Sistemi Distribuiti

ANNO: 17/18

RROK GJINAJ - LORIS CANGINI

Definizioni

Scienza
Ricerca e applicazione di conoscenza seguendo una metodologia sistematica basata sull'evidenza.

Fenomeno
qualsiasi occorrenza percepita.

Noumeno
la cosa in sé, così come appare all'osservatore.

Computer Science
la scienza che studia i computer.

Computer
cosa che computa!

Sistema computazionale
sistema di computer.

Computazione
sequenza ben definita di passaggi che porta alla soluzione di un problema.

La computazione è il core della computer science.
La computazione rappresenta il noumeno della Computer Science.

Sia problema che soluzione devono essere codificati in simboli.
Un passaggio è una manipolazione che trasforma un insieme di simboli in un altro.

Processo
serie di passaggi intrapresi per raggiungere un obiettivo.

Macchina
dispositivo con un unico scopo. Tutte le macchine hanno input, output e trasformazioni.

Sistema distribuito
punto di vista dell'utente

Macchina computazionale
macchina il cui scopo è cognitivo e non fisico. Gli input e gli output sono informazioni.

Contesto
macchina, risorse, tempo e spazio.

Computazione temporale
il tempo è rilevante nella computazione.

Computazione spaziale
lo spazio è rilevante nella computazione.

Computazione situata
il tempo o lo spazio sono rilevanti nella computazione.

Sistema computazionale
2 o + processi computazionali computano e interagiscono.

Computazione parallela
più computazioni possono essere eseguite nello stesso momento. Il contesto temporale è lo stesso per tutti i processi.
In sistemi paralleli gli eventi sono totalmente ordinati

Computazione concorrente
Il contesto temporale è diverso tra i vari processi.
In sistemi concorrenti gli eventi sono parzialmente ordinati.

Computazione distribuita
processi computazionali situati su macchine differenti che comunicano tramite messaggi. Il contesto spaziale è diverso tra i vari processi.

Insieme di computer indipendenti che appaiono all'utente come un unico sistema coerente.

punto di vista dell'ingegnere
Collezione di entità computazionali autonome concepite come un unico sistema coerente dal suo designer.

Sistema nel quale i componenti comunicano e si coordinano solo tramite scambio di messaggi.

Situatedness
Proprietà di un sistema di essere immerso nell'ambiente.

Risorsa
Qualsiasi cosa connessa ad un sistema computazionale utilizzabile da qualcuno.

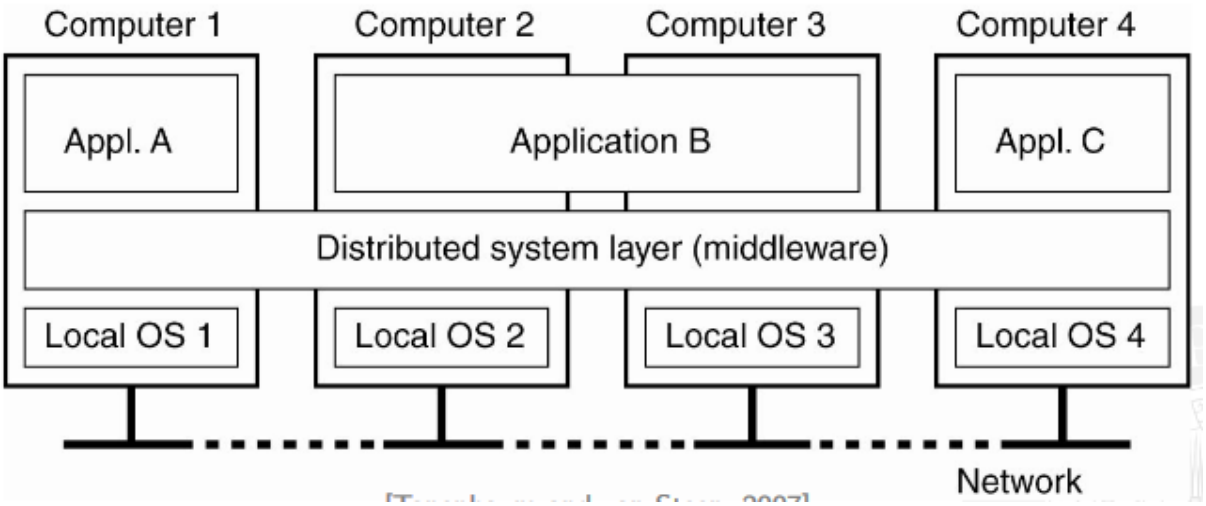
Apertura (Openness)
Abilità del sistema di funzionare con componenti non noti a tempo di design.

Meccanismo
mattone operazionale fornito agli sviluppatori.

Politica
uso di meccanismi per implementare scelte.

Obiettivi e problemi dei sistemi distribuiti

Per raggiungere la coerenza, le entità **autonome** devono **collaborare**.
Per raggiungere l'uniformità, le entità **eterogenee** devono **amalgamarsi**.



Un sistema distribuito è organizzato come un **middleware**, il quale si estende su diverse macchine ed offre a tutti la stessa interfaccia, permettendo l'interazione tra i componenti distribuiti e nascondendo le differenze.

Obiettivi

- Rendere **disponibili** risorse.
- Permettere di **nascondere** risorse quando non necessarie
    - **Access Transparency**: nasconde la differenza nella rappresentazione dei dati
    - **Location Transparency**: nasconde la locazione della risorsa
        - Si accede alle risorse tramite identificatori logici
    - **Migration Transparency**: nasconde il cambio di locazione di una risorsa
    - **Relocation Transparency**: nasconde il movimento di una risorsa
    - **Replication Transparency**: nasconde il fatto che una risorsa è replicata
    - **Concurrency Transparency**: nasconde la condivisione di una risorsa tra più utenti
        - La **consistenza** va comunque mantenuta.
    - **Failure Transparency**: nasconde fallimenti e conseguenti recuperi di risorse
- Promuovere l'**apertura**.
- Promuovere la **scalabilità**.
- Promuovere la **situatedness**.

Problemi

Apertura

Servono regole standard per permettere l'interazione di componenti in un sistema aperto.
Per definire le interfacce tramite le quali si accede ai servizi si usa IDL[1].

L'uso di interfacce non è sufficiente, i componenti devono essere piccoli e specifici, facilmente rimpiazzabili ⇒ i componenti forniscono meccanismi e non politiche.

Interoperabilità

Misura della difficoltà di far funzionare un componente con altri differenti da esso, tramite specifiche standard.

Portabilità

Misura di quanto un'applicazione può essere spostata su diversi sistemi distribuiti continuando a funzionare.

Estensibilità

Misura della difficoltà di aggiungere componenti ad un sistema distribuito esistente.

Scalabilità

Diventa un problema quando una qualsiasi delle dimensioni del sistema distribuito cambia di ordine di grandezza.

Centralizzazione

A volte potrebbe essere necessario centralizzare alcune porzioni del sistema (sicurezza, leggi da rispettare, ...), ma questo inficia sulla scalabilità, quindi va evitato ove possibile.
Anche gli algoritmi centralizzati possono causare problemi, in quanto i dati devono prima viaggiare in rete per raggiungere il punto di centralizzazione, qualsiasi problema nella trasmissione è un danno per l'algoritmo.

[1] Interface Definition Language

Andrebbero usati solamente algoritmi decentralizzati su sistemi distribuiti.

Scalabilità geografica

Su scala geografica la comunicazione è inaffidabile, utenti e componenti non possono essere verificati, ...
Per risolvere questi problemi esistono 3 tecniche:

- nascondere la latenza ⇒ **comunicazione asincrona**
  - Si evita di aspettare le risposte se possibile.
- **distribuzione**
  - Si divide un componente in parti e si spargono nel sistema.
  - Il DNS è un esempio.
- **replicazione**
  - Si replicano componenti nel sistema.
  - Replicare o meno è una decisione del proprietario della risorsa.
  - **caching**: replica fatta dell'utente di sua iniziativa.
  - ✗ Si introducono problemi di **consistenza**.

Dati i requisiti sempre più complessi che si richiedono ai sistemi computazionali, essi devono essere **context-aware**.

Con il mobile è divenuto chiaro che un sistema deve conoscere quando e dove sta funzionando per poter fare il suo lavoro ⇒ l'ambiente è fatto di tempo e spazio (contesto temporale e spaziale).

Tipologie di sistemi distribuiti

Sistemi distribuiti computazionali

Diversi computer distribuiti che fanno compiti ad alta performance.

Cluster

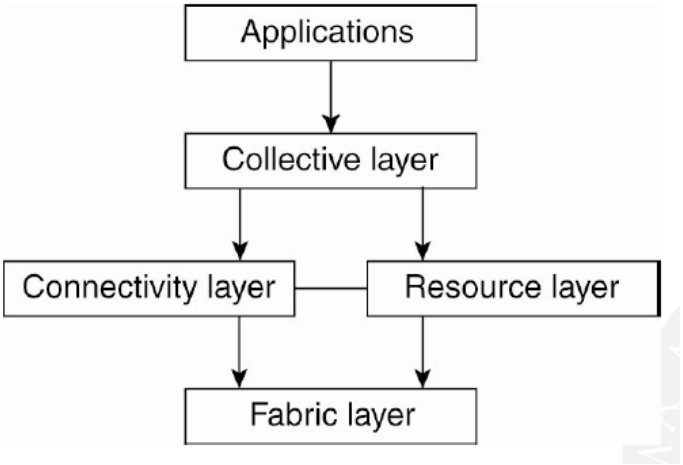
Insieme di **PC simili**, con lo stesso sistema operativo, nello stesso luogo, **connessi tramite LAN**.

- ✓ Più economici rispetto alla costruzione di un unico supercomputer
- ✓ L'aggiunta di potenza di calcolo è più semplice.

Tipicamente usati per computazioni parallele, lo stesso programma viene eseguito su diverse macchine.

Grid

Insieme di risorse provenienti da diverse organizzazioni, quindi molto **eterogenee**.
Hanno un architettura a 5 strati:



Il middleware in questo caso è formato dai 3 strati di mezzo, che forniscono un accesso uniforme a risorse altrimenti sparse.

Sistemi distribuiti informativi

Applicazioni in rete separate da integrare.

Transaction Processing Systems

Quando si hanno database distribuiti vanno fatte transazioni distribuite, mantenendo le proprietà ACID[2].
Se si hanno transazioni innestate, anche esse devono mantenere le proprietà ACID, questo significa che se una sotto-transazione fallisce, allora tutte quelle eseguite fino a quel momento devono essere annullate.
Per risolvere il problema tutte le transazioni vengono fatte su una copia dei dati, ed il risultato viene propagato solo se il tutto va a buon fine.

Enterprise Application Integration

L'integrazione deve avvenire a livello di applicazioni, che devono comunicare con senso tra di loro.
Per permettere ciò si usa un **middleware di comunicazione**. Ne esistono di 4 tipi:

- **RPC**[3]
- **RMI**[4]
- **MOM**[5]
- **Publish & Subscribe**

[2] Atomic, Consistent, Isolated, Durable
[3] Remote Procedure Call
[4] Remote Method Invocation
[5] Message-Oriented Middleware

Sistemi distribuiti pervasivi

Sistemi parte dell'ambiente, senza controllo amministrativo. L'**instabilità** diventa la condizione di base.
I dispositivi che compongono il sistema possono essere configurati da un amministratore, ma di loro natura sono pensati per capire l'ambiente in cui si trovano, adattandosi di conseguenza.
L'ambiente funge da fonte di eventi, l'aggiunta di nuovi dispositivi implica **condivisione**.
Si hanno 3 requisiti per poter costruire sistemi pervasivi:

- Accettare cambi di contesto
- Incoraggiare composizione ad hoc
- Condivisione come default

Home systems

Dato che l'utente è una persona comune, questi sistemi devono essere in grado di autoconfigurarsi ed automantenersi.

Health care systems

I sistema non pervade solamente l'ambiente ma anche le persone, possibilmente senza causare problemi!
La sicurezza dei dati trasmessi è fondamentale.

Reti di sensori

Dati da diversi sensori sparsi nell'ambiente devono essere acquisiti e processati.
Esistono 2 tipologie di sensori:

- **dumb**
- **smart**

Per fare le cose nel modo corretto serve impostare la rete come un albero, le informazioni passano dai livelli inferiori a quelli superiori, che le aggregano, per poi mandare i risultati aggregati dove servono.

