4 Distributed programming

4.1 Layers vs Tiers

Layers: logical division of software components that allow you to have clear separation of concern between the different functionality.

4.1.1 3-Layers

- 3-layers:
  - Resource management layer:

    deals with and implements the different data sources of an information system, indipendently of the nature of these data ources.(also knows as data layer to indicate that it is create using a DBMS); it storage, index, and retrieve the data necessary to support the application logic.

    example: in Facebook the are so many info to store such as, data, photos, video and so on, so you can imagine that there are some multimedia management systems or filesystem etc., not just a DBMS!
  - Application logic layer:

    determines what the system actually does. It takes care of enforcing the business rules and establish the business processes.

    example: is the layer where are implemented the algorithms, ex facebook: algorithm for match preferences.
  - Presentation: to communicate with external entities, in charge to present the result of computation and of the data management to the user; allow to interact with the system; it may en impl as a UI, or a module that formats data;
  - Client: Entities, interacting from external, that use the services (e.g. it can be a web browser, a web server)

it can be that presentation and client are merged (typical of client/server)

4.1.2 Information Systems design

Top-down

whe designing an information system an approach is to go top-down

start by defining the funtionality of the system from the point of view of the clients and of the interaction.

doesn't imply that the design starts fro UI

design can almost be completely driven by the functionality the system will offer. one the top-level goals are defined, the logic can then be designed, at the end, the resources for the appl. logic

it focus on design top-level goals and then what is required to achieve these goals.

thos type of design is used in homogeneous computing environments (tightly coupled: functionality of a component depends on functionality implemented on other components).

Functionality cannot be used separately

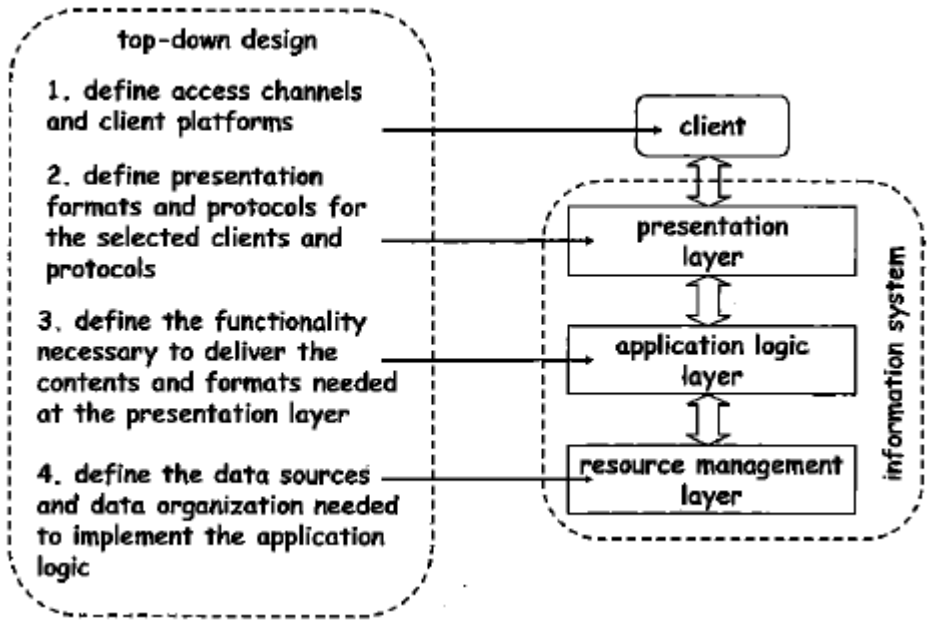
The design is component-based, but the components are not stand-alone (e.g. parallel DBMS)

advantages:

emphasizes the final goals of the system and can be tailored to address both functional and non-functional issues.

drawback:

it can only be applied to systems entirely developed from scratches (nowadays few systems are design top-down).



Bottom-up

occur from necessity

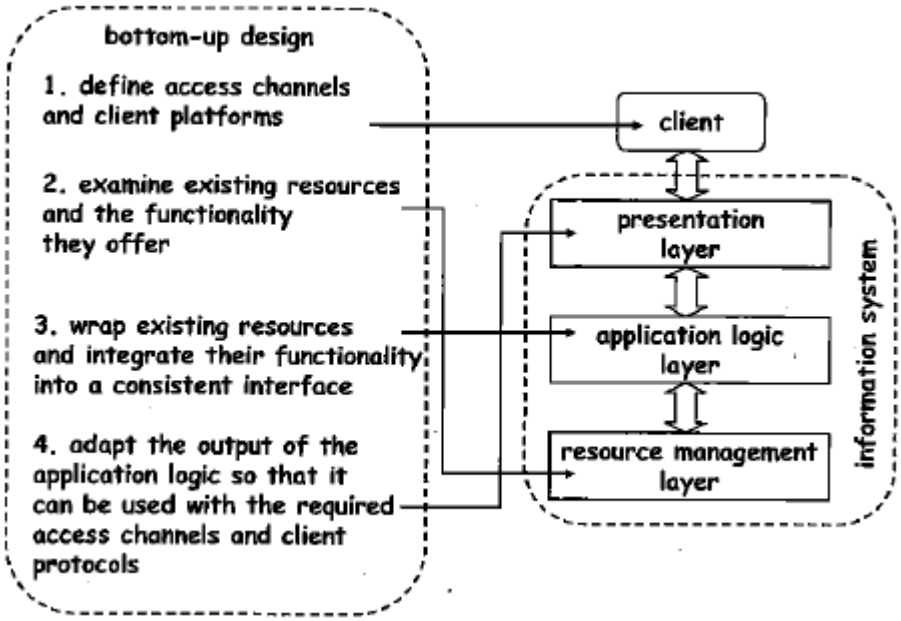
nowadays, IS are built by integrating existing systems (legacy apps or systems). An app become legacy when it is used for a purpose different from the on it has been developed for. the problem is how to integrate functionality in a coherent way.

In this case, reimplement functionality to adapt them to the system is not a good idea.

designers start defining top-level goals, but unlike the top-down, now the next step is to control the resource management to figure out the cost and feasibility.

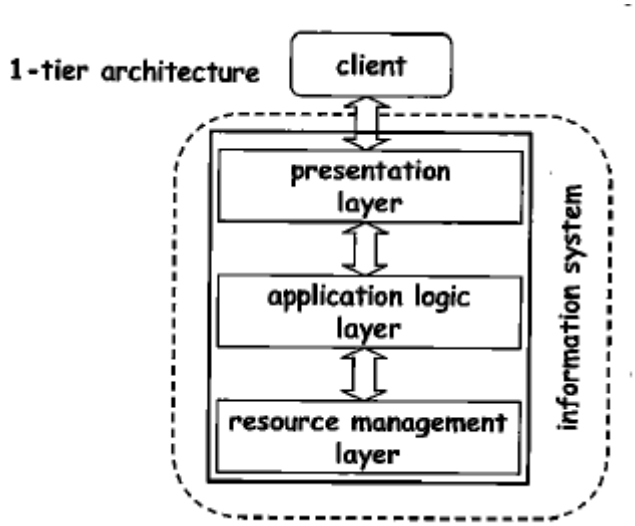
then the components are wrapped so that are ready for the application logic.

we don't talk about advantages and disadvantages, cause most of times there is no choice and we have to use bottom-up



4.1.3 Architecture of IS

- 1-tier:



The tier is the physical element, typical architecture of the legacy system, in which all the 3 layers are deployed in a mainframe(research more) machine.

All is centralized,managing and controlling resources is easier.
The design can be highly optimized by blurring the separation between layers. No forced context switches.

main concern ⟶ efficient use of CPU

interaction with the system was through dumb terminals(clients).

