COMMUNICATION MIDDLEWARE

INTRODUCTION TO COMMUNICATION MIDDLEWARE AND WEB SERVICE CONCEPTS

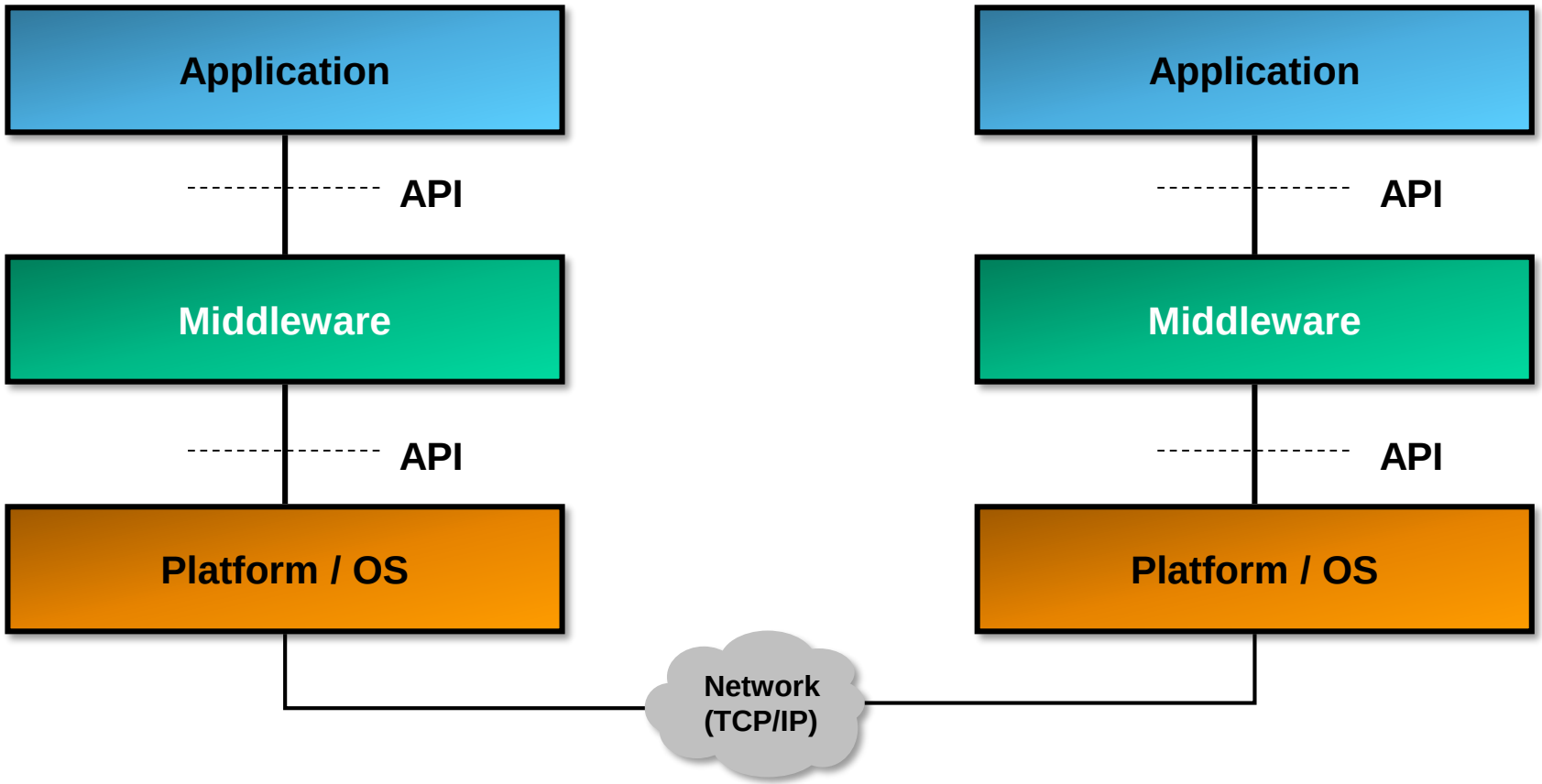
Peter R. Egli
peteregli.net

Contents

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**

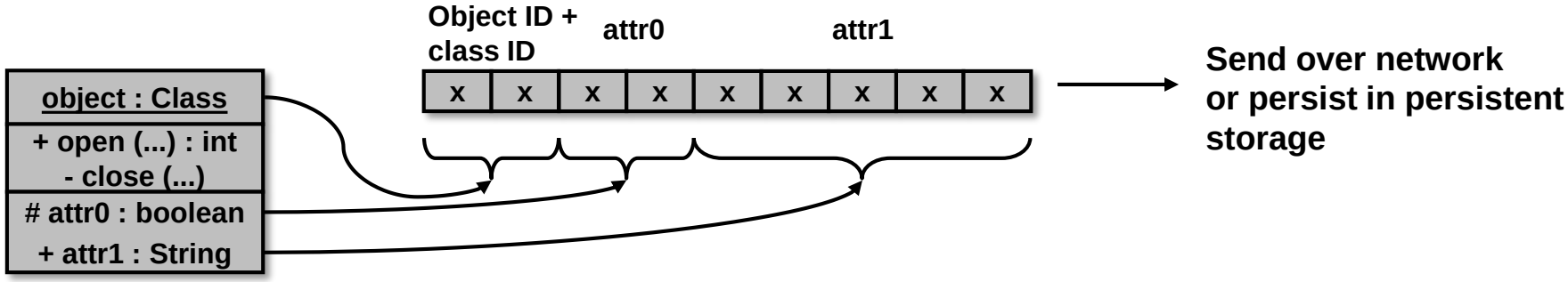
Basic (common) concepts of (distributed) middleware (2/6)

