



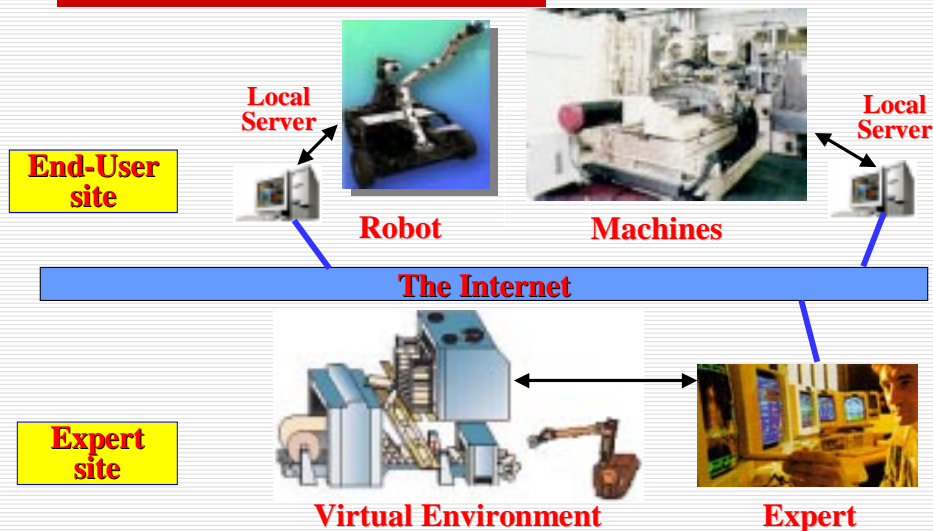
UNIVERSITA' DEGLI STUDI DI BERGAMO  
Facoltà di Ingegneria

## Informatica Industriale

Prof. Davide Brugali

### 2.1 – Modelli e paradigmi di distribuzione

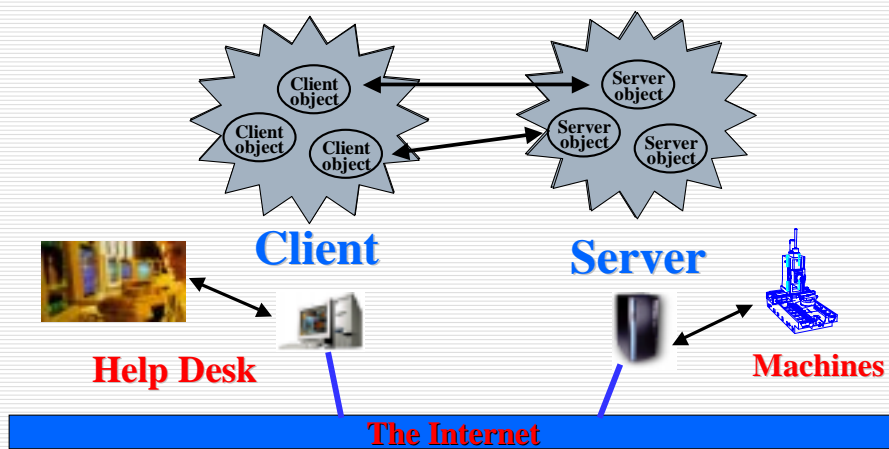
## Ispezione e Manutenzione Remota



## Architettura distribuita

- L'architettura di un sistema distribuito è descritta in termini di :
  - **Componenti** (le applicazioni o le loro parti)
  - **Connessioni** (le relazioni tra i componenti)
  - **Tipi di connessioni** (i meccanismi usati per realizzare le connessioni)

