1. Sistema distribuito

Definiamo come sistema distribuito un insieme di componenti hardware e software comunicanti per mezzo di passaggio di messaggi, posizionati su determinati computer connessi ad una rete.

Oppure possiamo definire un sistema distribuito come una collezione di computer indipendenti che appaiono all'utente come un singolo sistema coeso.

1.1. Caratteristiche principali

- Assenza di memoria condivisa
- Comunicazione per mezzo di messaggi
  - E' quindi richiesto che i componenti del sistema distribuito siano interconnessi in qualche modo
- Assenza di uno stato globale
  - Ogni componente conosce il proprio stato e lo stato istantaneo degli altri componenti
  - Per conoscere lo stato degli altri componenti deve fare una richiesta
- Esecuzione concorrente
  - Tutti i componenti eseguono insieme ed in modo autonomo
  - Le risorse sono condivise e distribuite tra tutti i componenti che eseguono
- Non esiste un controllore/clock/scheduler globale
  - L'unico coordinamento è fatto dai componenti che si scambiano messaggi
- Indipendenza ai fallimento
  - I singoli nodi possono fallire
  - Il fallimento di un singolo nodo non causa il fallimento del sistema distribuito
  - Gli altri componenti non sanno subito se un nodo è fallito
- Trasparenza
  - Il sistema all'esterno appare come unico e indivisibile.

1.1.1. Trasparenza

Costruita in diversi modi:

- Trasparenza di accesso, accedo a risorse remote come se fossero risorse locali
- Trasparenza di locazione, accedo senza sapere dove si trovano fisicamente le risorse
- Trasparenza di concorrenza, più processi eseguono su risorse condivise senza darsi fastidio
- t. di replicazione, permette la creazione di istanze multiple senza che i programmatori lo sanno
- t. al fallimento, se ci sono problemi cerca di mascherarli e non influisce l'esecuzione
- t. alla migrazione: posso trasferire utenti, dati, processi senza che le operazioni ne risentano
- t. alla performance: permette la riconfigurazone dinamica delle performance
- t. alla persistenza: permette di nanscondere se una risorsa sia in memoria o su disco

1.2. Caratteristiche accessorie di un sistema distribuito

- Collagare e standardizzare componenti eterogenei
  - Ad esempio Internet collega reti diverse
  - Reso semplice da un infrastruttura con middleware

- Openness
  - il sistema distribuito lavora secondo uno standard open ben definito
  - semplicità di configurazione e di aggiunta di nuovi componenti
- Mantenere sicurezza
  - Alcuni dati che viaggiano nel sistema distribuito potrebbero essere "privati"
  - E' necessario identificare e verificare l'identità di un altro host connesso al sistema distribuito, per evitare impersonamento
- Offrire scalabilità
  - Un sistema distribuito è scalabile quando funziona all'aumento esponenziale di risorse disponibili e di utenti
  - Ad esempio Internet, come rete IP, ha il limite di scalabilità imposto dalla dimensione finita dell'indirizzo IPv4: prima o poi gli indirizzi finiscono
- Gestione dei fallimenti
  - Il sistema distribuito può essere in grado di riconoscere i fallimenti dei suoi componenti
  - Il s.d. può essere in grado di nascondere all'utente finale i fallimenti
  - Il s.d. può fare recovery dai fallimenti
  - il s.d. e i suoi client possono essere costruiti in grado da sopportare un certo grado di errore
- Gestione della concorrenza
  - il s.d. deve essere in grado di gestire l'accesso simultaneo ad una stessa risorsa

1.3. Problematiche basilari di un sistema distribuito

- Individuare la controparte della comunicazione
  - Serve un modo per identificare l'interlocutore
  - Convenzione di naming: ad esempio tramite DNS, oppure tramite IP statico
- Accedere alla controparte
  - Una volta che l'ho individuata, come mi ci connetto?
  - Accedo direttamente o passo da un access point?
- Comunicare
  - Come invio e ricevo i messaggi? Serve un protocollo di comunicazione
  - Come estraggo informazioni dai messaggi? Il protocollo deve stabilire un formato di comunicazione

1.4. Architettura

Un architettura software definire la strutture del sstema, le interfacce tra i componenti e i protocolli di interazione.