The mainframe controls every aspect of the interaction with the client

Canonical example of legacy system

They can only be treated as black box

to interact with other systems the most-used method is *screen scraping*.

the metod is based on a program that poses as a dumb terminal, it simulates a user and try to parse the screen produced by the system.

it's not elegant and not even cheap (cause highly as hoc).

advantages:

designers are free to merge layers as they want

can use low-level optimizations

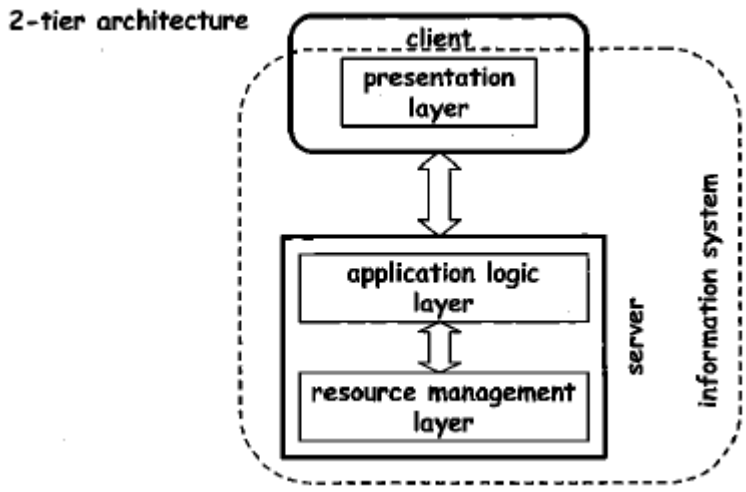
zero client development, deployment and mantainance cost.

drawbacks:

monolithic piece of code

systems are no long monolithic since mainframe have been relegated to critical aspects of the system.

- 2-tier:



Start to have computed station on "desk" not more mainframe of entire room.

For designers was no longer necessary to maintain the presentation layer with the rest of the system, it could be moved to the client.

advantages:

the presentation can use the power of a PC, freeing up resources on the mainframe.

It becomes possible to adapt the presentation for different purposes without increasing the complexity.

e.g. in client/server system

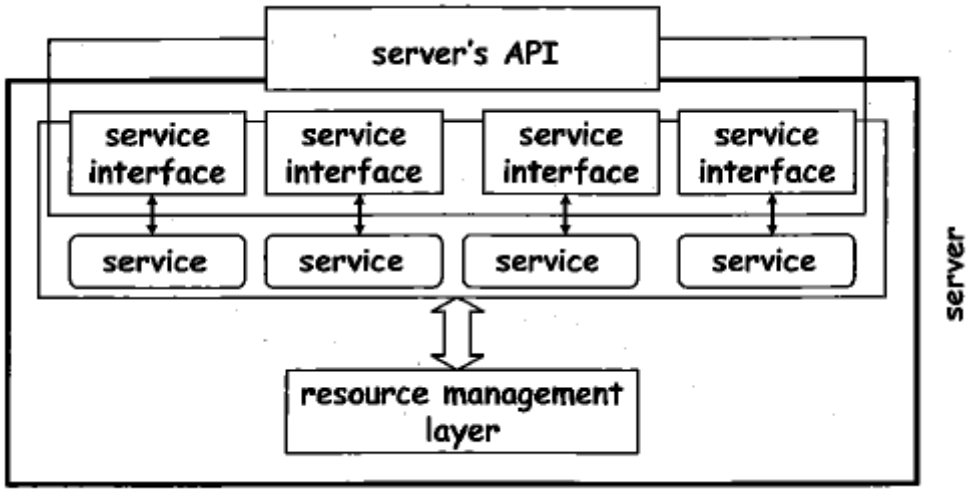
differentiation for thin client and fat client depending on the complexity.

drawback:

they have a large footprint, require considerable resources

The develop of this design resulted in the developing of API (specifies how to invoke a service, the responses and what effects the invocation will have on the internal state of the server).

the collection of service interfaces made available to outside clients became the server's API.



Interfaces and services

when the client invoke the application logic operation, you have to cluster them in order to have specific modules.

Services are the modules that permit you to invoke that functions.

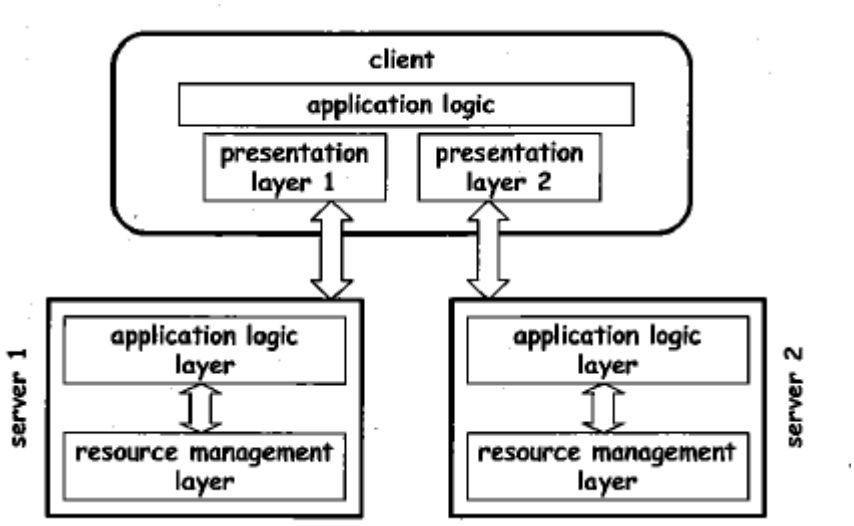
Services synonymous of set of functionality offered by someone in order to be invoked by someone else (layers on client machine).

advantages over 1-tier: keeping the app logic and the resources management togheter, the system is faster and don't need context switching or call between components.

Since there is one server, there is a problem when too many clients want to connect, cause server would maintain the context active, so 2-tier has limited scalability.

The problem starts thinking about connecting to multiple servers.

In this case a client should be responsible of integrating and managing the multiple connections and presentation layers



Clients are independent of each other: in each is managed the presentation layer. It introduces the concept of API (Application Program Interface). An interface to invoke the system from the outside.
The resource manager only sees one client: the application logic. There are no client connections/sessions to maintain.

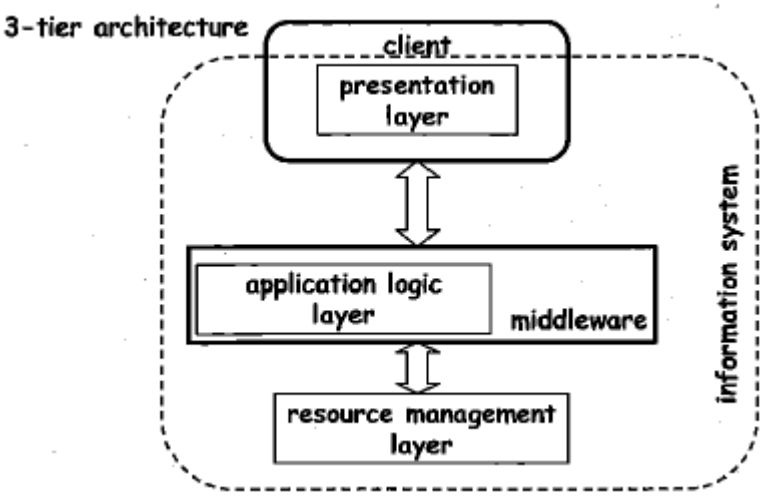
- 3-tier arch:

  The introducing of LANs and the will of communicate with other server implies the development of a new architecture, the 3-tier.

  An additional layer between client and server

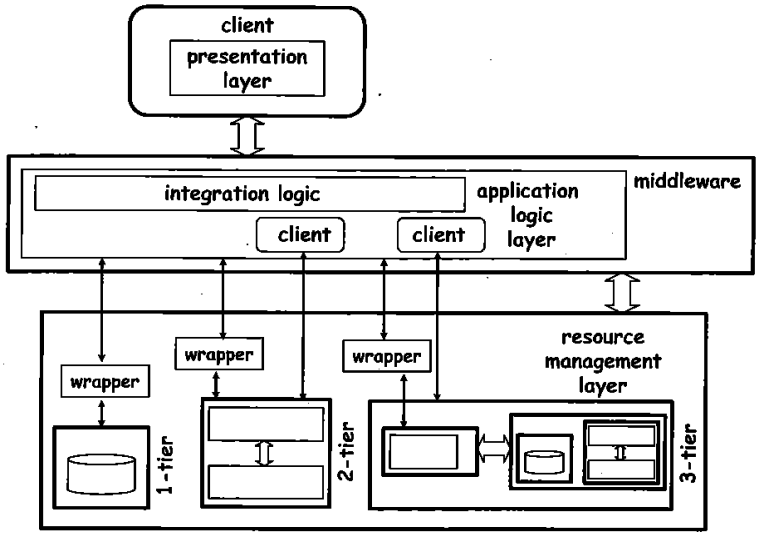
  introduction of **middleware**: a set of technologies that are in charge of managing the communication and allows you to focus on the pure application logic without pay too attention on the communication aspect.