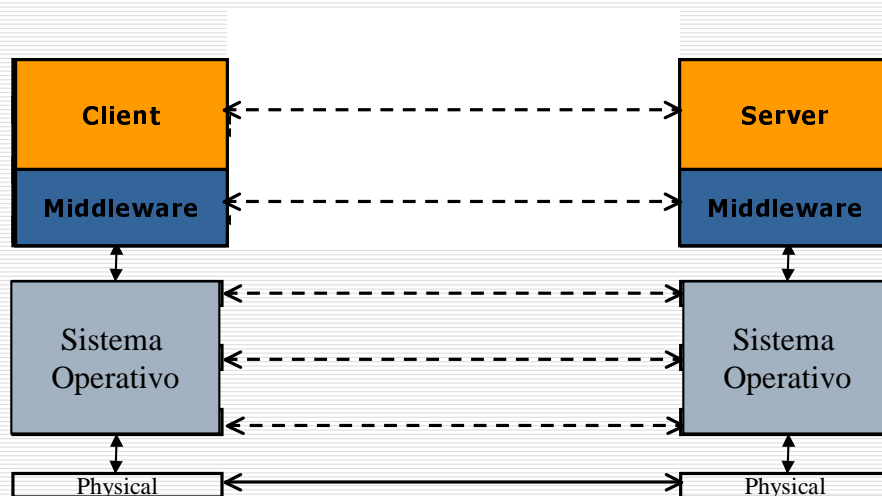
## Architettura Client / Server



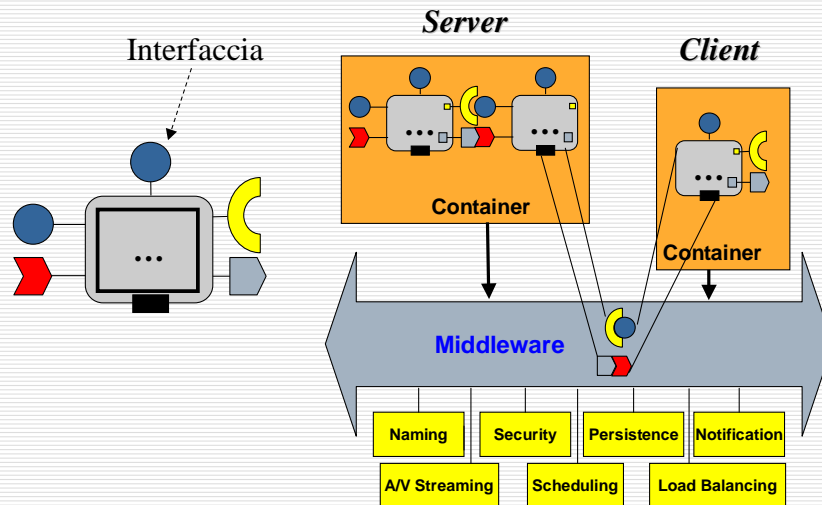
## Client / Server

- Nella programmazione Orientata agli Oggetti le interazioni tra due oggetti sono implementate come operazioni di un oggetto sull'altro:
  - L'oggetto che esegue le operazioni si chiama **Server**
  - L'oggetto che richiede le operazioni si chiama **Client**
- Quando sia il Client che il Server risiedono nello stesso spazio di indirizzamento (sono eseguiti sullo stesso computer), il Client ha un riferimento (puntatore) al Server per invocare le sue operazioni.
- Quando il Client e il Server risiedono su computer remoti collegati in rete, è necessario utilizzare un'infrastruttura software di interconnessione (middleware) per consentire lo scambio di dati tra Client e Server.

## Il modello ISO-OSI



# Modello a componenti



## Requisiti per un modello a componenti

- ❑ **Integrazione di sistemi pre-esistenti (legacy)**
  - Accesso ai dati
  - Accesso alle funzionalità
- ❑ **Persistenza**
  - Salvataggio dati su Database in modo trasparente
- ❑ **Transazioni**
  - Supporto trasparente all'esecuzione di operazioni atomiche
- ❑ **Distribuzione**
  - Supporto alla comunicazione di dati ed eventi
  - Supporto all'esecuzione remota di procedure
- ❑ **Web integration**
  - Supporto alla generazione dinamica di documenti Web
- ❑ **Scalabilità**
  - Multi-threading
- ❑ **Robustezza**
  - Replicazione dei dati