A seconda delle esigenze è possibile utilizzare uno stile di costruzione diverso per ogni sistema distribuito.

- Architettura a strati
  - Detta anche layered
  - I componenti del layer superiore chiamano quelli del layer inferiore, ma non viceversa
  - Più si sale nei layer più l'astrazione è elevata
- Architettura a livelli (tier)
  - Complementare al layering
  - Dato un singolo layer stabilisco su quali server e nodi mettere le funzionalità
  - E' utilizzata per l'organizzazione e la progettazione di applicazioni e servizi
  - E' praticamente come "splitto" logicamente il mio software. Ad esempio se voglio dividere il la logica dal database o la view dalla logica. Allo stesso tempo definisco quali parti voglio fare eseguire al client e quali al server.

Altre tipologie di architettura:

- centrate sui dati
    - i processi comunicano tra di loro tramite una "repository"/file system condiviso
    - ad esempio applicazioni che lavorano basandosi su un file system di rete
    - oppure ancora il Web
- basate sugli oggetti
    - ogni componente del sistema distribuito è visto come un oggetto
    - gli oggetti comunicano tra di loro tramite delle remote procedure calls
- basata sugli eventi
    - i componenti comunicano attraverso lo scaturimento e l'ascolto di eventi
    - ad un evento è associato anche un payload di dati
    - c'è un event-bus che si occupa di propagare gli eventi a tutti i componenti

1.5. DOS, NOS, Middleware

1.5.1. DOS

Distributed operating systems. Si prendono tante macchine omogenee e le si unisce in un unica entità. Le applicazioni usano il DOS senza sapere nulla di quello che sta sotto.

- L'utente non è aware della molteplicità di macchine
- Posso migrare i dati da un punto all'altro in base alle necessità
- Posso migrare la computazione da un punto all'altro
- Posso migrare processi interi tra i vari componenti

In particolare tutto ciò mi permette di realizzare:

- Load balancing
- Velocizzazione computazionale
- Trasparenza HW e SW.
- Esecuzione remota

1.6. NOS

Un sistema operativo senza coesione che collega macchine eterogenee. Le macchine devono solo poter comunicare tra di loro.

- Gli utenti sono coscienti del fatto che ci siano più macchine
- NOS permette di comunicare tra processi tramite socket
- NOS permette l'esecuzione concorrente di processi
- I servizi (come la migrazione dei processi o il load balancing) sono gestite dalle applicazioni
- L'accesso alle macchine del NOS è fatto esplicitamente

1.7. Middleware

Implementa servizi su un NOS per renderlo trasparente alle applicazioni. JavaRMI

Il middleware si mette a metà tra il NOS e le applicazioni.

Offre servizi come:

- naming
- trasparenza di accesso

- persistenza
- transazioni
- sicurezza

Il middleware in realtà è implementato come una applicazione che sta sopra il sistema operativo. Si connette agli altri componenti del sistema distribuito attraverso una comunicazione tramite rete parlando lo stesso protocollo.

1.8. Politiche e meccanismi

Un S.D. dovrebbe essere la composizione di componenti indipendenti, sia logicamente (cioè ogni componente fornisce un servizio autonomo) sia costruttivamente (ogni componente utilizza gi altri per mandare avanti un compito più complesso)

Posso quindi separare meccanismi e politiche (cioè la separazione tra cosa voglio fare e come voglio farlo).