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:

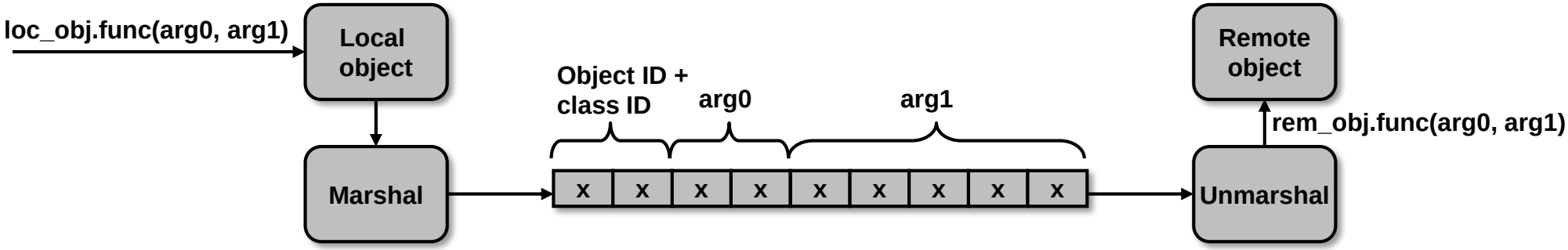
a. Serialization:

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)

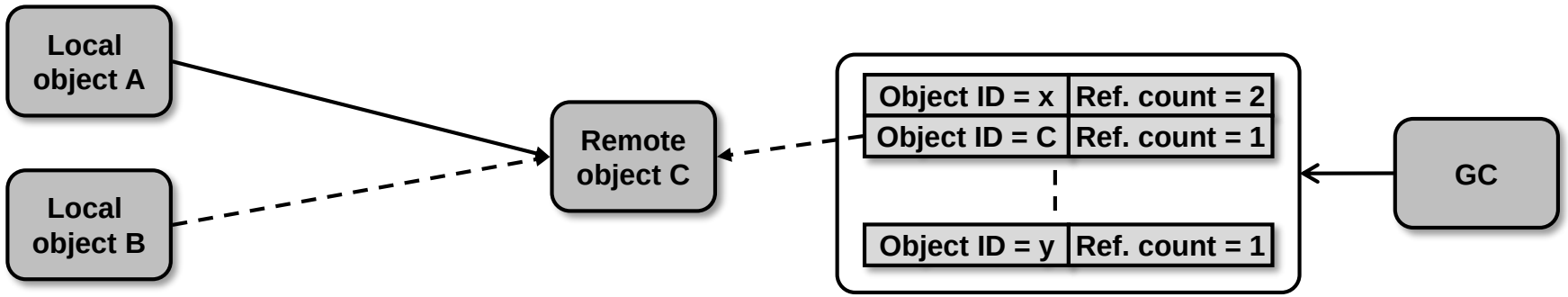
2. Data presentation:

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.