It introduces an additional layer of business logic/application logic.



the presentation is in the client

the application logic is in the middle level(middleware)



introduction of stored procedures: procedures implemented in the database, the application logic is part of the database (TP-lite systems)

With 1 server, 2-tier is more efficient than 3-tier

by using a middleware system, the designer of the application logica can rely on the support provided by the middleware to develop sophisticated interaction models without havintg to implement eertything from scratch.

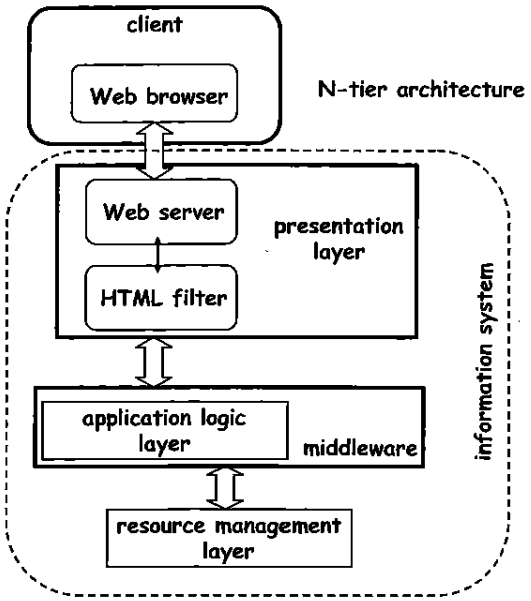
Main advantage: provide an additional tier where the integration logic can reside, the performance loss is compensated by the flexibility

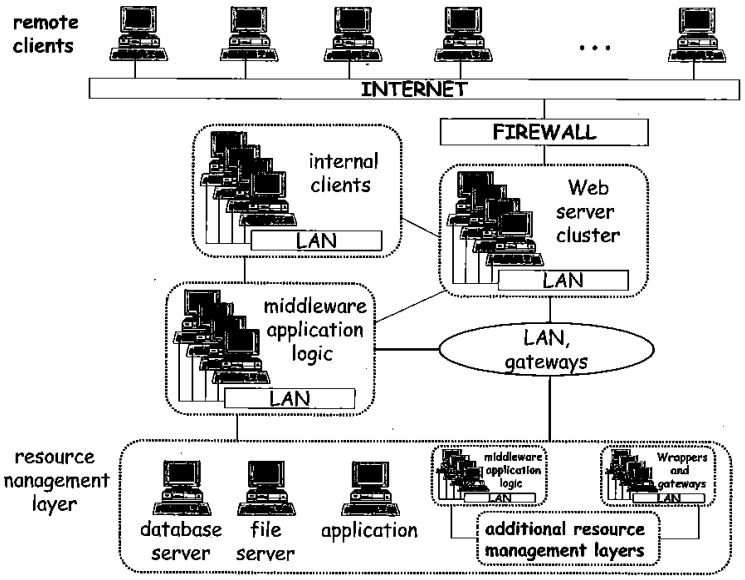
disadvantages:

It's a problem integrating accross the internet between different 3-tier systems

N-tier arch:

N-tire arise with the need to incorporate the web server as part of the presentation





the html filter translates the data provided by the app logic layer into html pages

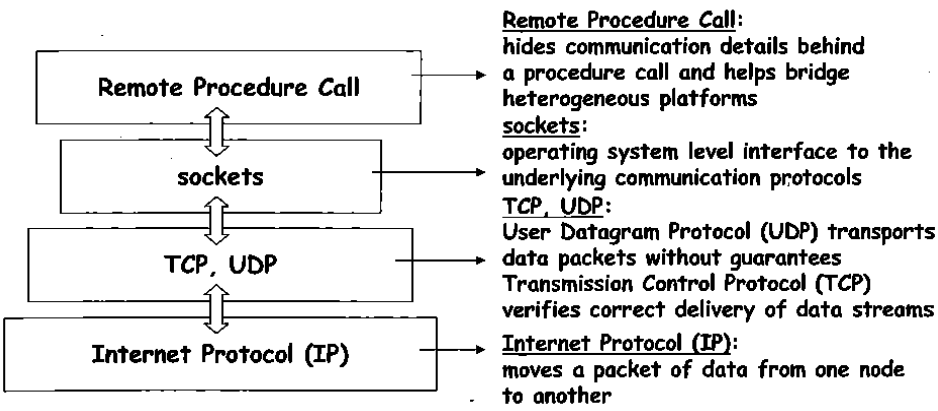
disadvantage: too much middleware, it difficult recognise where a system starts and where end

4.2 Middleware

facilitates and manages the interaction between apps across heterogeneous computing platforms.

RPC is a programming abstracion that hidesthe communication channel begind an interface that looks exactly like a normal procedure call.

With RPC, the only thing we ned to do is to reformulate the comunication between the 2 parts of the application as procedure calls.

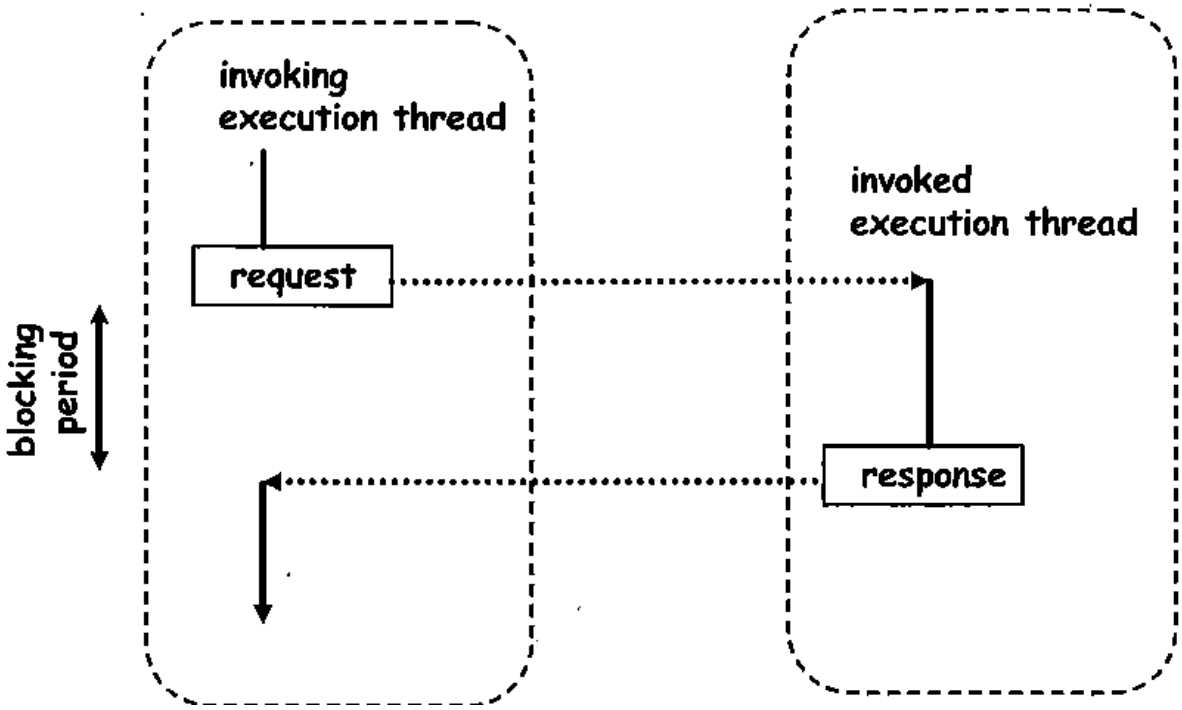


Allows us to develop distributed programs that generates for us all the communication code needed.