2. Comunicazione

2.1. Tipologie di Comunicazione

- Comunicazione Persistente: il middleware delle comunicazione memorizza il messaggio finchè il destinatario non è pronto a riceverlo
- Comunicazione transiente: se il ricevente non è collegato, il messaggio viene scartato dal middleware

Oppure ancora:

- Asincrona, mando il messaggio e non aspetto che il ricevente accetti la comunicazione
- Sincrona, mando il messaggio, aspetto la risposta, solo poi proseguo l'esecuzione

Combinando otteniamo:

- persistent async: invio il messaggio e vado avanti, il middleware fa lo store del messaggio
- persistent sync: invio il messaggio e vado avanti solo quando il middleware lo ha ricevuto del tutto e mi da l'ok
- transiente async: invio il messaggio solo se B è connesso, intanto vado avanti enlla comunicazione, B lo elbora con calma quando vuole
- transiente sync: invio il messaggio solo se B è connesso, vado avant solo quando ho l'ACK di B

Inoltre ho le seguenti:

- delivery-based transient sync: invio il messaggio, B lo riceve e riprendo l'esecuzione quando B si mette a processarlo
- response-based transient sync: invio il messaggio, B lo riceve, produce una risposta, me la invia, proseguo ad eseguire dopo la risposta.

2.2. Message oriented communication

2.2.1. Message queues

Un modo per realizzare la comunicazione tramite messaggi è attraverso delle message queues: delle code a storage di medio termine che non richiedono ne il trasmettitore ne il ricevitore attivo durante la trasmissione del messaggio.

Le posso usare attraverso 4 metodi:

- put, per fare il submit di un messaggio
- get, per ottenere in modo bloccante il primo messaggio
- poll, controllare in modo non bloccante la coda ogni X secondi
- notify, creare un handler di callback che viene chiamato quando arriva un messaggio

Non c'è garanzia del quando il messaggio viene ricevuto e nemmeno di quello che il ricevente ne farà. Si ha solo la conferma di aver aggiunto il messaggio alla coda.

le queue sono gestite da un queue manager, che si prende carico di dare i messaggi ai riceventi. Possono fare anche da relay: cioè mandare i messaggi da una queue all'altra. Ci potrebbero essere più queue per motivi di dimensione del sistema o per problemi legati al naming.

2.2.2. Message Broker

Necessario per convertire da un formato standard ad un formato capibile dalla applicazione. Si pone tra l'applicazione e le varie code di messaggi. Offre anche altre funzioni di utility all'applicazione che altrimenti dovrebbe implementare nativamente.

Un message broker permette anche di costruire un sistema di publish e subscribe, cioè faccio publish di un messaggio in un determinato topic e questo messaggio viene recapitato a tutti gli host che hanno fatto subscribe per quel topic.

2.3. Stream oriented communication

Questo tipo di comunicazione avviene tramite le socket.

Una socket è una connessione esplicita tra due host, in cui avviene una comunicazione attraverso un flusso di byte. Per effettuare questa comunicazione vengono usate le direttive read e write offerte dal sistema operativo sottostante.

Read e Write sono bloccanti, usano un buffer per garantire flessibilità e safety.

Lato server: socket, bind, listen, accept, read+write, close Lato client: socket, connect, write+read, close

2.3.1. Server

I server possono essere:

- iterativi
    - sodisfano una richiesta alla volta
- concorrenti processo singolo
    - simulano la presenza di un server dedicato
- concorrenti multi-processo
    - creano server dedicati
- concorrenti multi-thread
    - creano thread dedicati

3. RPC

Estensione distribuita alla normale chiamata di una procedura.

3.1. Vantaggi

- Sono semanticamente chiamate di procedura
- Facilmente implementabili

3.2. Svantaggi

- Esplicitamente realizzate dal programmatore
- Statiche: scritte nel codice
- Generalmente bloccanti

3.3. Tipologia di chiamate

3.3.1. Chiamata Sincrona

Al pari di una richiesta HTTP sincrona ad un server, faccio la richiesta, aspetto (bloccandomi) la risposta, riprendo la mia esecuzione.

3.3.2. Chiamata asincrona

Invio la richiestra, aspetto un ACK, riprendo l'esecuzione. Prima o poi arriverà la risposta.

3.4. Architettura

Il client ha metodi stub (circa placeholder) che al posto che eseguire la procedura mandano la richiesta al server, il server ha metodi stub che ricevono la richiesta e che in seguito eseguono la procedura.

3.5. Passaggio di parametri

Alla chiamata del metodo stub i parametri vengono pacchettizzati/serializzati e inviati al server (marshalling). Il server deserializza/spacchetta i parametri ed esegue le procedure.