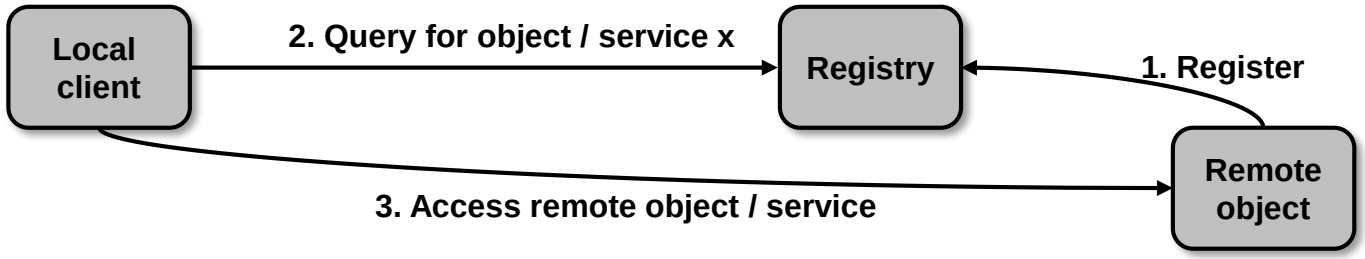


Basic (common) concepts of (distributed) middleware (4/6)

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.

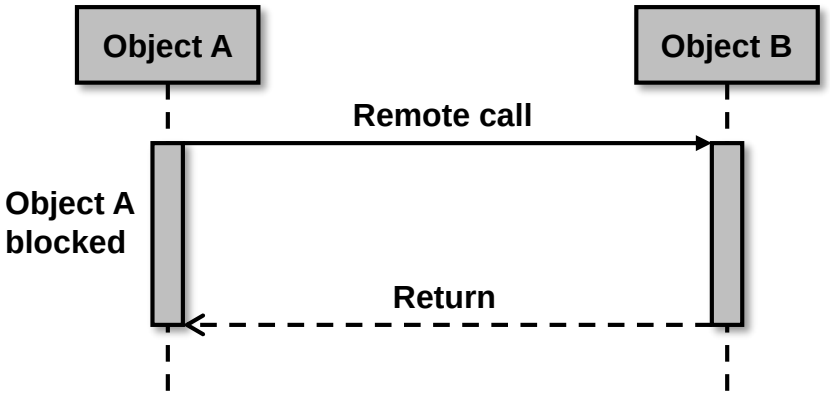


5. Interaction model (request/reply, publish/subscribe):

The interaction model defines the way how the local and remote parties interact.

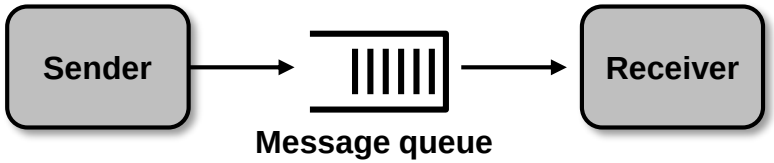
There are 2 main models:

a. Synchronous request / reply



b. Asynchronous messaging

The receiver receives messages independently from the sender.



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	→ IIOP
Web services	→ SOAP / XML over HTTP
RPC	→ 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	→ IDL (Interface Description Language)
Web service	→ WSDL (Web Service Description Language)
RPC	→ 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.

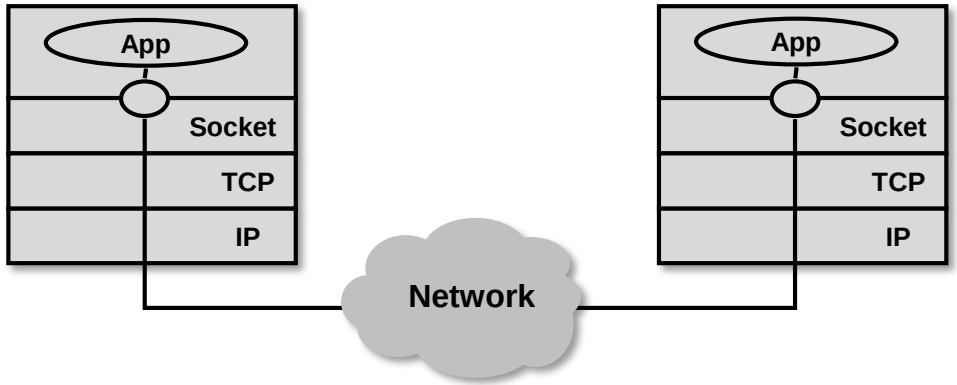
Example platform dependent MW:	JMS, RMI (both use the Java platform)
Example platform independent MW:	CORBA, web services (.Net and Java web service client and server interoperate)

Classification of middleware (1/5)

The following is a simple classification scheme for middleware technologies.