Middleware is primarily a set of programming abstractions developed to facilitate
the development of complex distributed systems
– to understand a middleware platform one needs to understand its programming
model
– from the programming model the limitations, general performance, and applicability of a given type of middleware can be determined in a first approximation
– the underlying programming model also determines how the platform will

evolve
and fare when new technologies evolve
• The infrastructure is also intended to support additional functionality that makes development, maintenance, and monitoring easier and less costly
– RPC ⇒ transactional RPC ⇒ logging, recovery, advanced transaction models, language primitives for transactional demarcation, transactional file system, etc.
– The infrastructure is also there to take care of all the non-functional properties typically ignored by data models, programming models, and programming languages: performance, availability, recovery, instrumentation, maintenance, resource management, etc.

4.2.1 Communication primitives



- blocking-interaction (improperly synchronous)

  The client sends a request to a service and waits for a response of the service to come back before continuing doing its work).

  Invoking and invoked contemporary online; can have connection issues; If the client or the server fail, the context is lost and resynchronization might be difficult.

  Synchronous interaction requires a context for each call and a context management system for all incoming calls. The context needs to be

passed around with each call as it identifies the session, the client, and the nature of the interaction.

disadvantages:
– connection overhead
– higher probability of failures
– difficult to identify and react to failures
– it is a one-to-one system

ex. in C main calls functions that take the flow control and block any other functino.

ex. online classes, we and the teacher are **blocked** in discuss about the lesson

Solutions:

- Enhanced Support

  Transactional interaction (..guarantees..)

  Service replication and load balancing (prevent failure)

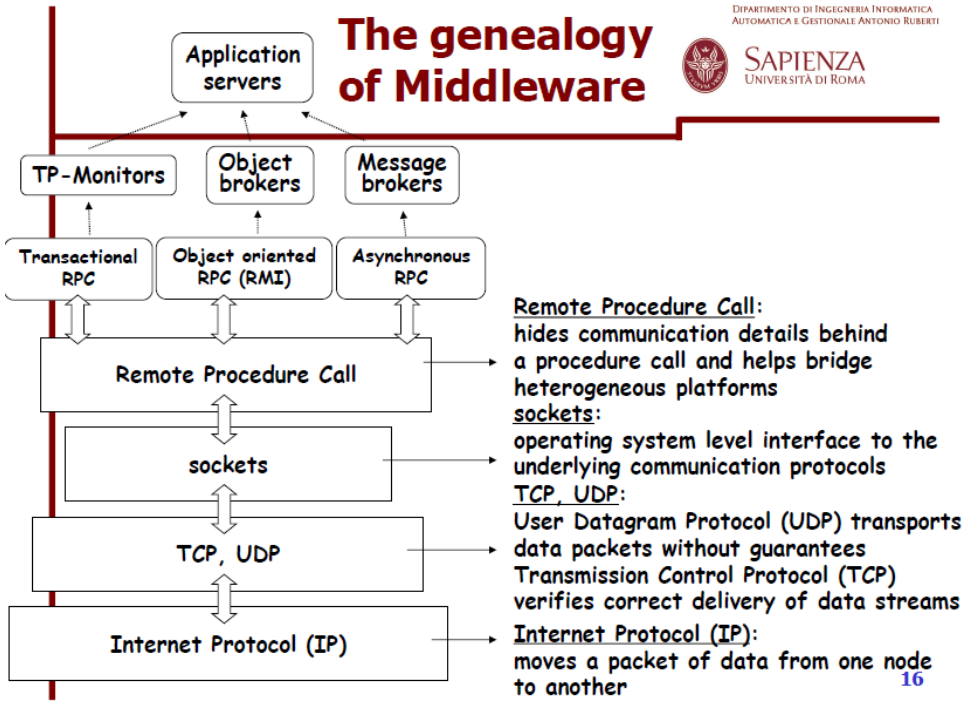
- Non blocking-interaction (asynchronous)

  The caller sends a message that gets stored somewhere until the receiver reads it and sends a response.

  Two forms: - non-blocking (no wait for response)

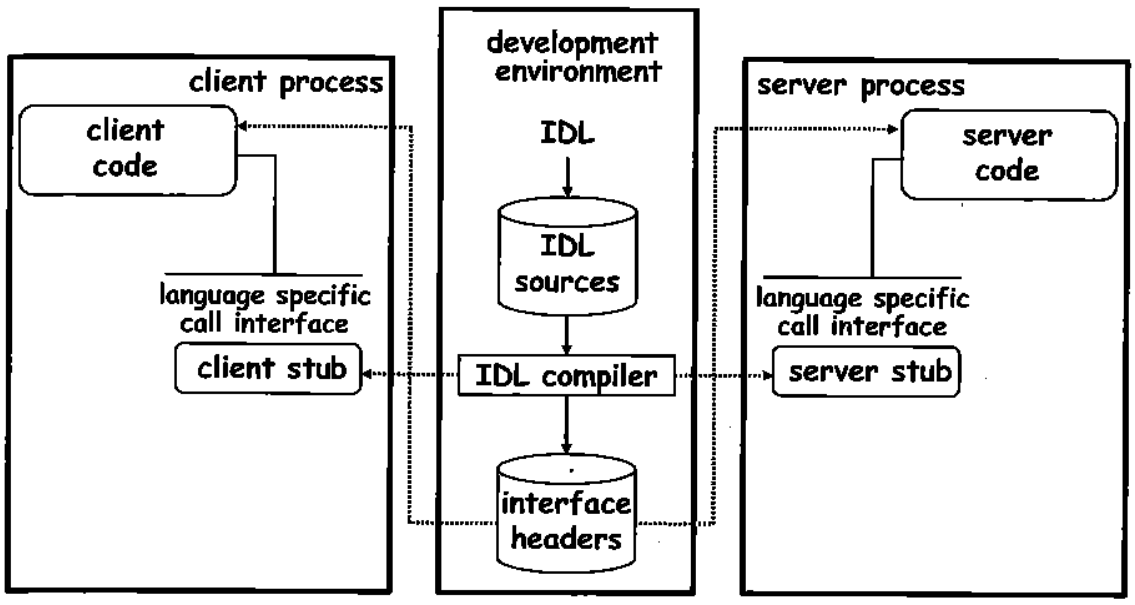
  - persistent queues (stored until they are accessed)

  ex: e-mail



RPC(1): Developing time

First, define the interface using an IDL.



Compiling the IDL description we have 2 results:

1) Client stub: piece of code to be compiled and linked with the client.

When the client clls a remote procedure, the call actually executed is a local call to the procedure provided by the stub.

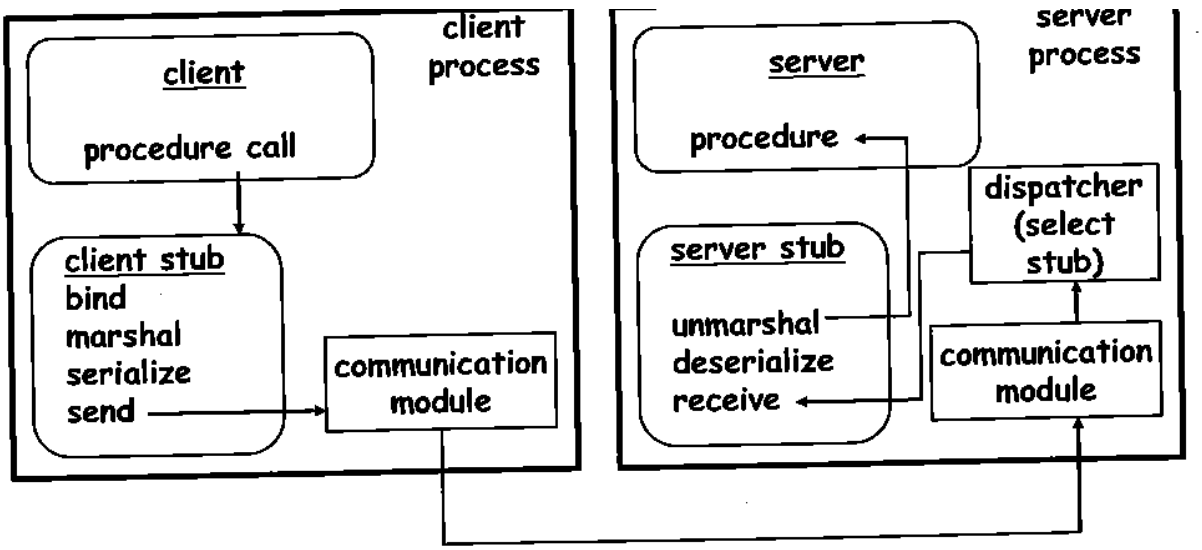
The stub makes the procedure appear as a normal local procedure.