Middleware

Uno strato che si frappone tra sistema operativo ed applicazioni.
Le interfacce fra middleware e applicazioni vengono dette **top interfaces**, mentre quelle verso il sistema operativo sono le **bottom interfaces**.
La caratteristica chiave è l'**interoperabilità**, cioè applicazioni che usano lo stesso middleware possono lavorare assieme.
Un middleware è **astratto** quando usa un **modello** comune per tutte le applicazioni

⇒ I vari sistemi vengono adattati al modello. 2 implementazioni dello stesso modello non è detto che siano interoperabili (CORBA 1 ne è un esempio!).

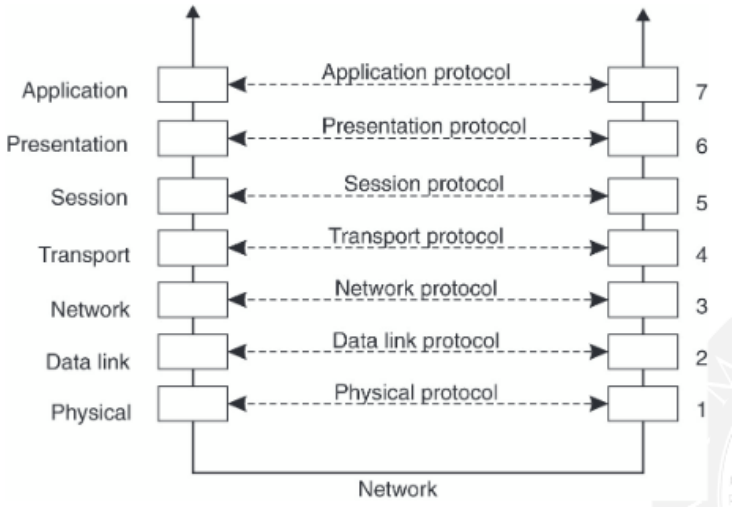
Un middleware è **concreto** quando imposta la stessa **infrastruttura** per tutti

⇒ Gestisce anche i dettagli più fini per garantire interoperabilità (CORBA 2.0).

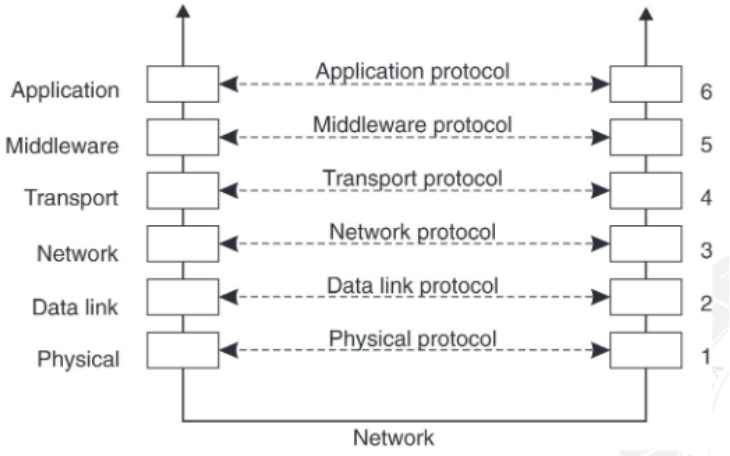
Gli **standard** diventano fondamentali per definire le infrastrutture.

Comunicazione

Nei sistemi distribuiti la comunicazione è un problema del middleware.
È utile usare un modello di riferimento per capire protocolli, comportamenti ed interazioni.
Il più diffuso è l'ISO/OSI:



Nell'inserire il middleware è utile modificare il modello per distinguere i vari strati.



Anche il livello di trasporto potrebbe essere incluso nel middleware.

Tipi di comunicazione

- **Comunicazione persistente**: un messaggio inviato viene memorizzato dal middleware finchè non viene consegnato al destinatario.
- **Comunicazione transiente**: un messaggio viene consegnato solo se mittente e destinatario sono entrambi online.
- ❖ **Comunicazione sincrona**: il mittente si blocca dopo aver inviato il messaggio in attesa di risposte dal middleware o dal destinatario.
- ❖ **Comunicazione asincrona**: il mittente continua ad eseguire dopo aver inviato un messaggio.
- ❑ **Comunicazione discreta**: vengono inviate unità di informazione.
- ❑ **Comunicazione streaming**: viene inviata una sequenza di messaggi ⇒ può essere continua.

Remote Procedure Call

I programmi possono chiamare procedure su altre macchine, mascherandole come chiamate locali.
L'informazione non passa direttamente da mittente a destinatario, si usano gli stub.
Eventuali parametri vengono passati assieme alla richiesta.

Il risultato ritorna al completamento ⇒ di base una RPC è **sincrona**.

Non è scambio di messaggi!

Per evitare problemi dovuti alle differenze tra le macchine servono:

- spazio di referenze comune
- formato dei dati comune
- meccanismi di tolleranza ai guasti

I passaggi che vengono fatti sono:

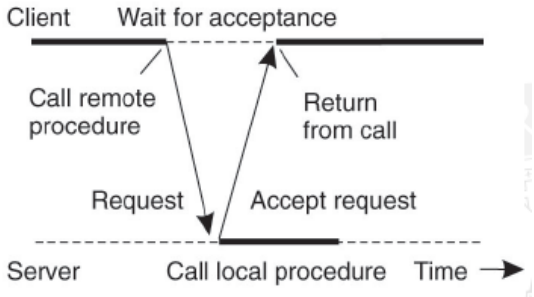
1. la procedura chiama il client stub
2. il client stub costruisce un messaggio e chiama il sistema operativo locale
3. il sistema operativo manda un messaggio al sistema operativo remoto
4. il SO remoto dà il messaggio al server stub
5. il server stub spacchetta i parametri e chiama il server
6. il server fa il lavoro e ritorna i risultati allo stub
7. lo stub impacchetta in un messaggio e chiama il SO
8. il SO del server manda un messaggio al client
9. il SO del client dà il messaggio al client stub
10. lo stub spacchetta e fornisce al client il risultato

Client e Server Stub

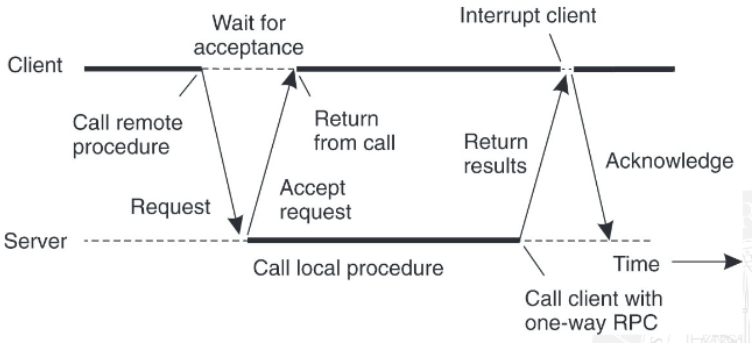
Intermediari che rendono trasparente la comunicazione di rete, preoccupandosi del **mapping** e della **traduzione** tra le rappresentazioni dei dati delle diverse macchine.

RPC asincrona

Il server manda una risposta al client non appena riceve la richiesta, permettendo al client di continuare.



RPC sincrona differita



Si usano 2 RPC asincrone per creare una forma di sincronicità ad-hoc.

Limiti della RPC

Il problema delle RPC è che chiamante e chiamato devono **coesistere** nel tempo, ed a volte questo è un requisito troppo stringente.

Comunicazione orientata ai messaggi

Può essere transiente o persistente.

Nel caso sia persistente si ha un MOM, viene usata quando non si è certi che il destinatario sia attivo. Il mittente mette il messaggio da inviare in una coda, ed inviato ad destinatario, che potrà ritrovarlo dalla sua coda.

Comunicazione orientata agli stream

Uno stream è una sequenza di messaggi tra loro **correlati**. Ci possono essere stream singoli oppure sequenze parallele di stream.
Esistono 3 modalità di trasmissione:

- **Asincrona**: gli oggetti dello stream vengono mandati in sequenza
- **Sincrona**: come asincrona, ma con un ritardo massimo.
- **Isocrona**: come sincrona, ma anche con un ritardo minimo.

Naming

Dare nomi alle entità computazionali in modo da poterle identificare indipendentemente dalla loro locazione.
La **risoluzione** è il fatto di trovare un'entità partendo dal suo nome.

Il **sistema dei nomi** è quello che si occupa delle risoluzioni, definirne uno consiste nel:

- definire un insieme di nomi ammissibili
- definire un insieme di entità con nome
- definire le associazioni tra nomi ed entità

Nome: sequenza di caratteri per riferirsi ad un'entità.
Entità: qualcosa sulla quale si può operare.
Indirizzo: percorso per trovare un entità.
Non è bene usarli come nomi in quanto sono umanamente incomprensibili e dipendono dalla locazione fisica.
Identificatore: nome univoco per un'entità.
Anche gli identificatori non sono tipicamente leggibili dagli umani, per questo serve definire **nomi human-friendly**.

Flat Naming

I nomi sono sequenze di caratteri, illeggibili per gli umani.
Esistono 2 modi per risolvere i nome:

- **Broadcast**: chiedo a tutti
- **Multicast**: chiedo ad alcuni

Structured Naming ⇒ Name Spaces

Si usano nomi composti leggibili dagli umani.
Si parla di Name Spaces quando i nomi sono strutturati gerarchicamente secondo un grafo, come il DNS[6].

Attribute-based Naming (Directory Services)

Si sfruttano insiemi di coppie attributo-valore per identificare le entità.
Questo permette di ricercare entità basandosi su tali attributi.
Esistono diverse possibilità:

- **RDF**[7]: si usano triplette <risorsa, proprietà, valore>

È possibile combinare nomi strutturati e ad attributi, creando directory services distribuiti.

- **LDAP**[8]: standard per accedere e modificare le informazioni di una directory

Middleware orientato agli oggetti

Applicazione di object-oriented a sistemi distribuiti.
Per estendere il paradigma ad oggetti in maniera distribuita il concetto fondamentale diventano le **interfacce remote**, sfruttate tramite **Remote Method Invocation**, che è il corrispettivo ad oggetti della RPC.
Le comunicazioni sono **implicite** ed avvengono tramite stub, che si occupano di trasformare chiamate locali in comunicazioni inter-processo; questo rende le RMI uguali a chiamate di metodo locali (**trasparenza di locazione**).

CORBA[9]

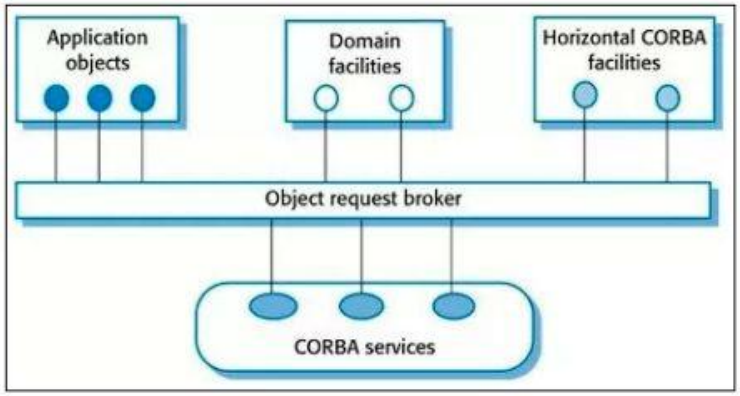
È uno **standard** per il calcolo distribuito creato per permettere la comunicazione fra componenti, indipendentemente dalla loro distribuzione sui diversi nodi della rete o dal linguaggio di programmazione con cui siano stati sviluppati.

[6] Domain Name Space
[7] Resource Description Framework
[8] Lightweight Directory Access Protocol
[9] Common Object Request Broker Architecture

I vari componenti comunicano attraverso un broker (ORB), che può essere visto come un intermediario fra le parti in comunicazione. I componenti sono presentati al broker attraverso un'interfaccia scritta in IDL.

Object Management Architecture (OMA)

L'architettura standardizza le interfacce dei componenti per creare un componente software plug-and-play basato sulla tecnologia OO.