Il passaggio di parametri è puramente per valore.

4. RMI

RMI = remote method invocation

4.1. Caratteristiche

- Ad oggetti
- Agli oggetti ci si riferisce con indirizzo + porta + id/nome oggetto

4.2. RMI

E' un middleware che:

- fornisce servizi avanzati
  - garbage collection
  - gestione oggetti replicati e persistenti
  - multithreading
- supporta l'invocazione tra oggetti in JVM distinte
  - si passano oggetti Java
  - le classi sono caricate dinamicamente
- si basa sulla portabilità del bytecode
- faccio OOP su un sistema distribuito
- supporta invocazioni statiche e dinamiche

4.3. Implementazione

Il client chiama il metodo di un oggetto interfaccia. La chiamata passa attraverso un proxy che la invia al server dove uno skeleton (l'opposto del proxy) riceve la richiesta. Lo skeleton attraverso un oggetto interfaccia effettua la chiamata dei metodi.

5. Web Services

Un WebService è una applicazione identificata da un URI le cui interfacce e binding possono essere descritte, definite e scoperte tramite documenti XML e con cui è possibile interagire direttamente usando messaggi basati su XML attraverso i protocolli di Internet.

Un WebService è un applicazione self-container, self-describing, modulare che può essere pubblicata, identificata e utilizzata su Web.

5.1. Vantaggi

- più flessibili e stabili dei sistemi classici
- combinabili tra di loro per formare nuovi processi di business
- efficienti e scalabili
- specifica standard
- possibilità di integrarsi con i sistemi legacy

5.2. Caratteristiche principali

- Indipendenti
- Con una interfaccia conosciuta e standard
  - Attraverso un linguaggio descrittivo come WSDL
- Accessibili tramite un URI
  - Scopribili tramite degli index (UDDI)
- Parlano documenti XML

5.3. Entità coinvolte

- Providers
  - Quelli che fanno il web service
- Brokers/Indexer
  - Quelli che lo mettono a disposizione / indicizzano
- Richiedenti
  - Quelli che lo usano

5.4. Operazion che posso fare su un Web Service

- Pubblicarlo
- Trovarlo
- Interagirci

5.5. Composizione di Web Service

I web service possono essere combinati tra di loro per creare una pipeline/workflow che va a soddisfare un processo business. La combinazione ottenuta può essere usata nella creazione di altri WebService.

- Componibili

    - Orchestrazione: descrive come i web services interagiscono tra di loro a livello di messaggio, includendo la business logic e l'ordine di esecuzione delle interazioni. E' rispetto alla prospettiva di un singolo endpoint.
    - Coerografia: descrive la sequenza di messaggi che potrebbero coinvolgere più web services. Definisce lo stato condiviso della interazioni tra le varie entità.

5.5.1. BPEL

BPEL è un linguaggio di definizione basato su XML che aiuta nella definizione di processi di business coinvolgenti vari WebServices.

Struttura

BPEL usa WSDL per specificare le varie attività. Ogni processo BPEL è un WebService esposto usando WSDL.

Un documento BPEL è strutturato in 5 parti:

- message flow: il flow basilare, invocare le operazioni, aspettare il completamento di altre e generazione della risposte
- control flow: definisce stati e transizioni nel flow
- data flow: le variabili che tengono messaggi riguardi lo stato del processo
- fault & exception handling: gestione degli errori
- orchestration: per relazione p2p ?!?!?!?

5.6. WSDL

Descrive Web Service e come accederci.

In particolare:

- elenca e descrive le operazioni pubbliche del WS
- dichiara i tipi di dati per le richieste e le risposta
- informazioni sul protocollo di trasporto
- informazioni di indirizzamento per localizzare il WS

5.6.1. Tipologia operazioni WSDL

- One-way: senza risposta
- request-response: la classica, ricevi subito risposta
- solicit-response: aspetto finchè non hai una risposta
- notification: non aspetto la risposta, pingami quando ce l'hai

5.7. SOAP

Protocollo basato su XML per far parlare WS e applicazioni sul Web.