2) Server stub: implements the server side of the invocation. it contains the code for receiving the invocation from the client stub

*marshaling involves packing data into a common message format prior to transmitting the message over a communication channel.

As with the client stub, it must be compiled and linked with the server code.

Basic functioning



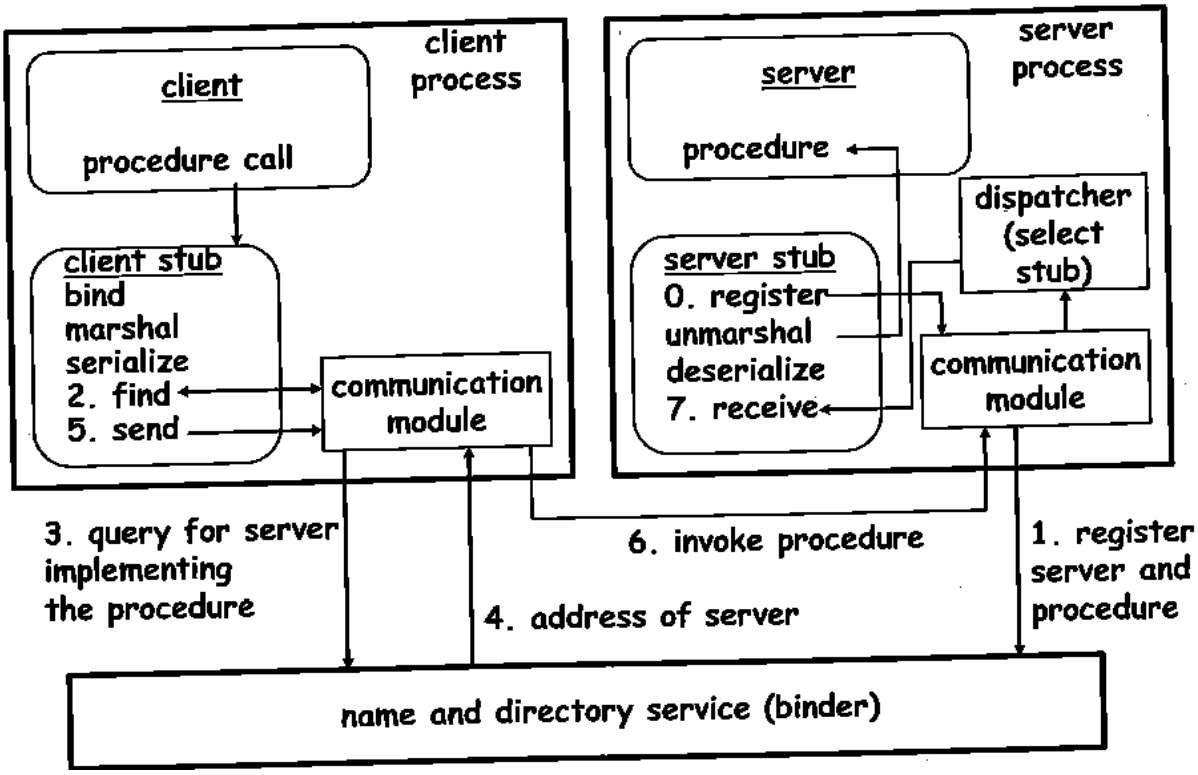
- Binding in RPC

The process whereby the client creates a local association for a given server in order to invoke a remote procedure.

Binding can be:

- static: advantages: simple and efficient, no addition infrastructure; disadvantages: client and server tightly coupled.

- dynamic: enables clients to use a specialized service to locate appropriate servers, adds a layer of indirection, generally called *name and directory server;* is capable of decouple client and server, that adds flexibility, the cost is additional infrastructure



4.3 Transactional RCP

Extend the RPC protocol with the ability to wrap a series of RPC invocations into a transaction(a set of operation with ACID (atomicity, consistency, isolation, and durability)).

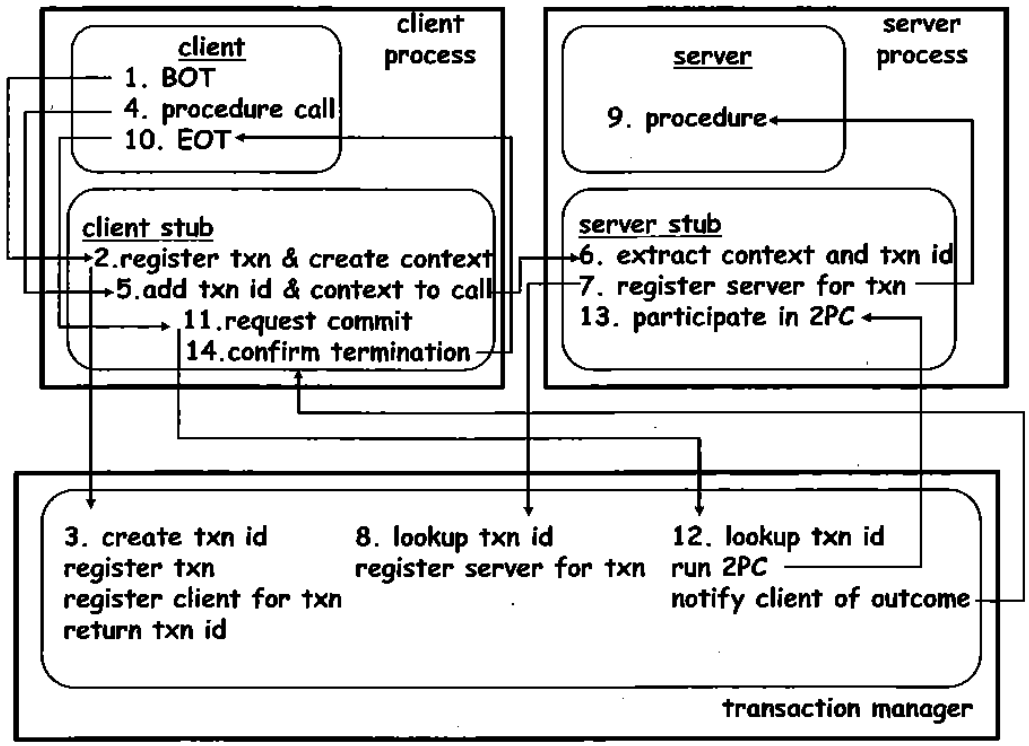
Tre transactional abstract implies all operations executed or none.

BOT and EOT are treated as one unit, achieved by a transaction management module.

BOT encountered, the cient stub contact the manager to obtain the trans id and create the trans conctext. when the client calls one of the servers, the server stub ectract the trans contxt, notifies the manager, when the EOT is reached, the client notifies the manager that initiates a two-phase commit (2PC) protocol between the two servers involved. Then returns if the transaction was succesful or not.

2PC: first phase: it contacts each server involved by sending a preparee to commit message.

> second phase: the manager examines all the replies obtained and, if all of them are ready to commit, it then instructs each server to commit the changes performed as part of the invoked procedure.



if a participant declare an abort, the manager declare abort to all the partners, the retry untill recceive an ack, if a participant go down, manager will retry indefinitively → starvation

optmistic approach:

assume that prob the most operation conclude in commit, if an abort, the interface of the servant should offer the possibility to undo.

a programmer should offer an undo for every operation with side effect

pessimistic:

the most of the case will end in abort. The operation is done in a way that when an operation is asked this is no executed, but in buffer

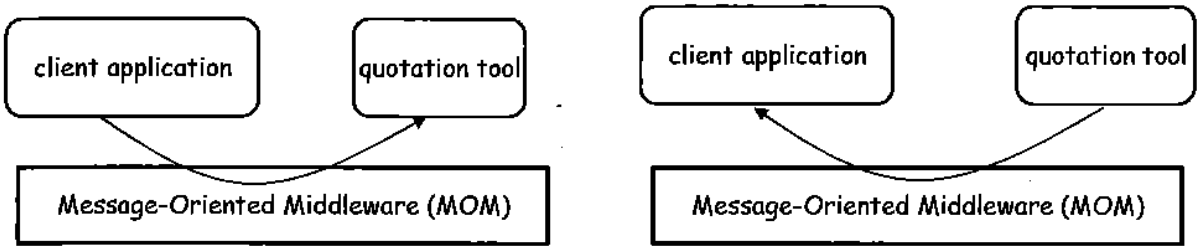
4.4 Message oriented Middleware

idea: message base interoperability refers to an interaction paradigm where client and service providers communicate by exchanging messages.