Object Request Broker (ORB)

Protocollo utilizzato per il trasporto a livello applicativo che garantisce l'interoperabilità è il protocollo IIOP.
Fa uso di diverse **interfacce**:

- Lato Client:
  - Client Stub
  - Dynamic Invocation Interface (DII)
- Lato Server:
  - Static Skeleton
  - Dynamic Skeleton Interface (DSI)
  - Object Adapter (OA)

Servizi

Servizi flessibili utilizzati per svariate funzionalità. Ne esistono di diversi tipi:

- **Naming**: funge da paginebianche
- **Object Trader**: funge da paginegialle
- **Life Cycle**: standardizzazione di movimenti, creazioni e cancellazioni di oggetti
- **Event**: consegnatore di notifiche
- **Notification**: miglioramento di Event con più espressività
- **Transaction**: garantisce ACID per le transazioni
- **Concurrency Control**: gestisce i lock

Strutture

Le strutture comuni sono utilizzate per lo sviluppo di applicazioni distribuite in vari settori.

- Le strutture **orizzontali** gestiscono i problemi comuni alla maggior parte dei domini delle applicazioni (GUI, Archiviazione persistente, Documenti composti…).
- Le strutture **verticali** trattano i tratti specifici di un determinato dominio (finanziario, sanitario…).

Interoperabilità

CORBA garantisce interoperabilità condividendo, tramite ORB:

- protocolli di comunicazione ⇒ IIOP[10]
- rappresentazione dei dati ⇒ CDR[11]
- formato delle referenze agli oggetti ⇒ IOR[12]

Architetture Software

Astrazione degli elementi di un sistema in una certa fase delle sue operazioni.

⇒ Modi di organizzare i componenti di un sistema distribuito.

Le varie istanziazioni di un'architettura vengono dette **architetture di sistema**.

Un'architettura è definita da una configurazione degli **elementi architetturali** per ottenere delle **proprietà** desiderate.

Elementi architetturali possono essere:

- **Componenti**: unità con interfacce definite
- **Connettori**: astrazioni di comunicazione
- **Dati**: informazioni trasmesse attraverso connettori tra componenti

La combinazione di componenti e connettori nei vari modi possibili danno luogo a svariate configurazioni, classificate in **stili architetturali**.

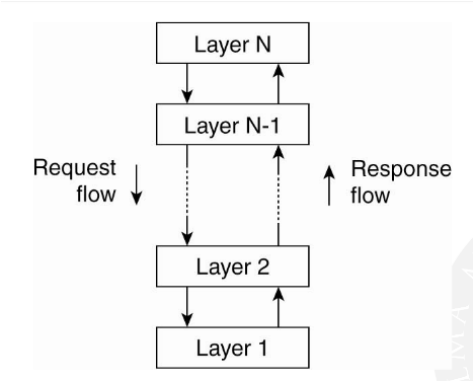
[10] Internet Inter-ORB Protocol
[11] Common Data Representation
[12] Interoperable Object Reference

Stile architetturale

Insieme di vincoli che restringono le caratteristiche degli elementi e le loro relazioni.

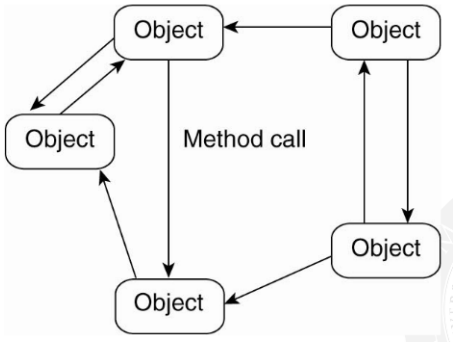
Architettura a strati

I componenti sono organizzati a strati, quelli di uno strato possono chiamare solo i componenti dello strato sottostante.



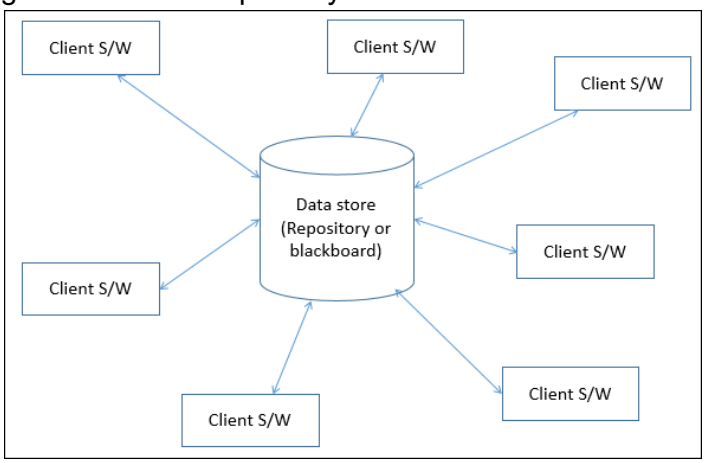
Architettura ad oggetti

I componenti sono oggetti, che comunicano tramite RMI.



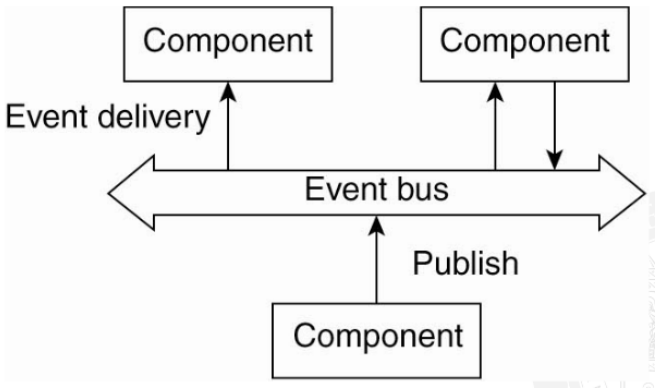
Architettura centrata sui dati

Le comunicazioni avvengono tramite un repository comune.



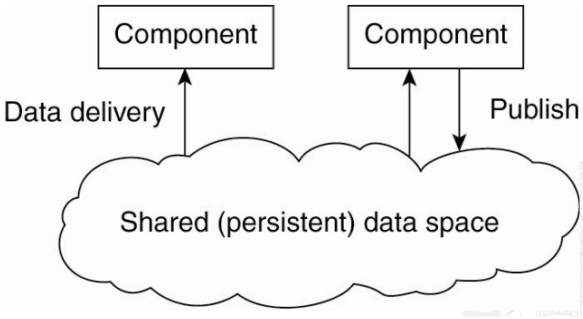
Architettura basata sugli eventi

I processi comunicano attraverso un **event bus**, attraverso il quale gli eventi vengono propagati.



Architettura con spazio dei dati condiviso *(Centrate sui dati + basate sugli eventi)*

Il repository è un spazio dei dati condiviso persistente, con presenza di un event bus dove gli eventi ed i dati vengono memorizzati ed acceduti.



Architetture di sistema

I componenti vengono messi da qualche parte quando un'architettura software viene istanziata.

Architettura centralizzata

I client richiedono servizi a dei server.
La comunicazione può essere:

- **stateless**: ogni interazione avviene a sé, la richiesta contiene tutte le informazioni necessarie (UDP).
    - perfetta per operazioni idempotenti
- **stateful**: le interazioni avvengono in canali preimpostati, la richiesta contiene solamente le informazioni per quella interazione (TCP).
    - meno efficienti ma + sicure

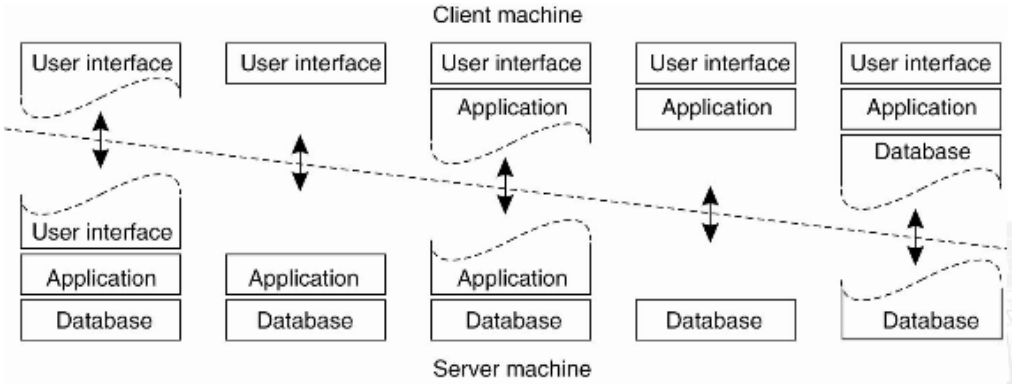
Distribuzione logica

Le applicazioni si compongono di 3 strati:

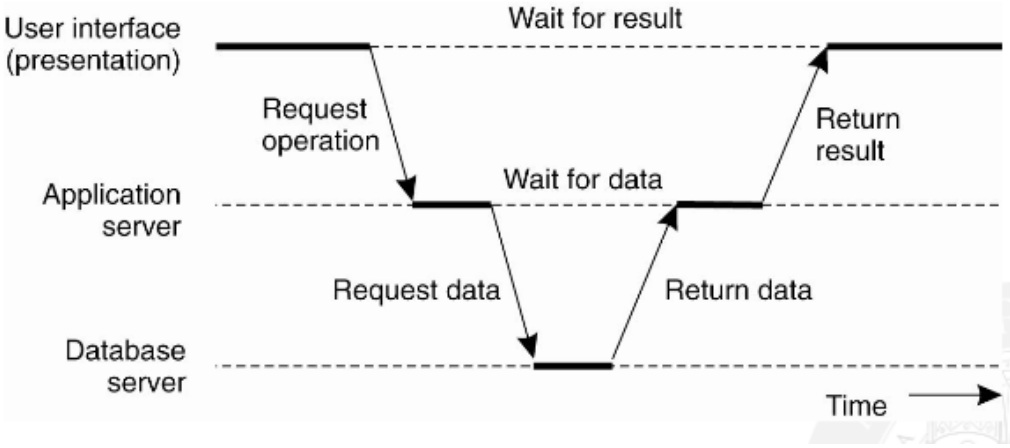
- **interfaccia**: presentazione
- **computazione**: logica di controllo
- **dati**

Distribuzione fisica (Architetture Multi-tiered)

- **Two-tiered**: 2 tipi di macchine, client e server.



- **Three-tiered**: i server possono agire da client rispetto ad altri server.



Distribuzione verticale

Le architetture multi-tiered sono a distribuzione verticale, cioè si mettono strati differenti su diversi computer.
Differenti macchine hanno differenti componenti logici, le funzioni sono logicamente e fisicamente suddivise su più macchine.

Architettura decentralizzata

Distribuzione orizzontale (P2P)

Distribuzione di un singolo strato su diverse macchine.
Client e server fisicamente divisi in parti logicamente equivalenti, tutti i processi sono equivalenti e ognuno lavora per la funzione principale del sistema.

Ogni processo è sia client che server in contemporanea → **Servent**

Architettura ibrida

Combina i benefici delle architetture centralizzate e decentralizzate.

Sistemi distribuiti collaborativi

Per partire si usa uno schema client-server, poi la collaborazione procede in maniera completamente decentralizzata. (BitTorrent, Skype)

Architetture vs Middleware

Un middleware non può adattarsi ad ogni scenario possibile, ognuno è adatto per alcuni specifici scenari.

La soluzione tipica è quella di separare politiche e meccanismi, questo permette di modificare il comportamento del middleware in accordo alle necessità dell'applicazione.
Per farlo possiamo usare il pattern interceptor. L'**intercettore** non è altro che un costrutto software che interrompe il normale flusso di controllo e consente ad altro codice, specifico dell'applicazione, di essere eseguito. L'intercettore permette di implementare la politica specifica per un'applicazione.

Approcci generali al software adattivo

Il problema principale è dato da cambiamenti non prevedibili. Ogni soluzione specifica potrebbe fallire di fronte a modifiche imprevedibili. Gli **interceptor** rappresentano una soluzione generica.

- **separazione degli ambiti**: Separare aspetti funzionali e non funzionali. Approccio non molto funzionante.
- **riflessione computazionale**: L'oggetto ragiona sulle proprie proprietà e assume un comportamento autoadattivo.
- **Progettazione basata sui componenti**: Adattabilità attraverso la composizione. Si può aggiungere un nuovo comportamento aggiungendo al volo un componente. Criterio alla base della creazione di architetture aperte.

