>
<td>Host, LAN, IPC</td>
<td>Host, IPC</td>
<td>Host, LAN, IPC</td>
<td>Host, LAN, IPC</td>
<td>Host, IPC, LAN, WAN</td>
<td>Host, IPC, LAN, WAN</td>
<td>Host, IPC, LAN, WAN</td>
<td></td>
</tr>
<tr>
<td>Platform dependence</td>
<td>All platforms</td>
<td>Unix, Linux</td>
<td>All platforms</td>
<td>Java</td>
<td>Windows</td>
<td>.Net</td>
<td>All platforms</td>
<td>Java (JEE)</td>
<td>Java</td>
<td>Windows</td>
<td>All platforms</td>
<td>All platforms</td>
</tr>
<tr>
<td>Middleware class</td>
<td>Socket</td>
<td>RPC</td>
<td>RPC</td>
<td>DOT</td>
<td>DOT</td>
<td>DOT</td>
<td>DOT</td>
<td>DOT</td>
<td>DOT</td>
<td>MOM</td>
<td>MOM</td>
<td>WS</td>
</tr>
</tbody>
</table>
### Comparison of middleware technologies (2/2)

#### Description of common middleware services (features):

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistence service (ORM)</td>
<td>Support for persistent storage of data</td>
</tr>
<tr>
<td>Transaction service</td>
<td>Support for atomic (ACID – Atomicity, Consistency, Isolation, Durability) sequences of actions (method calls) that can be rolled back in case of a failure</td>
</tr>
<tr>
<td>Concurrency control / synchronization</td>
<td>Services allowing to lock resources (transaction locks)</td>
</tr>
<tr>
<td>Naming and directory services</td>
<td>Registration / location / discovery of objects in the network based on a name (location based on an explicit name or ID)</td>
</tr>
<tr>
<td>Trading service</td>
<td>Similar to naming and directory service, but location based on operation names, parameters and result types (location based on properties)</td>
</tr>
<tr>
<td>Deployment infrastructure</td>
<td>Services for the deployment of objects into a run-time environment (e.g. bean container for EJB beans)</td>
</tr>
<tr>
<td>RPC support</td>
<td>Support for remote method call (call of methods on remote objects)</td>
</tr>
<tr>
<td>Life-cycle service</td>
<td>Creation / activation, copying, moving, deleting of objects (client- and / or server-activated objects)</td>
</tr>
<tr>
<td>Relationship definitions</td>
<td>Support for defining explicit relationships between objects</td>
</tr>
<tr>
<td>Query service</td>
<td>Mapping of objects to relational DBs (see ORM)</td>
</tr>
<tr>
<td>Licensing service</td>
<td>Controlled access to objects, definition of access control lists for different groups of clients</td>
</tr>
<tr>
<td>„Web service“ service</td>
<td>Access to objects via some kind of web service (e.g. access through HTTP)</td>
</tr>
<tr>
<td>Support for async callbacks / server push</td>
<td>Possibility to let server send callbacks to client (asynchronous, duplex interfaces)</td>
</tr>
<tr>
<td>Event / message service</td>
<td>Send / receive events (= messages) asynchronously</td>
</tr>
<tr>
<td>Externalization support</td>
<td>Possibility to store (=externalize) objects e.g. into the file system and load (=internalize) the object in the same or a different process</td>
</tr>
<tr>
<td>Security services</td>
<td>Support for identification, authentication, authorization, confidentiality (encryption), data integrity, propagation of credentials</td>
</tr>
<tr>
<td>Object pooling</td>
<td>Set of initialized (remote) objects kept ready for clients (improve performance, „recycle“ existing objects for new client requests)</td>
</tr>
<tr>
<td>Reflection / introspection</td>
<td>Possibility to query the available methods on an existing object</td>
</tr>
<tr>
<td>Load balancing</td>
<td>Distribution of client accesses over a defined number of server components or objects to evenly share the load</td>
</tr>
</tbody>
</table>
Fallacies of distributed computing
Common misconceptions or fallacies that architects / system designers should take into account when designing distributed applications:

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.
9. System clocks are identical.

(This list came about at Sun Microsystems by Peter Deutsch et.al.)

Distributing application logic over multiple hosts and servers incurs a non-negligible performance penalty that has to be taken into account.