## Requisiti specifici per la distribuzione

---

### 1. Interoperabilità

I componenti devono poter comunicare anche se:

- vengono sviluppati in linguaggi e S.O. diversi
- utilizzano protocolli di rete diversi

### 2. Location transparency

I componenti devono poter comunicare anche se:

- si trovano su computer diversi
- cambia l'indirizzo di rete del computer
- vengono spostati su altri computer

## Requisiti specifici per la distribuzione

---

### 3. Separazione tra interfaccia e implementazione

- Un'interfaccia è un insieme di servizi.
- L'interfaccia è un contratto tra il componente e l'applicazione che la vuole usare.
- I Componenti accedono l'un all'altro attraverso le loro interfacce.
- Componenti non chiamano mai direttamente l'implementazione degli altri.
- Il componente può cambiare l'implementazione, purché il contratto dell'interfaccia sia rispettato
- L'interfaccia di un componente deve poter essere scoperta dinamicamente

## Problemi della distribuzione

---

### ❑ Inconsistenza dei tipi di dati

- Linguaggi di programmazione diversi e computer diversi rappresentano gli stessi tipi in modo diverso
- Integer : 16, 32, 64, 128 bit oppure è un Oggetto
- Alcuni linguaggi non hanno il tipo "Data"

### ❑ Soluzione:

- Occorre definire una rappresentazione standard e le regole di conversione

## Problemi della distribuzione

---

### ❑ Metadata

- Sono informazioni sulle definizioni dei tipi di dati usati da un'applicazione
- Per esempio, la definizione di una classe (attributi e metodi)
- Queste informazioni vengono utilizzate dai compilatori, ma non vengono mantenute nel codice compilato.
- I metadata sono importanti per la "location transparency"
- Alcuni linguaggi consentono di accedere ai metadata anche a run-time. Per esempio Java offre il meccanismo della Reflection
- CORBA definisce le interfacce in IDL

# Problemi della distribuzione

---

## □ Supporto all'esecuzione

- Indipendenza dal linguaggio di programmazione: due componenti possono interoperare anche se implementati in linguaggi diversi (.NET)
- Indipendenza dal sistema operativo: un componente può essere eseguito da computer con sistemi operativi diversi (Java Virtual Machine)



□ UNIVERSITA' DEGLI STUDI DI BERGAMO  
Facoltà di Ingegneria

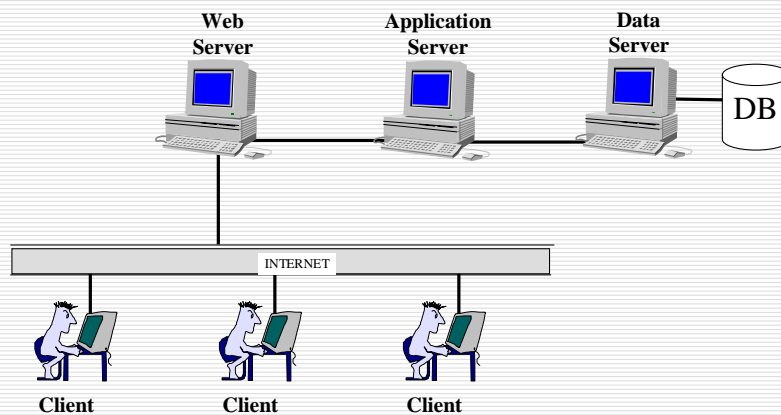
## Modelli ieli

---

## Client / Server

- ❑ Designing a client/server system requires the system's functionality to be split between the clients and the servers:
  - **Fat servers** (the server manages both the data and the processing) ?
  - **Fat clients** (the server manages only the data) ?
- ❑ It is a matter of how many clients share common data and operations on them and has implications on the system scalability, portability, and efficiency

## Three-Tiers Client/Server





## Three-Tiers Client/Server

---

- ❑ The Three-Tier architecture improves
  - system's robustness,
  - modularity, and
  - Dependability
- ❑ It decouples the independent functionality of a Client/Server system.