Autogestione nei sistemi distribuiti

Lo scopo è quello di arrivare ad una quasi autonomia dei sistemi, nei quali qualunque cambiamento porta a forme di elaborazione che inducono il sistema ad evolvere.
Tale fenomeno è conosciuto anche come **computazione automatica**.
In questi sistemi gli adattamenti hanno luogo tramite uno o più feedback loop, quindi sono sistemi che si evolvono in relazione all'ambiente che li circonda.

Architettura del World Wide Web

Il WWW è un'app network based dove la comunicazione tra componenti avviene con message passing. Le operazioni avvengono in modo non necessariamente trasparente all'utente.

Il WWW è un **sistema ipermediale distribuito**; ipermediale significa che ho informazioni di controllo e presentazione assieme (ad es. un documento non sequenziale, nel quale, cliccando su un link, si può andare in tanti altri punti).
L'ipermedialità distribuita permette di immagazzinare presentazione e controllo delle informazioni in locazioni remote.

I requisiti del WWW sono:

- fornire spazio condiviso di informazioni e dati per farli comunicare e interagire.
- le persone possono immagazzinare e strutturare le proprie informazioni.
- possibilità di referenziare e strutturare le informazioni immagazzinate dagli altri così che non è necessario che tutti abbiano una copia locale.

Semplicità

Tutti possono far parte del sistema, con 3 diversi ruoli.

- **Lettori**: I link ipertestuali rendono la navigazione nelle interfacce grafiche più semplice e generica.
- **Autori**: È possibile pubblicare contenuti senza passare da intermediari.
- **Sviluppatori**: I protocolli testuali sono la base per semplificare lo sviluppo delle applicazioni.

Estensibilità

Permette al sistema di evolversi attraverso le limitazioni di cosa è stato inizialmente sviluppato.

Latenza

Differenza di tempo che trascorre dal trasferimento della risorsa alla sua visualizzazione da parte dell'utenza.
Bisogna cercare di minimizzare le interazioni di rete così da ridurre la latenza.

Scalabilità e Sicurezza

La scalabilità porta un problema di sicurezza perché non conosco tutti gli utenti partecipanti al sistema, i quali potrebbero inviare dati malevoli.
L'autenticazione fa calare la scalabilità e le operazioni di base non dovrebbero richiedere dati che devono essere sicuri. In ambito web, **la condivisione viene prima della sicurezza**. Bisogna autenticare solo ciò che è strettamente necessario.

Deployment indipendente

Nel WWW devono coesistere nuove e vecchie implementazioni e queste ultime non devono impedire alle nuove implementazioni di fare uso delle loro capacità. Gli elementi esistenti devono essere progettati sapendo che vi potranno essere aggiunte feature.

ReST[13]

REST è un tipo di architettura software per i sistemi ipermediali distribuiti.

[13] Representational State Transfer

Garantisce le proprietà quali l'eterogeneità, scalabilità, possibilità di evolvere, affidabilità e resistenza, efficienza e prestazioni.

Il focus è su:

- Ruolo dei componenti
- Vincoli di interazione fra componenti
- Interpretazione dei dati significativi da parte dei componenti.

Il **null style** ha inizio con le necessità del sistema nel suo complesso, senza vincoli, i quali sono poi aggiunti in maniera incrementale.

1° **Client-Server**
Il principio è la separazione degli ambiti.
Server: Offre servizi e rimane in attesa di richieste.
Client: Richiede un servizio e invia una richiesta al server tramite un connettore.
- ✓° Migliora la portabilità dell'interfaccia utente tra più piattaforme.
- ✓° Migliora la scalabilità semplificando i componenti del server.
- ✓° Permette ai componenti di evolversi in maniera indipendente.

2° **Stateless**
Ogni richiesta del client verso il server deve contenere tutte le informazioni necessarie a capire la richiesta, e non può sfruttare in alcun modo alcuna informazione inclusa nel server.
- ✓° Migliora la visibilità permettendo di monitorare il sistema e determinare la completa natura della richiesta.
- ✓° Migliora l'affidabilità facilitando la risoluzione di fallimenti parziali.
- ✓° Migliora la scalabilità perché i componenti liberano le risorse non appena le hanno utilizzate.
- ✓° Migliora la semplicità semplificando l'implementazione perché il server non deve gestire le risorse utilizzate nelle richieste.
- ✗ Interazione senza stato è più pesante in termini di overhead perché ogni volta i messaggi contengono informazioni ripetute.
- ✗ Il server ha meno controllo sul client.

3° **Cache**
Il client dà la possibilità di cachare i dati. Se una risposta è cachabile, allora la cache del client può utilizzare quei dati per richieste equivalenti in seguito.
- ✓° migliora l'efficienza, la scalabilità e la performance percepita dall'utente
- ✓° Riduzione latenza media.
- ✗ Cala l'affidabilità

4° **Interfaccia uniforme**
Devo identificare le risorse in maniera uniforme. Non posso interagire direttamente con le risorse ma tramite una loro rappresentazione.
- ✓° Semplificazione complessiva dell'architettura di sistema
- ✓° Migliora la visibilità delle interazioni
- ✓° Incoraggia l'evoluzione indipendente dei componenti.
- ✗ perdita di efficienza trasferendo informazioni in una forma standardizzata.

5° **sistema a strati**
Architettura gerarchica a livelli, vincolando il comportamento come se ogni componente non potesse vedere oltre al livello con cui interagisce.
- ✓° Migliore semplicità del sistema e promuove l'indipendenza restringendo la conoscenza del sistema al singolo livello.
- ✓° Migliore scalabilità abilitando il load balancing dei servizi agli intermediari
- ✓° Migliora la sicurezza rinforzando le policy.
- ✗ Cala la performance percepita dall'utente aggiungendo latenza e overhead ai dati processati.

6° **Code on demand** (opzionale)
I server possono temporaneamente estendere o personalizzare le funzionalità del client trasferendo del codice eseguibile.
- ✓ Client semplificato riducendo il numero di feature pre-implementate.
- ✓ Migliora l'estensibilità perché le feature sono scaricate dopo la progettazione.
- ✗ Il vincolo è opzionale perché riduce la visibilità.

Elementi architetturali

Il principio fondamentale di ReST è quello di essere un'architettura client-server priva di stato. Una macchina server può ospitare uno o più servizi, ognuno di quali gestiscono una o più risorse.
Ogni informazione a cui può essere dato un nome ed è abbastanza importante da essere referenziata può essere una **risorsa**.

I componenti ReST comunicano inviando la rappresentazione di una risorsa. Che il formato di questa sia differente da quello effettivo della risorsa non è importante, in quanto il formato sarà nascosto dall'interfaccia uniforme del componente.
Una risorsa è il **mapping concettuale** ad un insieme di entità, quindi non è la risorsa fisica in sé, ma è il concetto.

Una risorsa può essere:
- **Statica**: dopo la creazione della risorsa, questa non cambia il suo insieme di origine.
- **Dinamica**: altrimenti.

L'unica cosa che deve necessariamente rimanere statica per una risorsa è la sua semantica, che distingue una risorsa da un'altra.
Ogni risorsa deve avere almeno un **URI**[14], che è il nome e l'indirizzo della risorsa.

Una **rappresentazione** è una sequenza di Byte + metadati che descrivono quei Byte. Una risorsa può avere diverse rappresentazioni, tra loro semanticamente equivalenti.
Una rappresentazione contiene tutte le informazioni utili per descrivere lo stato corrente della risorsa.

Il messaggio di risposta potrebbe includere:

- **Representation metadata**: coppie nome-valore che descrivono struttura e semantica della risorsa.
- **Resource metadata**: tutte le informazioni non specificate nella rappresentazione.

Control Data

Definisce lo scopo del messaggio tra i componenti. Indica a quale parte del dataset il server deve operare il metodo richiesto dal client. Sono anche usati per parametrizzare la richiesta e sostituire il comportamento standard di alcuni elementi in connessione.

Media Types (MIME)

Permettono al client di interpretare correttamente la rappresentazione della risorsa richiesta.

Connettori

Forniscono una generica interfaccia per l'accesso e la manipolazione dell'insieme dei valori di una risorsa. I connettori incapsulano le attività di accesso alle risorse e di trasferimento delle rappresentazioni di una risorsa.
I connettori intervengono in modo stateless, interagendo con la risorsa conoscendo URI ed azione.
Una richiesta può essere mediata da un variabile numero di connettori.
Le tipologie di connettori sono:

- **Client-Server:** Il client invia una richiesta e il server risponde.
- I **risolutori** traducono l'identificatore in un indirizzo di rete necessario per stabilire una connessione tra componenti.
- Un **tunnel** abilita la connessione tra i confini, come un firewall o un gateway di basso livello.
- Il **connettore cache** salva le risposte cachabili di una interazione così che possano essere riutilizzate per interazioni future.

Componenti

- **Origin Server**: ricevitore ultimo delle richieste, offre un'interfaccia generica con struttura gerarchica per l'identificazione delle risorse.
- **User Agent**: Usa un connettore client per iniziare una richiesta e diventa il ricevente ultimo della risposta.
- **Proxy e Gateway** - Agiscono come client e server per poter inoltrare richieste e risposte.

Servizi Web

Applicazioni client/server che comunicano via messaggi su HTTP.

[14] Uniform Resource Identifier

Le funzionalità fornite sono descritte da un contratto esplicito (SOA[15]) o implicito (ROA[16]).
Sono **autonomi**, **lascamente accoppiati**, **componibili**, **riusabili** e **interoperabili**.
I servizi web **incapsulano la logica** all'interno di un contesto.

Per interagire tra di loro, i servizi devono capirsi l'un l'altro, quindi è necessaria una **descrizione**, che specifichi:

- l'indirizzo ed il nome del servizio
- input attesi ed output

Esistono 2 approcci architetturali per progettare servizi web:

- **SOA** (Top-down)
- **ROA** (Bottom-Up)

Service Oriented Architecture (SOA)

Architettura aperta, agile, estendibile, federata e componibile di servizi autonomi, interoperabili, scopribili e riusabili.
Il principio alla base di SOA è il "divide et impera", cioè decomporre le attività di logica applicativa in sotto unità.

Principali feature SOA:

- Un servizio che **incapsula** una unità **logica** all'interno di un certo contesto
- **Loose coupling** e interazioni basate su **messaggi**
- **Autonomia**
- **Componibilità**
- **Riusabilità**
- Supporto ai multivendor e **interoperabilità**

SOA e Web Service non sono sinonimi. SOA è la definizione di un'architettura e il WS è la concreta implementazione del modello architetturale orientato ai servizi.
Un'applicazione web non è necessariamente SOA.

[15] Service Oriented Architecture
[16] Resource Oriented Architecture

Servizi Web Basati su SOA

Un servizio web può essere associato a:

- un **ruolo**: classificazione temporanea dipendente dalle sue responsabilità
    - **fornitore**
      Tramite WSDL viene descritta l'interfaccia, cioè:
        - cosa può essere utilizzato
        - come utilizzarlo, ovvero i vincoli del servizio
        - dove utilizzare il servizio
    - **richiedente**
      Il richiedente legge le descrizioni dei servizi e sceglie quello a lui più appetibile. Invoca il servizio mandando un messaggio.
    - **intermediario**
      Un messaggio può essere processato da più intermediari prima della sua destinazione finale.
        - **passivi**: Si limitano a passare il messaggio.
        - **attivi**: Processano/modificano il messaggio.
- un **modello**: classificazione permanente dipendente dal suo scopo all'interno di una applicazione.

SOAP[17]

Protocollo di trasporto standard per per lo scambio di messaggi tra servizi web.
Usa HTTP come protocollo di trasporto.

Messaggi SOAP contengono:

- **involucro**: il contenitore del messaggio
- **header**: meta-informazioni
- **corpo**: in contenuto

Nodi Soap

I WS sono unità di processo logico dipendenti da infrastrutture fisiche di comunicazione. Ogni piattaforma ha la sua implementazione SOAP.
Un nodo soap è un programma tramite il quale si spediscono/ricevono messaggi SOAP.

WSDL[18]

XML usato per definire le descrizioni dei servizi. Definisce le funzionalità fornite ed il comportamento del servizio

.

Contiene:

- Una **descrizione astratta**: indica il tipo di messaggi scambiati, la definizione delle operazioni e le porte astratte di connessione (interfacce).
- Una **descrizione concreta**: binding (lega le porte astratte al protocollo) + servizi (insieme di porte correlate) + porte (legano le porte astratte agli indirizzi di rete).