A message is a structurd set of type, name and value, nowadays most are XML types.

```
Message : quoteRequest {
 QuoteReferenceNumber: 325
 Customer: Acme,INC
 Item:#115 (Ball-point pen, blue)
 Quantity: 1200
 RequestedDeliveryDate: Mar 16,2003
 DeliveryAddress: Palo Alto, CA
}
```

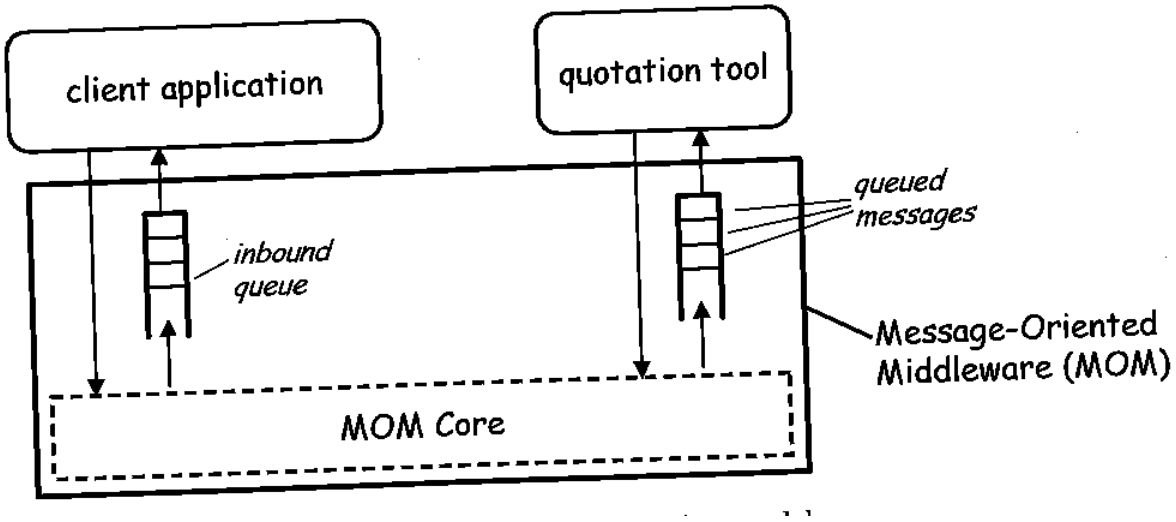
```
Message: quote {
 QuoteReferenceNumber: 325
 ExpectedDeliveryDate: Mar 12, 2003
 Price:1200$
}
```



Once clients and service providers agree on a set of message types, they can communicate by exchanging messages. The class of this middlewares is called message-oriented middlewares (MOM).

4.4.1 Message queues

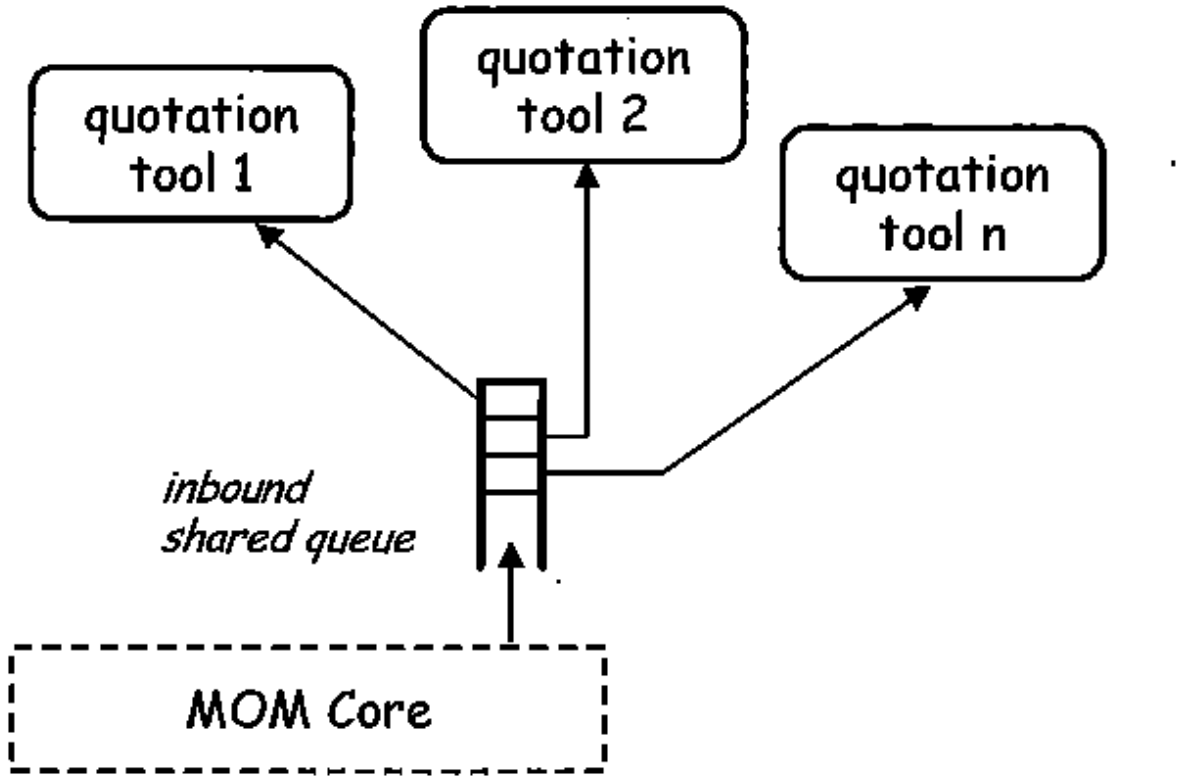
In a message queueing model, messages seny by MOM clients are placed into a queue, identified by a name and bound to a specific recipient.



Benefits: it gives recipients control of when to process messages. Recipients can retrieve a new message only when they can or need to process it.

If an app is down or unable to receive messages, these will be stored (till a specific date) and then delivered when necessary.

Other features: senders can assign priorities to messages.

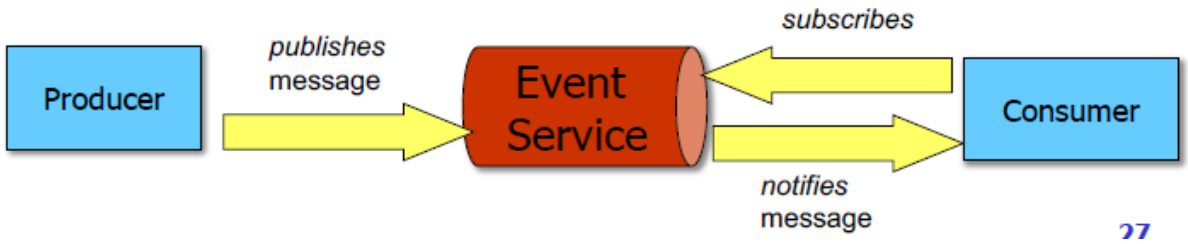


4.4.2 Publish/Subscribe

In this paradigm, apps communicate by exchanging messages, now senders (publishers) just publish (not to a specific recipient) to the middleware that manage.

If an app is interested in receiving messages of a given type, then it must subscribe with the publish/subscribe middleware.

Whenever the publisher sends a message of a given type, the middleware retrieves the list of all apps that subscribed to messages of that type, and delivers a copy of the message to each of them.



- topic-based: info is divided into topics
- content-based: filters can be used for a more accurate selection of info to be received.