## Broker-based

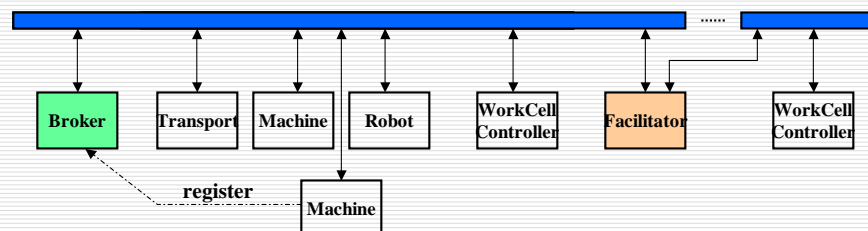
---

- ❑ In an open distributed environment, like the Internet, resources (the servers that manage them) are free to appear and disappear and the mapping of client requests to server services is highly dynamic.
- ❑ The problem is finding what service is available from which server. Since it is not feasible to broadcast on the Internet the information that a resource has appeared, specific components take on the role of broker.

## Broker-based

- A broker represents a mediation level between clients and servers. According to the information that it manages, the broker acts as:
  - **Domain Name Server** (it knows the name and Internet location of a group of server objects)
  - **Machmaker** (it knows the service offered by the server objects)
  - **Facilitator** (it knows the policies and protocols to access the services of the server objects)
  - **Mediator** (it offers new services as composition of other servers' services)

## Broker-based



## Multi-Agent

---

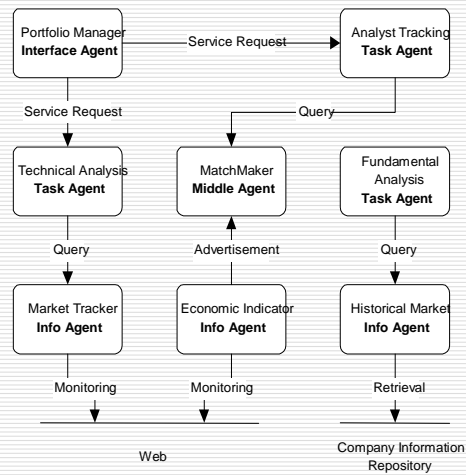
- ❑ Software Agents are integrated systems that incorporate major capabilities drawn from several research areas: Artificial Intelligence, Databases, and Operational Research to name a few.
- ❑ They are usually implemented as software objects that may be customized and made up with other similar objects to build complex multi-agent systems.

## Multi-Agent

---

- ❑ Two aspects characterize multi-agent systems.
  - **Task-based roles.** There is not a clear-cut distinction between clients and servers. The roles are determined by the specific task assigned to the system.
  - **Task-oriented relationships.** Software agents formulate problem-solving plans and carry out these plans through querying and exchanging information with other software agents. They react to external or internal events, or they perform autonomous activities such as monitoring the external environment.

# Multi-Agent



2.1 - Modelli di Distribuzione

Informatica Industriale - Prof. Davide Bruggali

23/41



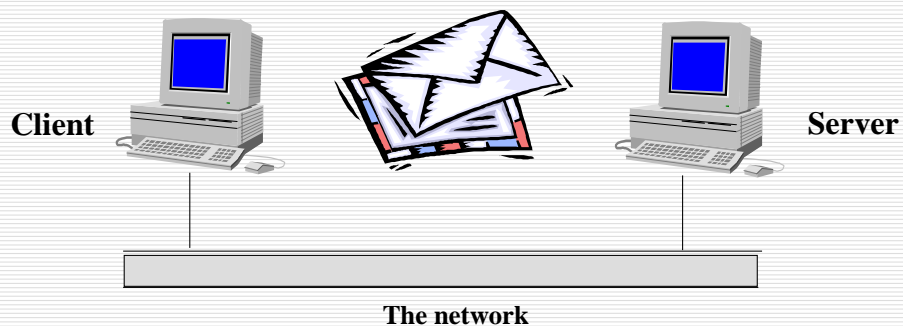
UNIVERSITA' DEGLI STUDI DI BERGAMO  
Facoltà di Ingegneria

## Paradigmi di distribuzione

## Paradigmi di distribuzione

- Information sharing across a network assumes several forms at different levels of abstraction.
- In this context, abstraction means the capability of accessing distributed resources in a ***transparent way*** with regards to their physical location, data format, structure, implementation language, and execution support.