Message Exchange Pattern (MEP)

Definizione delle possibili dinamiche di interazione tra servizi.

[17] Simple Object Access Protocol
[18] Web Service Description Language

Simili ai pattern, ma orientati allo scambio di messaggi.

- Request-Response o In-out
- Solicit-Response o Out-in
- One-way o In-only
- Notification o Out-only
- Robust In-only
- Robust Out-only
- In-optional-out
- Out-optional-in

Universal Description Discovery and Integration (UDDI)

Standard che tenta di risolvere i problemi legati alla scoperta e composizione di servizi. I WSDL dei servizi vengono registrati nell'UDDI, che funge da registro interrogabile per trovare le funzionalità offerte dai vari servizi.

Aspetti avanzati

Con la seconda versione dei Web Service sono state aggiunte specifiche per gestire funzionalità avanzate.

WS-Coordination

Framework per gestire attività complesse.
Include 3 servizi:

- **Attivazione**: responsabile della creazione del contesto di coordinazione
- **Registrazione**: tiene traccia dei partecipanti all'attività complessa
- **Coordinatore**: gestisce la coordinazione

WS-Security

Framework che si occupa di tutti gli aspetti riguardanti la sicurezza.
Comprende:

- **WS-Security**: permette alle applicazioni di scambiare messaggi SOAP garantendo la loro integrità, confidenzialità e autenticità.
- **WS-Policy**: definisce le politiche, in termini di requisiti, capacità, ...
- **WS-Trust**: utilizzo di token per potersi fidare dei partecipanti
- **WS-SecureConversation**: sfrutta WS-Security e WS-Trust per creare conversazioni sicure.

WS-BPEL

Linguaggio che permette di definire processi di business e protocolli di interazione.

Web Semantico

Rete nella quale i computer sono capaci di analizzare tutti i dati su Internet, ottenuta estendendo internet fornendo informazioni sul contenuto.

Servizi web RESTful

Basati su ROA. Servizi di questo tipo espongono risorse.
Vedi ReST!

Cloud Computing

Fornisce infrastrutture, piattaforme e applicazioni scalabili, in rete e astratte come servizi on-demand pagati a consumo.

5 caratteristiche fondamentali:

- **on-demand self-service**: i servizi possono essere forniti ai consumatori senza il bisogno di interventi umani.
- **broad network access**: i servizi sono disponibili in rete attraverso meccanismi standard.
- **resource pooling**: le risorse sono suddivise per fornire servizi a molti utenti.
- **elasticity**: richieste di risorse extra sono gestite in automatico.
- **measured quality of service**: si usa un metro di valutazione dei servizi qualitativo e quantitativo.

Service Level Agreement

Accordo tra fornitore e consumatore sul livello di servizio.

Tecnologie

Virtualizzazione

Si astrae dalla natura fisica di una risorsa.

- ✓ Bilanciamento del carico di lavoro
- ✓ Si evita intervento umano
- ✓ Utilizzo contemporaneo di risorse
- ✓ Disponibilità aumentata
- ✓ Delegazione al cliente della gestione in maniera sicura
- ✗ Gestione più complessa

SOA

Componenti come servizi indipendenti che comunicano tramite messaggi.

Servizi web

Accoppiamento lasco, asincrono e basato sui messaggi.

Architettura

Visione organizzazionale

Cloud pubblico: offerto da provider esterni all'organizzazione dei clienti.
Cloud privato: offerto dalla stessa organizzazione dei clienti.
Cloud ibrido: si usa un cloud privato in situazioni normali, gestendo i picchi con un cloud pubblico.

Visione tecnica

Infrastructure as a Service (IaaS)
Platform as a Service (PaaS): ambienti di sviluppo ed esecuzione forniti via Cloud.
Software as a Service (SaaS): software fornito via Cloud.
Humans as a Service (HuaaS)

Replicazione e Consistenza

È utile replicare per incrementare l'**affidabilità,** migliorare le **performance, scalare**, ma ciò causa **problemi di costi e consistenza**.

Per risolvere i problemi di consistenza si ricorre ad un **modello di consistenza**, cioè un contratto tra i processi e l'archivio dei dati che assicura la correttezza dei dati dato un insieme di regole da seguire.

Modelli di consistenza data-centrici

Lo scopo è garantire la consistenza dei dati tra le varie copie.

Consistenza continua

Si pone un limite alle differenza tra le copie.
La differenza viene misurata in **conit**[19], che viene suggerito dal data store.

Consistenza sequenziale

Gli aggiornamenti vengono visti da tutti nello stesso ordine.

Consistenza causale

Indebolimento della consistenza sequenziale. L'ordinamento si limita alle operazioni con relazione di causa/effetto.

Modelli di consistenza client-centrici

Assicurano che ogni qualvolta un client si connette ad una nuova replica, essa è aggiornata rispetto agli accessi dello stesso client ad altre repliche.

Consistenza eventuale

Un processo scrive, gli altri leggono. L'unico conflitto che si può generare è scrittura-lettura, che è normalmente tollerato in quanto, se non ci sono aggiornamenti per un pò, il tutto torna consistente.

Letture monotone

Letture successive di un dato da parte di un client ritornano sempre lo stesso valore o uno più aggiornato.

Scritture monotone

Una scrittura di un dato si completa prima di iniziare una qualsiasi altra operazione da parte dello stesso processo.

Leggi le tue scritture

La scrittura da parte di un processo verrà sempre vista dallo stesso processo con successive letture.

[19] Consistency unit

Scritture seguono letture

Una scrittura da parte di un processo fatta dopo una lettura ha luogo sul dato letto o su uno più aggiornato.

Altre repliche

Replicare non significa necessariamente solo dati. Anche servizi e processi possono essere replicati, per le stesse ragioni per le quali si replicano dati.

Tolleranza ai guasti

Fallimento parziale

Un componente fallisce, ma il sistema continua a funzionare.

Ci si può fidare di un sistema se ha:

- **Disponibilità**: il sistema è pronto per l'uso immediato.
- **Affidabilità**: il sistema può funzionare continuamente senza fallimenti
- **Sicurezza**: quando qualcosa fallisce, non succedono catastrofi
- **Manutenibilità**: la facilità nel riparare guasti

Guasti, Errori, Fallimenti

Guasto: il sistema non fa quello che deve.
Errore: la parte di sistema che ha causato il guasto.
Fallimento: il risultato dell'errore.

Per controllare i guasti è possibile:

- prevenirli
- rimuoverli
- prevederli

I guasti possono essere:

- **transienti**: accadono e scompaiono
- **intermittenti**: continuano ad accadere e scomparire
- **permanenti**: accadono e continuano ad esistere

È possibile mascherare i fallimenti tramite **ridondanza**.

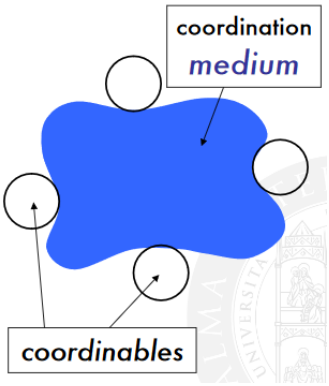
- Ridondanza di informazione: si aggiungono bit
- Ridondanza di tempo: si rifà
- Ridondanza fisica

Coordinazione

Dato un sistema concorrente, gli eventi non sono totalmente ordinati.
Dato un sistema distribuito, le interazioni avvengono in contesti diversi.
L'**interazione** rende il sistema impredicibile, ma tutto ciò non è un problema in quanto, in questo modo, si possono **generare molti comportamenti**, che portano a **più espressività**.

I componenti di un sistema di coordinazione sono:

- **entità**: entità da coordinare, detti anche coordinabili.
- **media**: i mezzi tramite i quali la comunicazione è possibile. *(media è il plurale di medium)*
- **leggi**: regole per descrivere come gli agenti si coordinano.



Modello di coordinazione

I modelli di coordinazione regolano l'interazione tra agenti.
È un framework nel quale l'interazione tra agenti può essere espressa.

Un modello deve permettere la **creazione** e la **distruzione** di agenti, la **comunicazione**, la **distribuzione spaziale e temporale** e la **sincronizzazione**.

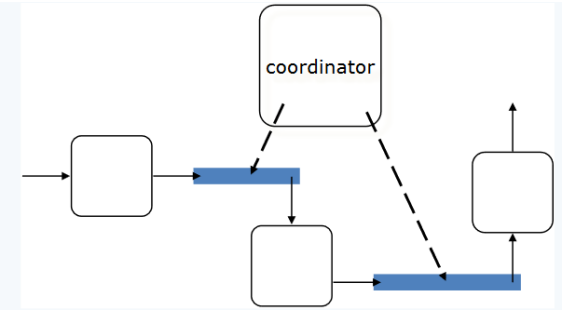
Alcuni meccanismi di comunicazione possono essere:

- Scambio di messaggi
- Agent Communication Languages (ACL)
- Protocolli di comunicazione
- Architetture orientate ai servizi (SOA)
- Web server
- MIddleware

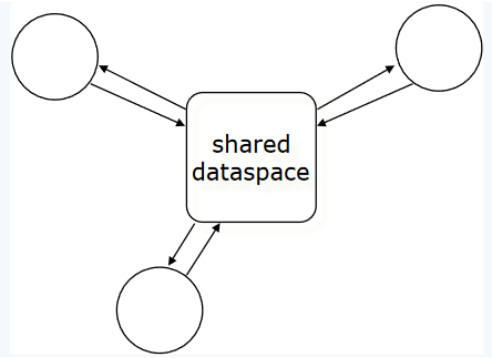
Abilitare l'interazione significa permettere ai componenti di comunicare, senza dare regole su cosa devono dire e quando.
Governare l'interazione regola cosa i componenti devono dire in ogni momento.

Classi di modelli di coordinazione

- **orientati al controllo**: incentrati sull'atto della comunicazione.
  - I processi sono scatole nere
  - I coordinatori creano processi coordinati e canali di comunicazione, determinano la topologia delle comunicazioni.

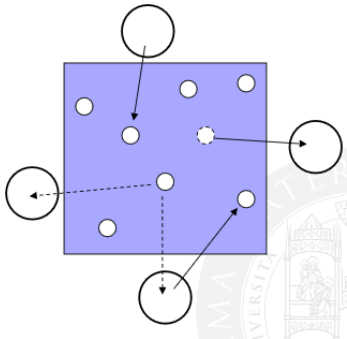


  - ✓ Si ottiene alta flessibilità e controllo.
  - ✓ Si separa coordinazione da computazione
- **orientati ai dati**: incentrati sui dati scambiati nella comunicazione.
  - memoria condivisa
  - canali stateful



Coordinazione basata su tuple

Sono architetture basate sui dati nelle quali i coordinabili interagiscono scambiandosi **tuple**, disponibili in uno **spazio di tuple** condiviso, tramite **primitive**.



Una **tupla** è collezione di informazioni.
I **templates** (o anti-tuple) sono classi di tuple.
I templates vengono cercati nello spazio delle tuple tramite matching.

Linda

Linda è un linguaggio di coordinazione. Fornisce primitive per fare le operazioni di base:

- *out* ⇒ inserire una tupla nello spazio delle tuple
- *in* ⇒ trovare una tupla che rispetti il template dallo spazio delle tuple
    - la tupla trovata viene rimossa dallo spazio delle tuple
    - se più tuple andrebbero bene, una viene scelta non deterministicamente
    - se non viene trovata nessuna tupla, l'operazione viene sospesa in attesa di una tupla conforme.
- *rd*
    - come in, ma la tupla non viene rimossa
- *inp* e *rdp*
    - come i precedenti, ma se non viene trovata un tupla si risponde con fallimento
- *rd_all* e *in_all*
    - agiscono su più tuple

Lo spazio delle tuple può diventare il collo di bottiglia. Per risolvere il problema è possibile definire più spazi di tuple e specificare su quale agire prima di ogni primitiva.

Non è necessario che 2 processi siano nello stesso posto (**disaccoppiamento spaziale**), che eseguano nello stesso momento (**disaccoppiamento temporale**), e nemmeno che conoscano il nome dell'altro (**disaccoppiamento di nome**).

Agenti

Azioni situate

Le azioni vengono fatte in un contesto, il quale le influenza e viene influenzato.

Situatedness

L'**ambiente** è quello che denota il contesto, ed è esplicito nelle computazioni.
I componenti all'interno del sistema sono prevedibili, ma l'ambiente non lo è.