4.5 Middleware on the internet

4.5.1 The web

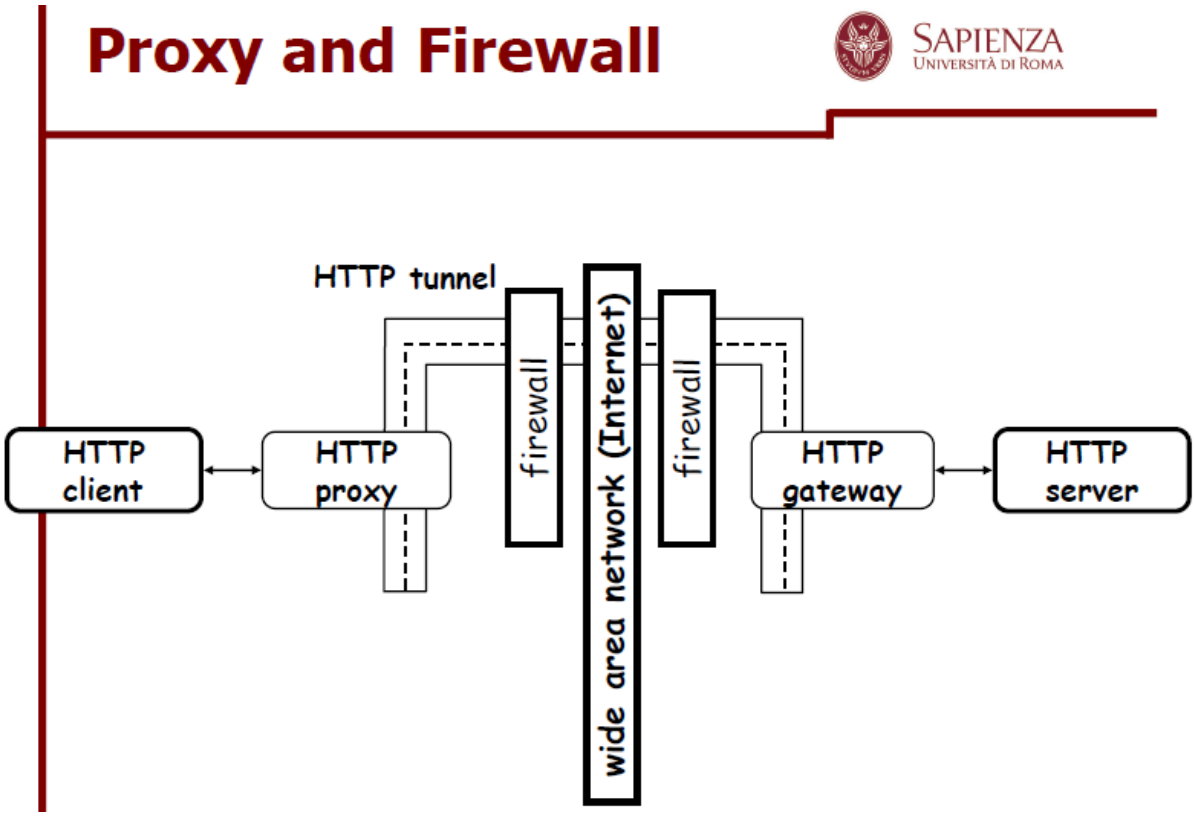
HTTP is a gneric stateless protocol that governs the transfer of files across a network.

It was designed HTML.

Information is exchanged over HTTP in form of documents (statics or dynamics: content is generated at access time), identified by an URI. Every resource accessible over the Web has a URL that identifies the location of the resource and describes how to access it.

HTTP Mechanism: client/server uscing TCP/IP sockets. An HTTP client opens a connection to an HTTP server and sends a request message consisting of a request method, URI, and protocol version followed by a "MIME-like message". The server then returns a response message consisting of a status line, followed by a "MIME- like message" and close the connection.

Typical mehods include also parameters as : OPTION, GET, POST, PUT, DELETE.



As shown, one or more intermediaries, such a a proxy, gateway, or tunnel, may lie between the client and the server.

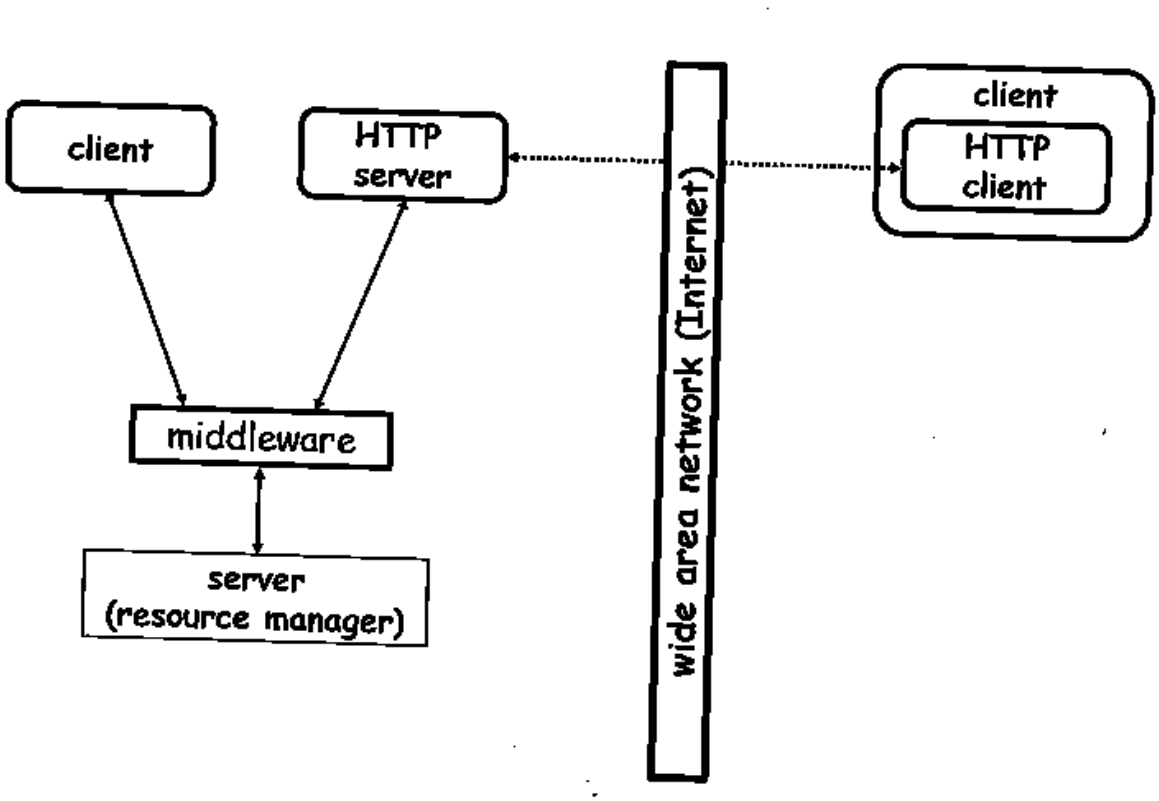
Proxy:"an intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients; a gateway as a server which acts as an intermediary for some other origin server for the requested resource; and a tunnel as an intermediary program which is acting as a blind relay between two connections".

4.5.2 3-tier on the web

Conventional 3-tier operate with a single company.

Users wanting to take advantage of the opportunity would need to have a specialized client for every company they wanted to interact with, which is not a practical solution. Moreover, if the clients were distributed over the Internet, the middleware-based system would have been more complex.

Nowadays, these architectures are implemented not by moving the client to the remote computer, but by letting the remote computer use a web browser as a client.



4.5.3 Applets

Originally web browsers were implemented for stati documents returned by HTTP calls.