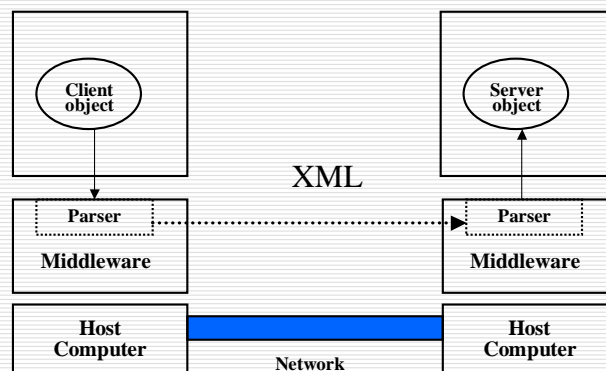
## Scambio di messaggi



## Scambio di messaggi

- *Agents*, can cooperate since they share the same communication language and a common vocabulary, which contains words appropriate to common application areas and whose meaning is defined in a shared ontology.
- An agent **communication language** is a messaging protocol, where a message is a textual expression composed of
- A **communication primitive**, called performative, that corresponds to a specific linguistic action (e.g. query, answer, assert, define)
- The **content**, i.e. a sequence of declarative statements built using a knowledge representation language.

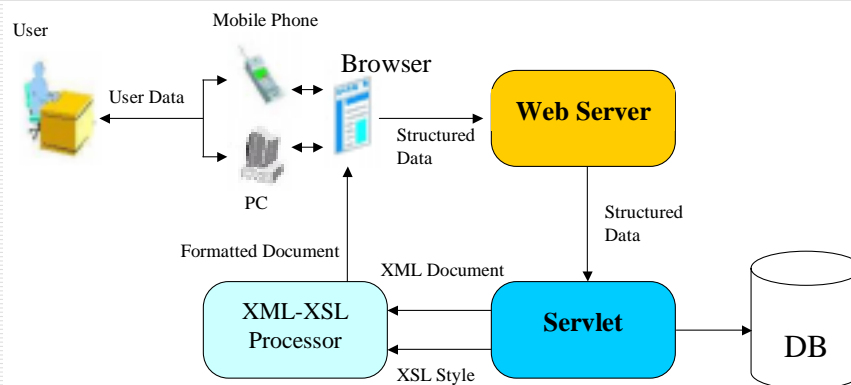
## Scambio di messaggi



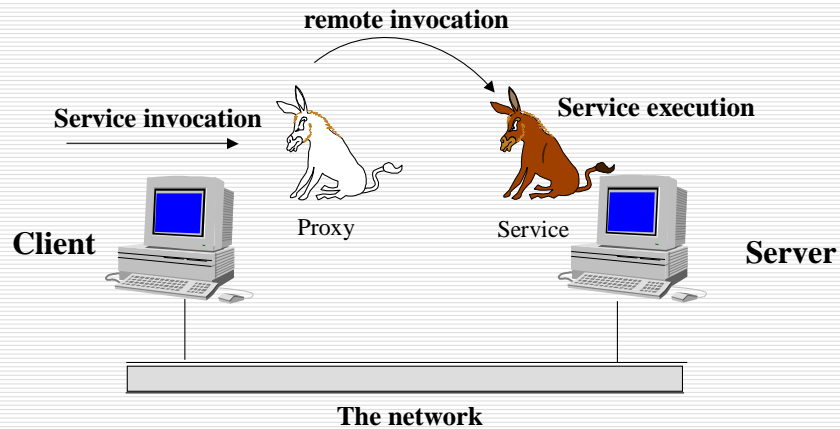
# Paradigma Web

- The Web paradigm is based on:
  - a standard application protocol : **HTTP**
  - A standard client application : the **Browser**
  - A standard data structure language : **HTML**
  - A standard resource locator mechanism : **URL**
  - A standard interface to services : **CGI**