Apertura

L'aver definito cosa sta dentro e cosa è ambiente non rende i sistemi isolati. I confini sono perlopiù convenzionali e permettono interazioni.

Controllo locale

I componenti devono essere **autonomi**, cioè devono avere il loro flusso di controllo.
Il fatto di essere autonomi permette ai componenti di adattarsi alla dinamicità dell'ambiente.

Interazioni locali

I componenti autonomi interagiscono con l'ambiente, ma limitatamente a leggi fisiche o regole logiche.

Programmazione ad agenti

La programmazione ha subito diverse evoluzioni nel tempo:

monolitica ⇒ modulare ⇒ ad oggetti ⇒ **ad agenti**

L'unità di base diventano gli **agenti**.
Sono entità **autonome** con controllo sul loro stato e sono attivi, cioè non vengono invocati, al massimo puoi mandare un messaggio ma non invocare una procedura.

I dati passano tra agenti tramite scambio di messaggi, il flusso di controllo no (ognuno ha il suo e per questo sono autono).
Gli agenti interagiscono tramite messaggi **ACL**[20].

Sistemi multi agente (MAS) sono intrinsecamente **non prevedibili**, **non** si ha un controllo **centralizzato**, il che dà tipicamente vita a fenomeni **emergenti**, che sono il risultato delle interazioni tra gli agenti, ciò che accade è quello che interessa di più .

Per integrare gli agenti con altri paradigmi di programmazione già esistenti si adottano middleware ad agenti.
L'integrazione permette di non far percepire il cambiamento di paradigma come incompatibile con tutto ciò che si è fatto finora.

[20] Agent Communication Languages

Molti campi hanno contribuito alla **definizione degli agenti**:

- **Intelligenza artificiale**
  - agenti come unità che incapsulano **intelligenza**
- **Intelligenza artificiale distribuita**
  - gli agenti sono individui in società, dette **MAS**[21], e sono distribuiti nell'ambiente.
  - ogni agente:
    - ha una **rappresentazione del mondo**
      - necessariamente parziale, **include** il suo **stato attuale** e le **leggi che lo governano**.
      - deve essere **reattivo ai cambiamenti**.
    - è **situato**
    - risolve problemi che richiedono intelligenza
      - ha capacità di inferenza
      - può agire limitatamente sul mondo o su altri agenti, cambiando il loro stato
    - pianifica le sue azioni
      - può avere un **goal**, cioè uno stato del mondo da raggiungere
      - può avere un **task**, cioè un'attività da completare
      - agisce autonomamente per raggiungerli ⇒ è **proattivo**
    - è **flessibile**
    - è **adattabile**
    - **impara**
      - nuova conoscenza
      - nuove leggi del mondo
      - nuove regole di inferenza
- **sistemi paralleli e distribuiti**
- **computazione mobile**
  - un agente non è legato alla macchina nella quale è nato ⇒ è **mobile**
- linguaggi di programmazione e paradigmi
  - un agente **incapsula il proprio flusso di controllo**, non viene mai invocato ⇒ è **autonomo**.
  - non è né un programma né un oggetto.
    - un programma è solo il flusso di controllo
    - un oggetto viene invocato, un agente no
- robotica
  - un agente robotico è intrinsecamente **fisico** e **situato**.

[21] Multi Agent Systems

Un **agente debole** è:

- **autonomo**
  - l'essere intelligente aiuta l'autonomia, ma non è fondamentale.
  - la mobilità può essere espressione di autonomia, ma non è fondamentale.
  - tramite apprendimento un agente migliora, ma non è fondamentale.
- **proattivo**
  - agisce di sua iniziativa per fare operazioni, tipo quando vuole raggiungere un gol, lo decide da solo.
- **reattivo**
  - reagisce a messaggi e/o cambiamenti dell'ambiente dove è situato
- **sociale**: vive in MAS, un'agente in solitudine non ha motivo di esistere.

Un **agente forte** in più ha:

- **credenze**
  - ciò che secondo lui è vero nel mondo circostante.
- **desideri**
  - risultati da ottenere in qualche modo, attuando piani.
- **intenzioni**
  - azioni che sta svolgendo e che deve portare a termine.
- **goal**
- **piani**
  - sequenze di azioni che l'agente può attuare per raggiungere un obiettivo.
- **conoscenza del mondo**
- …

Tipicamente gli agenti autonomi intelligenti vengono concepiti come forti.

Tempo

Tempo fisico

In un sistema distribuito non c'è una definizione naturale di tempo.

Ogni CPU ha un timer (tipicamente un cristallo di quarzo oscillante) con un contatore ed un registro.
Quando il contatore si azzera viene generato un interrupt e il timer si resetta dal registro.
Ogni interrupt è un **clock tick**.

Quando si hanno più CPU i loro clock non si mantengono in sincronia (2 cristalli non oscillano mai allo stesso modo).
Tale fenomeno viene detto **drift** e porta ad una differenza chiamata **skew**.
Servono algoritmi di sincronizzazione:

- **Tempo globale assoluto**: tutti usano un tempo universalmente riconosciuto (UTC[22])
    - Esistono algoritmi per sincronizzarsi, come NTP[23], nel quale un server ha l'orario assoluto e tutti devono adattarsi.
- **Tempo globale relativo**: ci si accorda tra le macchine del sistema sul tempo comune da usare.
    - L'algoritmo Berkeley usa processi in background in ogni macchina che si interrogano a vicenda accordandosi su un tempo comune.

Tempo logico

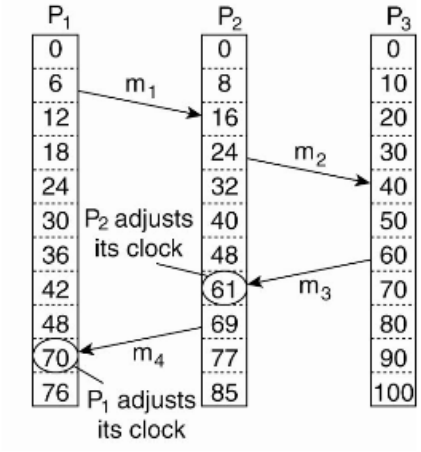
Usare il tempo fisico non è sempre necessario, a volte può essere sufficiente un tempo logico.

Ciò che importa è l'ordine degli eventi. La relazione ***happens-before*** (a → b) indica che prima accade a, poi accade b. Può essere osservata se:

- a e b sono eventi dello stesso processo, ed a accade prima di b
- a è l'invio di un messaggio e b è la sua ricezione

Se non è vero nè a → b nè b → a allora a e b sono concorrenti.

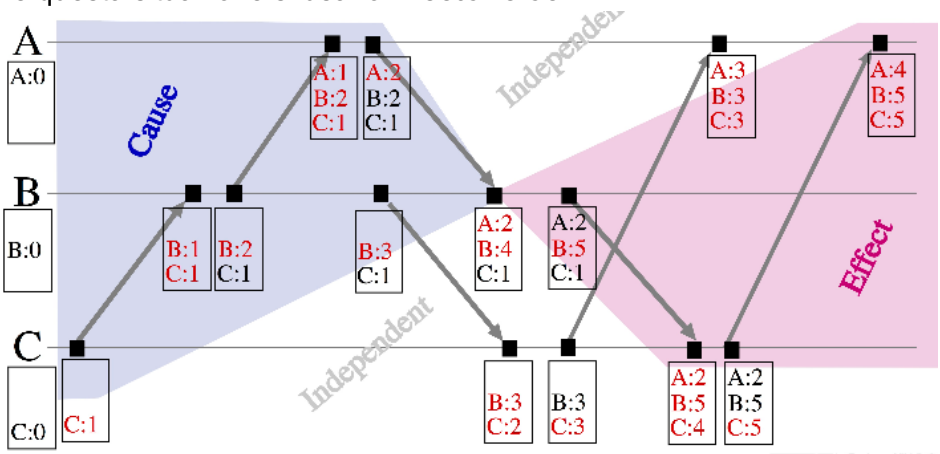
Per accordarsi sul clock logico e garantire la relazioni happens-before si usa l'**algoritmo di Lamport**.



[22] compromesso tra Coordinated Universal Time e Temps Universel Coordonné
[23] Network Time Protocol

a → b implica che C(a) < C(b), ma non si può dire nulla se C(a) < C(b).

Per risolvere questa situazione si usano i **vector clock**.



Con i vector clock, se **tutti** gli elementi di un vettore sono ≤ a quelli di un altro vettore, allora il primo evento è accaduto prima del secondo.

Spazio

Il modo col quale viene rappresentato lo spazio incide sul modo di ragionarci.
Nel tempo sono stati proposti diversi modi per rappresentarlo:

- Geometria Euclidea
- Riemann
- Bolyai e Lobacevskij
- …
- Tarski
- Logiche modali ⇒ logica S4
- Morfologia matematica

Migrazione del codice

A volte è utile spostare il luogo di esecuzione del codice:

- **Bilanciamento del carico**: i processi vengono spostati da macchine molto cariche a macchine con meno carico.
- **Minimizzare le comunicazioni**: se la comunicazione è molto densa non si deve sovraccaricare la rete ma semplicemente spostare i processi in modo che comunichino in locale.
- **Ottimizzare la performance percepita**: si sposta un processo in un'area nella quale la performance percepita dall'utente migliora.
- **Migliorare la scalabilità**: se si riesce a rendere il sistema più grande, si riesce a bilanciare il carico nel miglior modo possibile.
- **Flessibilità** attraverso una configurabilità dinamica: se una macchina risulta poco performante, si può riconfigurare il carico dinamicamente per alleggerire la macchina in questione.

- **Migliorare la tolleranza ai guasti**: avere un'alta flessibilità significa anche migliorare la tolleranza ai guasti.

Un processo è formato da 3 segmenti:
- **code segment**: istruzioni del processo
- **execution segment**: lo stato del processo
- **resource segment**: referenze a risorse esterne

In base a cosa viene trasferito si classificano diversi tipi di code mobility:
- **Weak mobility**: solo il code segment viene trasferito.
  Il codice viene eseguito da capo nel destinatario.
  - Può essere eseguito su un processo separato o meno.
- **Strong mobility**: code + execution segment trasferiti.
  Il processo viene fermato, trasferito e fatto ripartire nel destinatario.
  - Possibile anche clonare invece che migrare il codice.
    La **clonazione** è una copia dell'intero processo sul destinatario, che verrà eseguito in parallelo.

Se la migrazione viene iniziata dal processo dove il codice attualmente risiede viene detta **Sender-initiated**, altrimenti è **Receiver-initiated**.

Per quanto riguarda le risorse, vanno distinti diversi casi:
- **Collegamento processo-risorsa**
  Indica come i processi sono collegati alle risorse:
  - per id: serve una risorsa con un certo nome
  - per valore: serve una risorsa con un certo valore
  - per tipo: serve una risorsa di un certo tipo
- **Collegamento risorsa-macchina**
  Come una risorsa risiede nella macchina:
  - unattached resources: risorse facilmente trasportabili tra macchine.
  - fastened resources: risorse trasportabili, ma con dei costi.
  - fixed resources: risorse legate ad una macchina.

	Unattached	Fastened	Fixed
By identifier	MV (or GR)	GR (or MV)	GR
By value	CP (or MV,GR)	GR (or CP)	GR
By type	RB (or MV,CP)	RB (or GR,CP)	RB (or GR)

MV: sposta la risorsa
GR: stabilisci una referenza globale
CP: copia il valore della risorsa
RB: ricollega il processo a risorse locali

Geometria computazionale

Si usa lo spazio per rappresentare problemi.
Si usa la geometria per renderli computabili.
Si usano algoritmi per trovare soluzioni.

Geographical Information Systems (GIS)

Sistema che supporta lo studio di fenomeni con una specifica locazione nello spazio.

Virtual Reality (VR)

Simulazione di mondi virtuali dove entità tridimensionali interagiscono, con gli utenti in immersione tramite canali sensomotori.

Location-based services (LBS)

Si usa la posizione dell'utente per fornire informazione. (*Pokemon GO*)

Augmented Reality (AR)

Mescolamento di mondo reale con contenuto digitale basato sulla posizione.