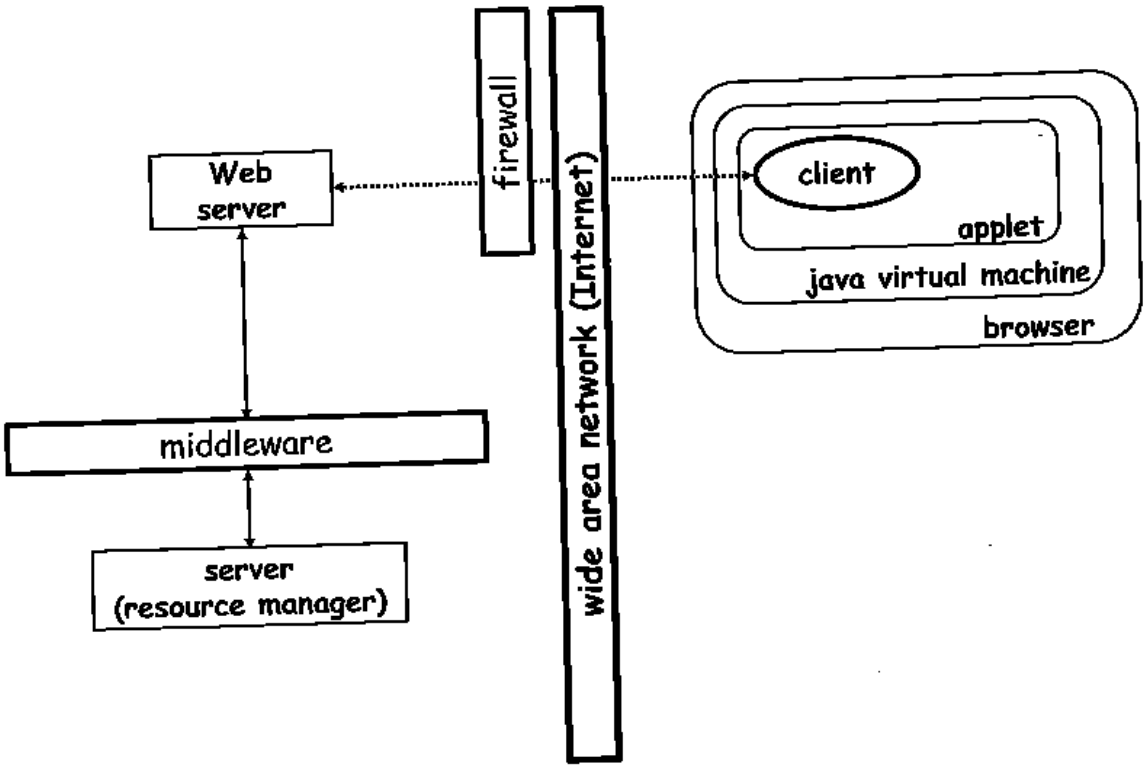
Introduction of Applets: Java programs that can be embedded in an HTML document.

When the document is downloaded, it is executed on the JVM in the browser.

This suffers of the limitation of having to download the code every time the client is used.

For thin clients applets are very used.

Advantage: turn a web browser into an application-specific client without complex configuration or installation procedures.

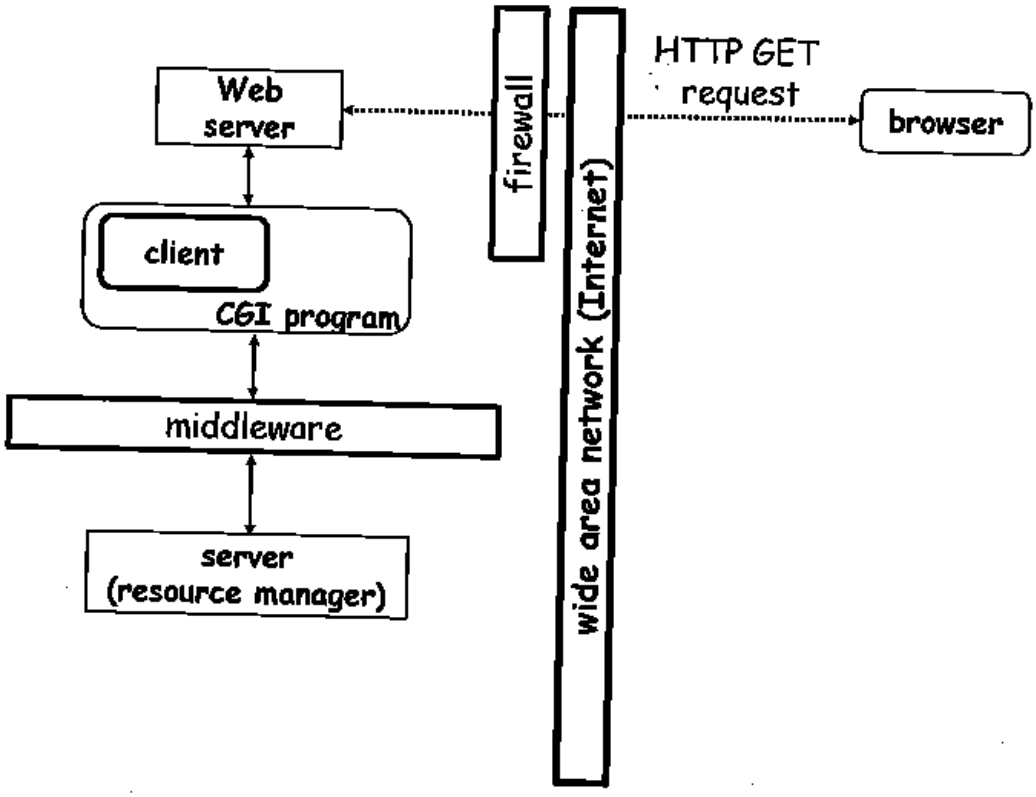


4.5.4 Common Gateway Interface (CGI)

One of the first approaches to make a web server responding to a request by invoking an application that will automatically generate a document to be returned was CGI.

CGI enables HTTP to interface with external apps (as gateways).

CGI assigns a URL, when it is invoked the program is executed. From the URL received, the web server extract any information it may need to pass on to the program.



e.g. : CGI programs often serve as an interface between a DB and a web server. The params of the query are embedded in the URL. The program executes the query, then packs the query results into a HTML document to be returned to the remote browser.

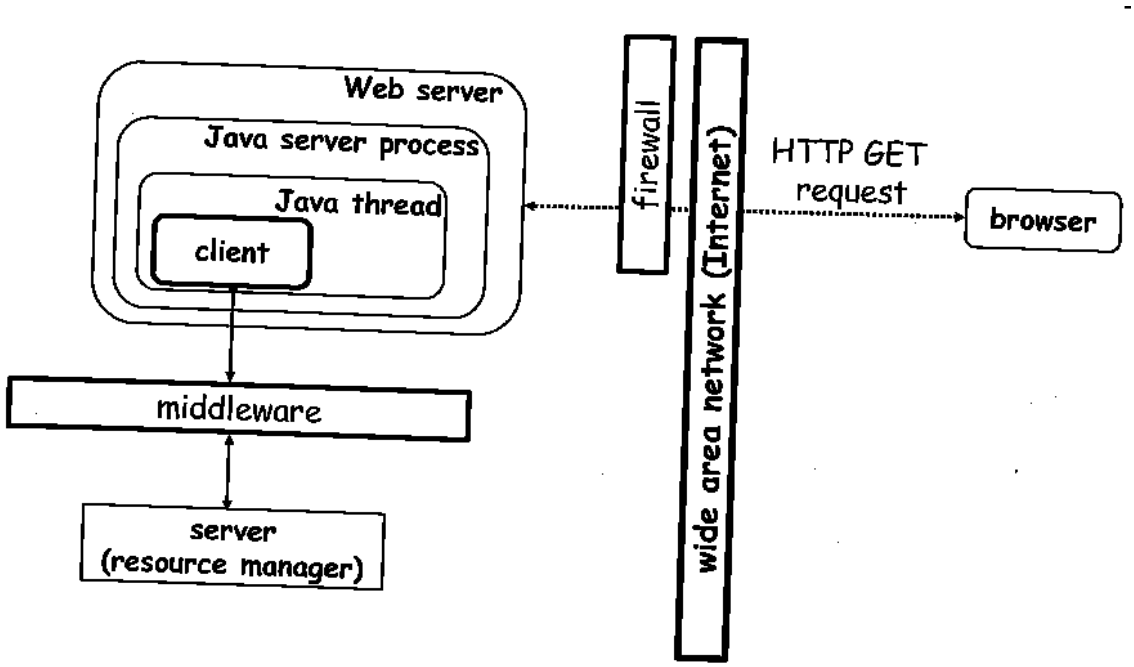
4.5.5 Servlets

CGI involve a certain overhead.

To avoid it, Java servlets can be used instead of CGI programs.

The execution of a servlet is triggered in the same way as a CGI script, but Servlets are invoked directly by embedding servlet-specific information within an HTTP request. Servlets run as threads of the Java server process rather than as independent processes. They run as part of the web server.

Results can be cached so that identical requests from different clients can be answered without actually having to execute the operation again.


Web server
Java server process
Java thread
client
middleware
server
(resource manager)
firewall
wide area network (Internet)
HTTP GET
request
browser