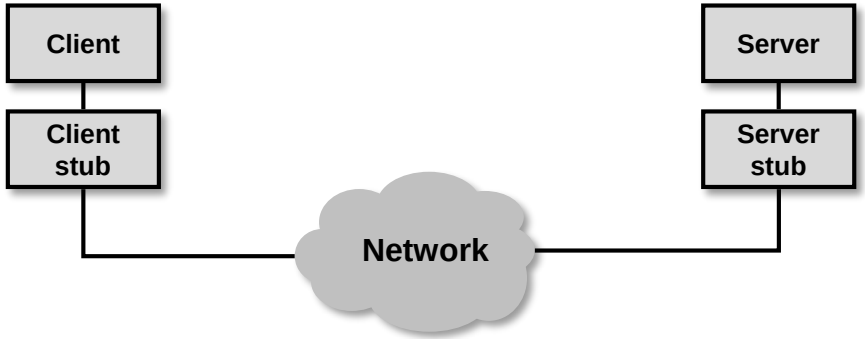
1. Plain old sockets:

Sockets are the basis of all other middleware technologies.



2. RPC – Remote Procedure Call:

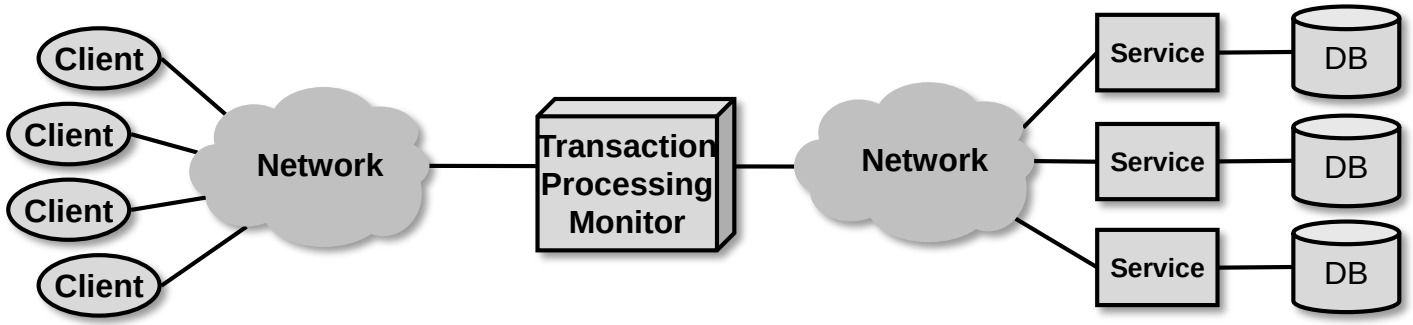
RPC technologies provide a simple means of distributing application logic on different hosts.



Classification of middleware (2/5)

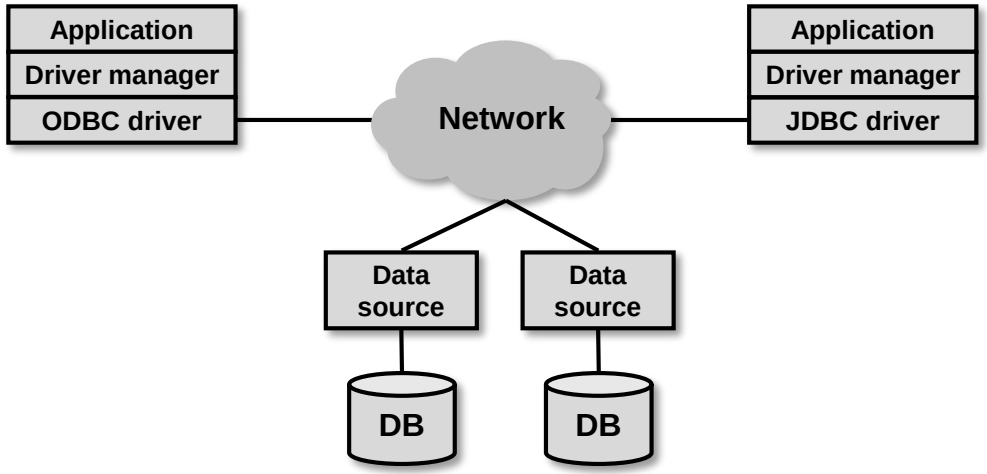
3. TPM - Transaction Processing Monitors:

TPMs are a specialty MW targeted at distributed transactions.