# Paradigma Web



## Invocazione remota

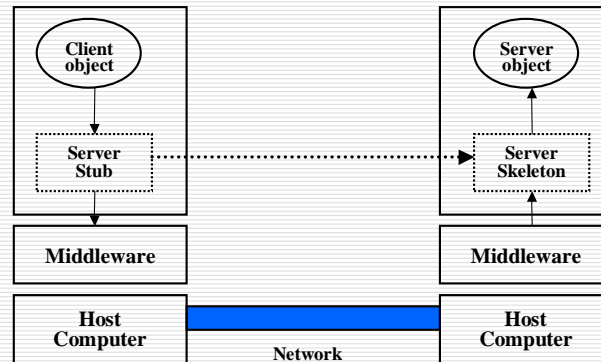


## Stub/Skeleton

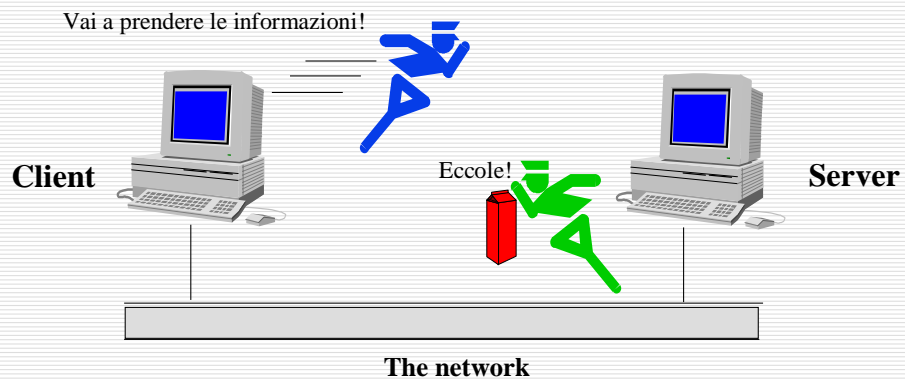
- It consists in implementing a distributed address space where client and server objects can exchange messages. This is accomplished by means of two ancillary objects:
  - The **Stub** is a surrogate (proxy) of the server and resides in the client's address space. It offers the same set of operations (called the *interface*) as the remote server and is in charge of marshalling the client's request and transmitting it through the network.
  - The **Skeleton** resides in the Server's address space and is in charge of receiving and unmarshalling the client's request and of invoking the corresponding server's operation.



## Stub/Skeleton



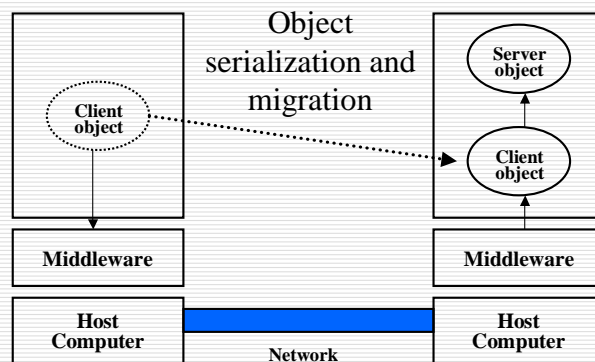
## Oggetti mobili



## Codice Mobile

- ❑ Code mobility can be defined as the capability to dynamically change the bindings between objects and the location where they are executed.
- ❑ An object can migrate at run-time from the Client's to the Server's execution environment or vice-versa.