Computazione Spaziale

Quando lo spazio non può essere ignorato nella computazione.
Si distinguono:

- **sistemi distribuiti**: lo spazio è un mezzo o una risorsa
- **sistemi situati**: lo spazio è essenziale per la computazione
- **sistemi spaziali**: lo spazio serve per rappresentare il problema

Linguaggi per la computazione spaziale (SCL)

Nati per includere lo spazio nei linguaggi di programmazione.
Un esempio è Proto (di ***Viroli***!).

Per confrontare diversi SCL va definito l'**Abstract Device Model**, composto da:

- **regione di comunicazione**: la copertura spaziale della comunicazione.
- **granularità di comunicazione**: il numero di ricevitori
- **mobilità del codice**: le relazioni tra codici di diversi dispositivi

Per operare sullo spazio servono **3 classi di operatori**:

- **misurare lo spazio**: tradurre lo spazio in informazioni
- **manipolare lo spazio**: tradurre informazioni in spazio
- **computare sullo spazio**: qualsiasi computazione spaziale

Per definire l'espressività si usa il **T-program**, composto da 3 fasi:

I. creare un **sistema di coordinate locali** ⇒ richiede di misurare lo spazio
II. creare una **struttura a T** ⇒ richiede di manipolare lo spazio
III. determinare il **centro di gravità** ⇒ richiede di computare lo spazio

Agenti nello spazio

Alcune proprietà sono essenziali:

- **autonomia**
- **reattività al cambiamento**
- **situatedness**: un agente è immerso nell'ambiente.
- **mobilità**
- **intelligenza**: può essere usata per fare ragionamenti spaziali

Modelli di coordinazione spaziale

Le attività degli agenti possono dipendere l'una dall'altra su base spaziale, questo rende necessaria la **coordinazione spaziale**, ottenuta tramite modelli di coordinazione con concezione dello spazio.

- GeoLinda
- LIME[24]
- TOTA[25]
- SAPERE[26]
- Tuple spaziali
- Spatial ReSpecT

MAS

Anche i MAS devono tenere in considerazione nozioni spaziali.

Jade

I Container mappano la distribuzione fisica degli host.

TuCSoN

Usa centri di tuple spaziali per fornire coordinazione spaziale.

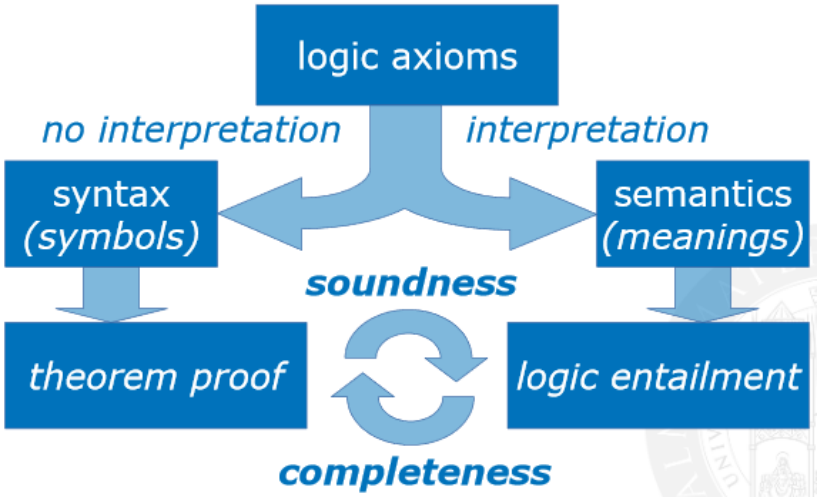
[24] Linda In a Mobile Environment
[25] Tuples On The Air
[26] Self-Aware PERvasive service Ecosystems

Logica

Studio del ragionamento corretto.
Logica computazionale: logica applicata alla computer science.

Implicazione logica: quando una conclusione è vera supponendo che le premesse siano vere, allora le premesse implicano logicamente la conclusione.

La logica può essere discussa in 2 modi:



Computazione: si parte da un'espressione e, tramite regole, si genera il risultato.
Deduzione: si parte da una congettura e, tramite regole, si costruisce la dimostrazione.
Computare è un processo meccanico, mentre dedurre è un processo creativo.

Giudizi: oggetti della conoscenza. (A è vero, ...)

Date premesse (J1,J2,J3,...), conclusioni (J) ed una regola di inferenza(R), una **deduzione** è la prova che partendo dalle premesse, applicando la regola data, si ottengono le conclusioni volute.

$$\frac{J1, J2, J3, \ldots}{J} R$$

Se una delle premesse non è conosciuta, è possibile dedurla a sua volta, rendendo possibile costruire catene di inferenza per provare le conclusioni.

Assiomi: premesse vere senza bisogno di essere dimostrate. Principi indimostrabili.

Concatenamento in avanti: si parte dagli assiomi e si tenta di dimostrare le conclusioni volute.
Concatenamento all'indietro: si parte dalla congettura e si tenta di ricondurla ad assiomi.

Programmazione logica

Lo scopo è quello di costruire un dimostratore automatico di teoremi.
Si sfrutta la logica del primo ordine, tramite il principio di risoluzione di Robinson e l'unificazione.
Definendo un programma come un insieme di clausole Horn e limitando il principio di Robinson, Kowalski[27] ha messo le basi della programmazioni logica.

Termine: la computazione ha luogo su un insieme di termini definiti in un alfabeto.
Una **variabile** è un termine, inizia come non inizializzata; il loro assegnamento viene detto **sostituzione** (X / t ⇒ alla variabile X si dà il valore t).

Un **funtore** con arità 0 è una **costante** ed un termine.
Un funtore di arità n *f(t1,t2,...,tn)*, con n termini al suo interno, è a sua volta un termine.

Backtracking: il controllo è fornito unicamente dal backtracking, cioè la capacità di tornare indietro a fronte di una scelta ed analizzare le altre alternative..

La programmazione logica è di tipo **dichiarativo**, cioè si dice cosa fare, non come farlo.
Come fare le cose è demandato al sistema sottostante.

L'**equazione** è l'operazione di base nella programmazione logica, ha l'effetto di rendere 2 termini uguali.
Una sostituzione che rende due termini uguali è detta **unificatore**.
L'unificatore meno restrittivo viene detto **MGU**[28], rappresenta la soluzione all'equazione logica.

Proposizioni logiche vengono scritte tramite **predicati**.
Gli **atomi** rappresentano la proposizione elementare.
A è un atomo e significa che A è vero.
¬A è una formula e significa che A è falso.
Sia A che ¬A sono **letterali**.

I letterali possono essere combinati tramite **connettori** logici:

- **congiunzione** (A∧B): sia A che B sono veri
- **disgiunzione** (A∨B): almeno uno tra A e B è vero
- **implicazione** (A→B): se A è vero allora B è vero
- **equivalenza** (A↔B): A è vero se e solo se B è vero

Una **clausola** è una disgiunzione finita di letterali.

$$A1,...,An \leftarrow B1,...,Bm$$

Una **Clausal Normal Form** è una congiunzione di clausole.
Una clausola è **definita** se ha un solo letterale positivo.

[27] non credo sia questo Kowalski, ma meritava citazione!
[28] Most general unifier

$$A \leftarrow B1, \ldots, Bm$$

Una clausola è **unitaria** se è definita con 0 letterali negativi.

$$A \leftarrow$$

Un **goal definito** è una clausola senza letterali positivi

$$\leftarrow B1, \ldots, Bm$$

Una **clausola Horn** è una clausola definita o un goal definito.

In un programma:

- clausola definita = **regola**
- clausola unitaria = **fatto**
- goal definito = **goal**

Un programma è una congiunzione di clausole Horn.

Il **principio di risoluzione di Robinson** funziona per qualsiasi clausola, e mostra che è possibile computare se una CNF implica una formula, ma non fornisce la strategia di prova. Kowalski ha dimostrato che tutto ciò è ottenibile restringendo i programmi a contenere solo CNF di clausole Horn; tale principio viene chiamato **risoluzione SLD**[29].

Principio di risoluzione SLD (informalmente!)

Per provare un goal:

1. si cerca, nel programma, una clausola la cui testa unifichi con il goal.
2. si applica il loro MGU alla clausola trovata, specializzandola, cioè assegnando certe variabili.
3. si procede ricorsivamente cercando di dimostrare i sottogoal della clausola nella stessa maniera.

Questo procedimento si conclude con successo quando non si hanno più sottogoal da dimostrare; potrebbe anche fallire o persino non concludersi mai.

⇒ La risoluzione di un programma viene fatta tramite **concatenamento all'indietro**, dai goal agli assiomi.

SLD è non-deterministico in quanto:

- più di una clausola può unificare col goal ⇒ **non-determinismo or**
- più di un goal può essere in attesa di essere provato ⇒ **non-determinismo and**

Programmazione logica distribuita

Il meccanismo di ricerca delle soluzioni può essere valutato in parallelo, in quanto contiene passaggi non-deterministici.

- **parallelismo and**: dato che il corpo di una clausola Horn è una congiunzione di atomi, essi possono essere dimostrati in parallelo se indipendenti tra di loro.
- **parallelismo or**: quando un goal unifica con più di una clausola, esse possono essere provate in parallelo.

[29] Selective Linear Definite

Prolog

Essendo un linguaggio per programmazione logica, Prolog adotta la risoluzione SLD.
Il non-determinismo viene risolto nei seguenti modi:

- i goal sono rimpiazzati da sinistra verso destra
- le clausole vengono valutate dall'alto verso il basso
- i sottogoal vengono considerati immediatamente.

Questa strategia viene detta **depth-first**.

Sfrutta il backtracking automatico salvando dei checkpoint ogni qualvolta ci sarebbe una possibile alternativa alla strada scelta.

Supporta sia parallelismo and che parallelismo or.

Logic Programming as a Service

LPaaS è un'efficace tecnologia abilitante per dispositivi IoT intelligenti.
È un approccio basato sulla logica e orientato ai servizi, concepito e progettato come la naturale evoluzione della programmazione logica nei sistemi pervasivi di oggi.
Lo scopo è di abilitare il ragionamento situato. Diversi **motori Prolog distribuiti** che forniscono programmazione logica in scenari paralleli, concorrenti e distribuiti.

Si fornisce una vista della programmazione logica come servizio, promuovendo **interoperabilità**, **incapsulamento**, **situatedness** e **context-awareness**.

Jade[30]

Framework basato su java per sviluppare applicazioni ad agenti, rispettando le specifiche FIPA[31]. FIPA è un'organizzazione che promuove standard per tecnologie ad agenti e la loro interoperabilità con altre tecnologie.

Jade è un **middleware basato sugli agenti**.
Offre:

- una **piattaforma distribuita ad agenti**
    - distribuita in quanto un sistema Jade può essere diviso tra vari host.
- un sistema a **scambio di messaggi** trasparente e distribuito
- un servizio di **naming** trasparente e distribuito
- paginebianche e gialle per **scoprire** servizi
- **mobilità** degli agenti intra-piattaforma
- tool grafici di **monitoraggio**
- …

Una **piattaforma** ad agenti FIPA può essere distribuita se:

[30] Java Agent Development Framework
[31] Foundation for Intelligent Physical Agents

- ogni host funge da **contenitore** di agenti, cioè fornisce l'ambiente per permettere agli agenti di funzionare.
- c'è almeno 1 **contenitore principale**.
- il container principale mantiene il registro di tutti gli altri container della stessa piattaforma.

Agent Management System

Data una piattaforma JADE, essa ha un unico **Agent Management System (AMS)**, il quale:

- tiene traccia di tutti gli agenti nella stessa piattaforma
- viene contattato da tutti gli agenti prima di fare qualsiasi altra cosa (finché non lo fanno non esistono nemmeno nel sistema!)

➢ è il sistema di **paginebianche**, cioè un servizio di naming trasparente alla locazione.

Directory Facilitator

Ogni piattaforma ha anche un **Directory Facilitator (DF)**, il quale:

- tiene traccia dei servizi forniti dagli agenti nella stessa piattaforma
- viene contattato dagli agenti che vogliono pubblicare servizi.
- è il servizio di **paginegialle**, secondo il paradigma publish/subscribe.

Agent Communication Channel

È presente anche un **Agent Communication Channel (ACC)** ogni piattaforma, che:

- controlla lo scambio di messaggi
- permette comunicazioni **asincrone**, supportando anche quelle sincrone
- si occupa di rendere il tutto conforme a FIPA ACL[32], che riguarda il formato dei messaggi e la loro serializzazione/deserializzazione.