4. DAM - Database Access Middleware:

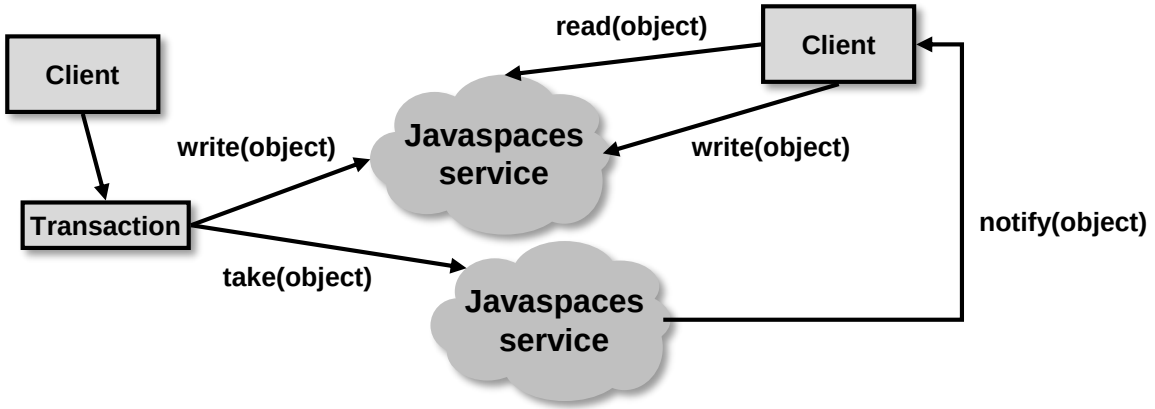
Databases can be used to share and communicate data between distributed applications.



Classification of middleware (3/5)

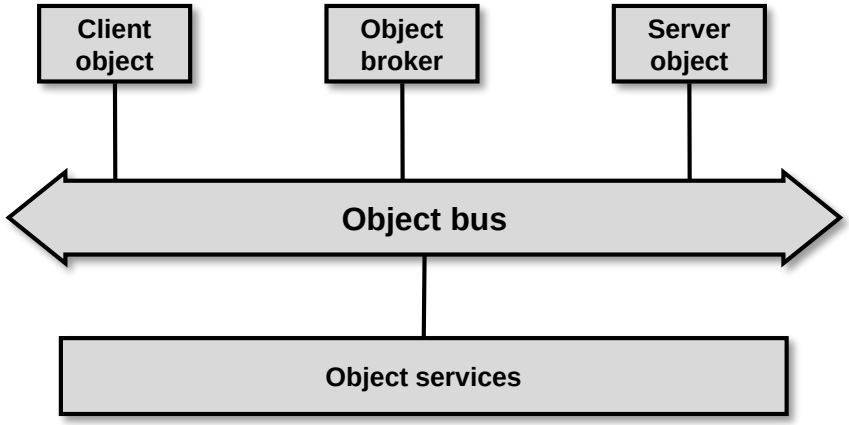
5. Distributed Tuple:

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



6. DOT (Distributed Object Technology) / OOM (Object Oriented Middleware):

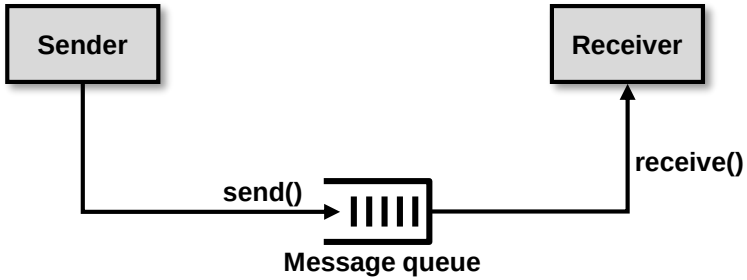
DOT extends the object-oriented paradigm to distributed applications.