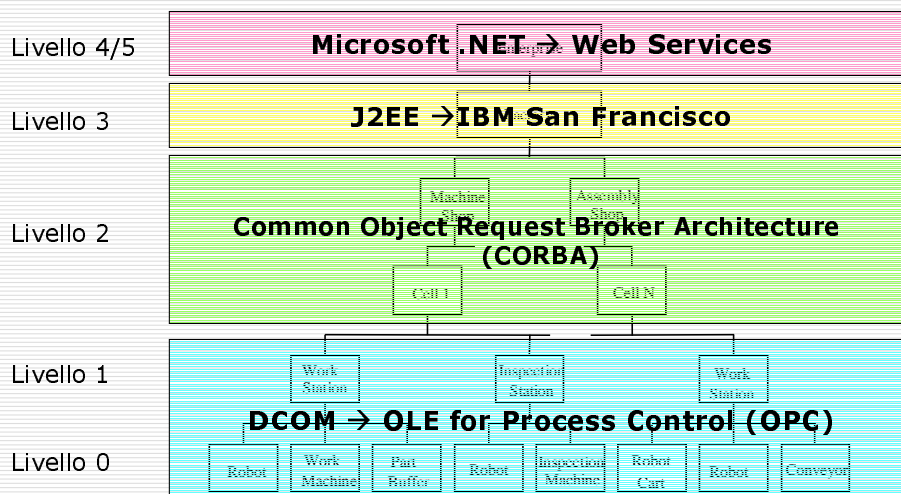
## Codice Mobile



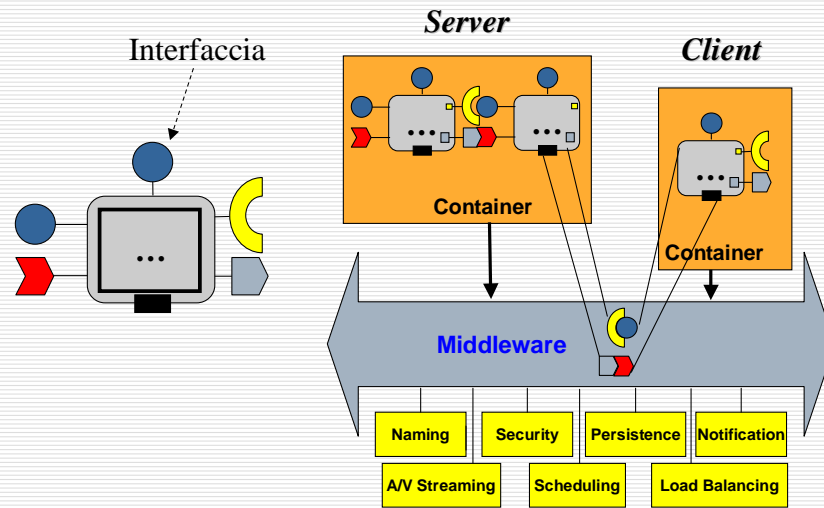
# Codice Mobile

- Depending on the direction of the migration, code mobility is classified in:
  - **Code on Demand.** The Client downloads from the Server's site the software components, called mobile objects, that it needs to execute its tasks. The mobile objects are linked at run-time to the Client's application and are executed in its address space.
  - **Mobile Agents.** The Client sends an active object called mobile agent to the Server's site. The mobile agent uses the Server's resources to perform some useful activities on behalf of the Client.

## Modello di integrazione: middleware



# Middleware



## Middleware frameworks

- ☐ An integrated set of service components that allow distributed systems to operate together.
- ☐ **Distributed event management** supports dynamic notification of events raised by remote objects.
- ☐ **Location-transparent access to remote objects** allows distributed objects to co-operate regardless of their network location, the operating platforms where they are executed, and of the implementation language

# Middleware frameworks

---

- ❑ ***Distributed objects location*** allows client objects to identify at run-time, which server objects offer the functionality they need.
- ❑ ***Network security management*** supports digital signatures and data encryption.
- ❑ ***Persistency and transaction management*** supports persistent storage, access to distributed data, data replication, and data consistency