Ogni agente ha una coda dove ACC consegna i messaggi provenienti da altri agenti.
Ogni qualvolta un messaggio viene aggiunto alla coda, l'agente viene notificato.
Se e quando processare i messaggi in coda dipende dall'agente, in quanto autonomo.

Per far si che gli agenti possano capirsi, il formato e la semantica dei messaggi è decisa da ACL. Hanno tutti gli stessi campi con lo stesso significato.

Per interagire gli agenti usano metodi preimpostati:

- **send**
- **receive**: prende il primo messaggio dalla coda (se c'è) in modo asincrono
- **timed receive**: receive sincrona dalla mailbox, con timeout
- **selective receive**: recupero di un messaggio che sia conforme ad un template

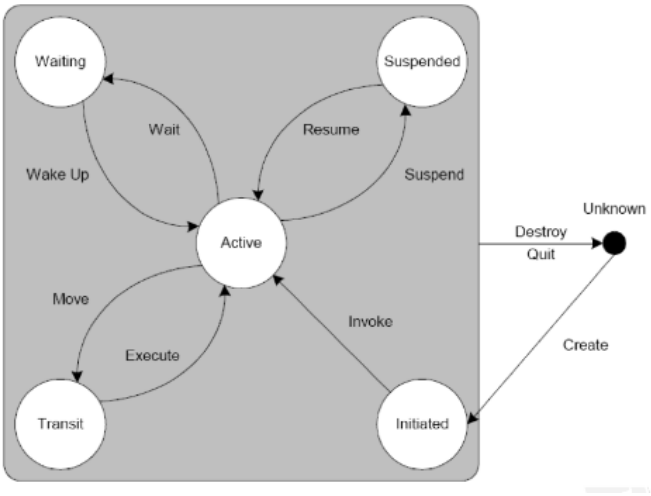
[32] Agent Communication Language

Agenti Jade

In JADE gli agenti sono prima di tutto oggetti Java, con caratteristiche che promuovono la loro **autonomia**.
Ogni agente gira su un unico **thread**, ha **globalmente un nome univoco** e la logica viene espressa in termini di **comportamenti**.

Un'agente può trovarsi in uno dei 5 stati:

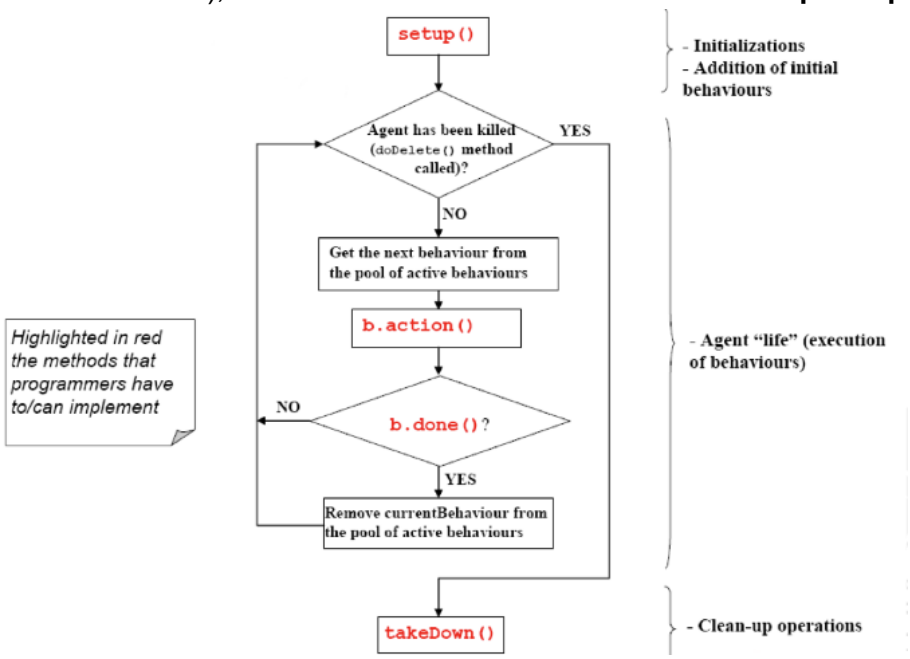
- **inizializzato**: l'agente è stato creato, ma non è ancora registrato nell'AMS.
- **attivo**: l'agente sta eseguendo il suo comportamento.
- **in attesa**: l'agente è bloccato in attesa di qualcosa, tipicamente un messaggio.
- **sospeso**: l'agente è fermo, non sta eseguendo nessun comportamento.
- **in transito**: l'agente ha iniziato a migrare.
- **sconosciuto**: l'agente è morto, è stato deregistrato dall'AMS.



Comportamenti

Un **comportamento** è un'attività da eseguire per completare un task.
Può rappresentare un'attività proattiva o reattiva.

Vengono implementati attraverso oggetti Java e vengono eseguiti concorrentemente (sempre da un unico thread), usando uno **scheduler round-robin**[33] **non-preemptive**[34].



[33] comportamenti ordinati in cerchio, senza priorità.
[34] un comportamento non viene interrotto contro la sua volontà.

Tools di Jade

Il **Remote Monitoring Agent (RMA)** è un agente che permette di controllare il ciclo di vita della piattaforma.
Permette di:

- far partire, fermare ed uccidere agenti
- mandar loro messaggi
- clonare e far migrare agenti
- aggiungere, togliere e spegnere piattaforme.
- …

L'**agente dummy** permette ad umani di interagire con gli altri agenti mandando loro messaggi.
L'**agente sniffer** permette di tracciare e mostrare tutti i messaggi passanti per un agente o un gruppo di agenti.
L**'agente introspettivo** permette di controllare la coda dei messaggi ricevuti e dei comportamenti.

TuCSoN

Modello di coordinazione di processi distribuiti ed agenti autonomi basato su tuple.
Gli **agenti TuCSoN** sono i coordinabili.
Il **centro di tuple ReSpecT** è il media di coordinazione.
I **nodi TuCSoN** sono l'astrazione topologica di base, che contengono i centri di tuple.
Agenti, centri di tuple e nodi hanno tutti identificatori univoci.

I coordinabili necessitano di operazioni di coordinazione per agire sul media di coordinazione.
Queste operazione sono costruite con il **TuCSoN Coordination Language**, definite dall'insieme di **TuCSoN Coordination Primitives** che gli agenti usano per interagire.
I centri di tuple forniscono lo spazio comune per la comunicazione (lo **spazio delle tuple**) e lo spazio programmabile dei comportamenti per la **coordinazione basata su tuple**.

Ogni operazione di coordinazione viene invocata da un agente su un centro di tuple, e si compone di 2 fasi:

- **invocazione**: la richiesta dell'agente
- **completamento**: la risposta del centro di tuple

tname[35] @ netid[36] : port[37] ? op[38]

[35] nome del centro di tuple, se non specificato usa il nome del centro di tuple di default del nodo chiamante
[36] indirizzo del nodo, se non specificato usa l'indirizzo del centro di tuple di default del nodo chiamante
[37] porta, se non specificata usa la 20504
[38] operazione da effettuare

TuCSoN fornisce 9 **primitive di coordinamento** di base:

- **out**: metti una tuple nel centro di tuple
- **rp**, **rdp**: leggi una tupla conforme ad un template da un centro di tuple
- **in**, **inp**: ritira una tupla conforme ad un template da un centro di tuple
- **no**, **nop**: verifica l'assenza di tuple conformi ad un template da un centro di tuple
- **get**: leggi tutte le tuple in un centro di tuple
- **set**: sovrascrivi tutte le tuple di un centro di tuple

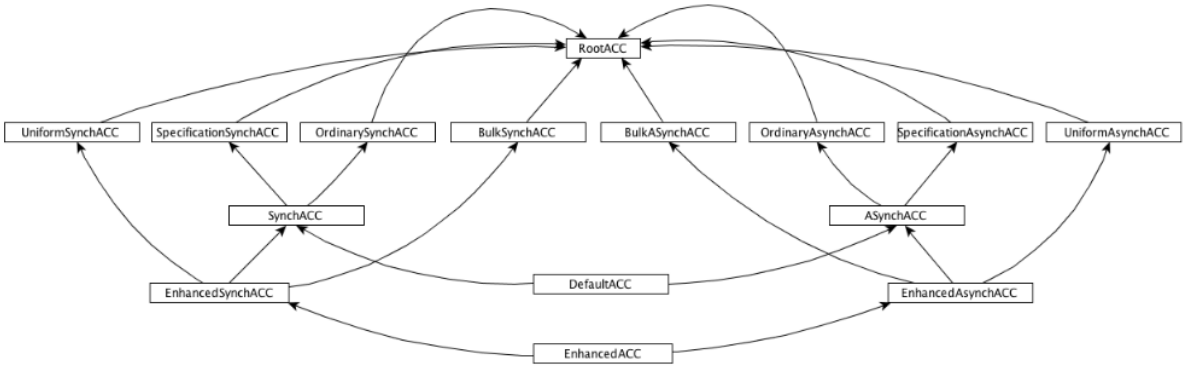
Inoltre fornisce 4 **primitive di massa**:

- **out_all**
- **rd_all**
- **in_all**
- **no_all**

Inoltre, per permettere agli agenti di delegare operazioni complesse legate al coordinamento, TuCSoN fornisce la primitiva **spawn**, che attiva un'attività concorrente da portare avanti in modo asincrono. Uno spawn è **locale** al centro di tuple sul quale viene eseguito.

Un nodo è un dispositivo in rete in ascolto su una porta. Più nodi possono eseguire sullo stesso dispositivo, a patto che usino porte diverse.
Ogno nodo definisce un **centro di tuple di default**, che viene usato ogni qualvolta non viene specificato su quale centro eseguire un'operazione.
TuCSoN è un middleware basato su **Java** e **Prolog**, per questo fornisce API per essere usato da Java e da tuProlog.

Un **Agent Coordination Context** è un'interfaccia fornita ad un agente per eseguire operazioni sui centri di tuple di una specifica **organizzazione**.



ReSpecT

Programmazione degli spazi di tupla.
Esistono due tipi di coordinazione:

- data-driven
- control-driven

Questi due non rispondono però ad ogni esigenza, per questo abbiamo a disposizione un **modello di coordinazione ibrido**.
Si aggiunge un livello basato sul controllo ad uno basato su Linda, lasciando tutte le primitive e meccanismi uguali, aggiungendo due abilità:

- **definire nuovi comportamenti** coordinativi che incorporano le politiche di coordinamento richieste
- **associare comportamenti** coordinativi **a eventi** di coordinamento

Centri di tuple

Un centro di tuple è uno spazio di tuple nel quale è possibile definire **specifiche di comportamento** in risposta ad eventi di comunicazione.
La specifica di comportamento consiste nell'associare qualsiasi evento del centro di tupla a un insieme (anche vuoto) di attività computazionali, chiamate **reazioni**.

Reazioni

Quando un'invocazione raggiunge il centro di tuple, le corrispondenti reazioni vengono invocate.
Ogni reazione può

- accedere e modificare l'attuale stato della tupla
- accedere alle informazioni relative all'evento scatenante
- invocare primitive su altri centri di tuple

Il risultato è la somma degli effetti delle primitive e di tutte le reazioni scaturite, percepito però come una transazione unica arbitrariamente complessa.
Le reazioni vengono associate ad eventi tramite tuple di specificazione, nella forma:

reaction(E,G,R)

La reazione Rθ viene associata all'evento Ev se θ=mgu(Ev,E) e la guardia G è vera.

Tempo

ReSpecT può essere esteso con la concezione del tempo introducendo predicati e guardie per poter ottenere informazioni sul tempo.

- **predicati**: *current_time(time), start_time(time), event_time(time)*
- **guardie**: *before(time), after(time), between(min, max)*

Spazio

Per rendere ReSpecT consapevole dello spazio, sia logico che fisico, si introducono ulteriori predicati e guardie.

- **predicati**: *current_place(place), current_node(node), start_place(place), ...*
- **guardie**: *at(place), near(place, radius), on(node)*

Situatedness

Per fare in modo che ReSpecT possa interagire con l'ambiente, catturando anche eventi che scaturiscono da esso, si introducono altri predicati e guardie.

- **predicati**: *current_env(key, value), start_env(key, value), event_env(key, value)*
- **guardie**: *from_env, to_env*