Classification of middleware (4/5)

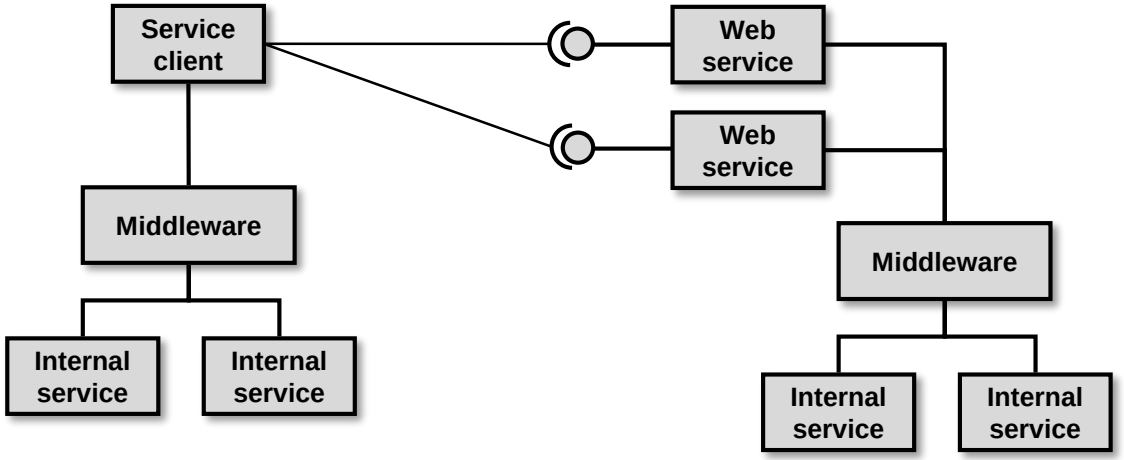
7. MOM (Message Oriented Middleware):

In message oriented middleware, messages are exchanged asynchronously between distributed applications (senders and receivers).



8. Web services:

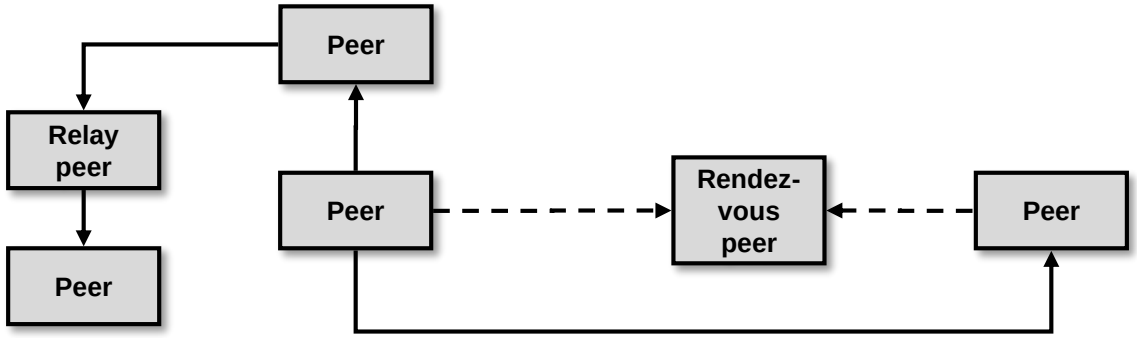
Web services expose services (functionality) on a defined interface, typically accessible through the web protocol HTTP.



Classification of middleware (5/5)

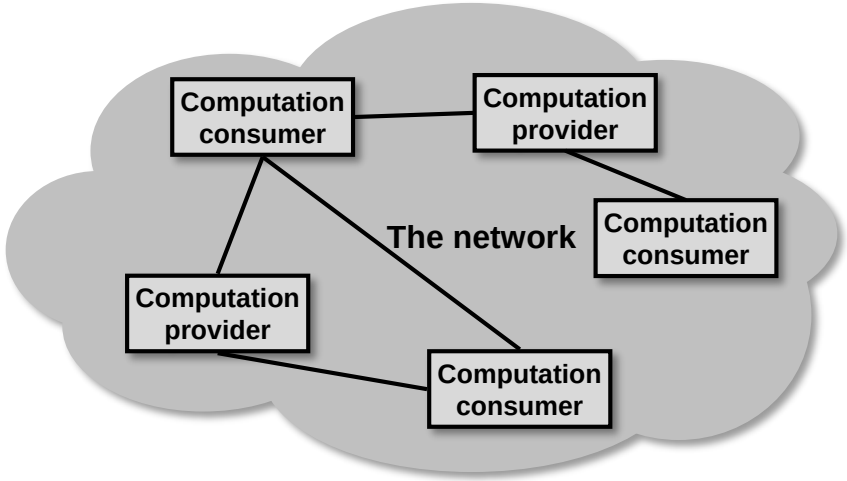
9. Peer-to-peer middleware:

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.



Comparison of middleware technologies (1/2)

Comparison of some concepts of middleware technologies

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

Comparison of middleware technologies (2/2)

Description of common middleware services (features):

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

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.