- SOAP è stateless
- ignora la semantica dei messagi
- incapsula tutte le interazioni tra due endpoint
- lascia il resto ai layer sottostanti

5.7.1. Messaggio soap

- Envelope: che definisce i contenuti del messaggio (a chi è destinato e di che tipo è)
- Header: contiene informazioni su come processare il messaggio e autenticazione
- Body: il contenuto del messaggio

5.8. UDDI

Per risolvere il problema di fare la disovery di altri webservices

Vari approcci possibili:

- approccio registry based
  - autoritativa, controllata da un entità in particolare
  - il proprietario decide quando e cosa pubblicare
  - UDDI
- approccio index
  - Non autoritativo
  - La pubblicazione è passiva
  - ognuno può creare un index
  - Ad esempio Google
- p2p discovery
  - i WS si scoprono a vicenda in modo dinamico

5.8.1. Cosa è UDDI

UDDI è una iniziativa industriale con lo scopo di rendere semplice, veloce e dinamico trovare WS.

UDDI è sia una specifica tecnica che un registro su cui si possono fare query. Con UDDI si parla in SOAPP

6. REST

6.1. REST vs SOAP

Transport protocol

- REST: HTTP
- SOAP: several protocols (e.g., HTTP, TCP, SMTP)

Message format

- REST: several formats (e.g., XML-SOAP, RSS, JSON)
- SOAP: XML-SOAP message format (Envelope, Header, Body)

Service identifiers

- REST: URI
- SOAP: URI and WS-addressing

Service description

- REST: Documentazione Testuale or Web Application Description Language (WADL)
- SOAP: Web Service Description Language (WSDL)

Service composition

- REST: Mashup
- SOAP: BPEL

Service discovery

- REST: no standard support
- SOAP: UDDI

6.2. Principi REST

- Le risorse sono definite da URI
- Le risorse sono manipolate dalla loro rappresentazione
- Ci sono più rappresentazioni per una risorsa
- I messaggi sono stateless
- Lo stato è dettato dalla manipolazione di risorse

6.3. Vantaggi REST

- Posso fare il caching (cisto che è stateless)
- Una stack software molto piu semplice.

Usato per applicazioni CRUD (CREATE READ UPDATE DELETE)

6.4. Design REST

Nome+Verbo+Content Type

wikipedia/pagina + GET + html

7. Domande

Quando un thread, in un programma Java, riceve una notifica attraverso il metodo notifyAll, in quale stato viene spostato?

Stato di Ready

Un vantaggio della virtualizzazione è che

Viene garantita l'integrità del sistema anche a fronte di fallimenti software

In JS è possiible modificare dinamicamente il contenuto e la struttura di una pagina HTML; per selezionare un elemento della pagina:

Si possono usare funzioni di base (come document.getElementById) oppure funzionalità offerte da framework come i selettori jQuery.

Quali vantaggi o svantaggi comporta un'organizzazione a messaggi di lunghezza fissa:

Semplificazione della gestione dei buffer.

Le applicazioni AJAX/JSON sono più efficienti di quelle AJAX/XML?

Si perchè i dati JSON sono in formato compatibile con JS.

Le applicazioni native in Android in che formato sono?

Sono in .apk

Quali sono le informazioni necessarie contenute in una reference Java RMI?

- Indirizzo IP
- Porta
- ID Univoco oggetto

Dare la definizione e descrivere le caratteristiche delle architetture NOS, DOS e Middleware. In particolare illustrando i 3 aspetti principali di cui ogni sistema distribuito deve occuparsi e discuterne come sono trattate nelle architetture citate.

DOS

Distributed operating systems. Si prendono tante macchine omogenee e le si unisce in un unica entità. Le applicazioni usano il DOS senza sapere nulla di quello che sta sotto.

- L'utente non è aware della molteplicità di macchine
- Posso migrare i dati da un punto all'altro in base alle necessità
- Posso migrare la computazione da un punto all'altro
- Posso migrare processi interi tra i vari componenti

In particolare tutto ciò mi permette di realizzare:

- Load balancing
- Velocizzazione computazionale
- Trasparenza HW e SW.

- Esecuzione remota

NOS

Un sistema operativo senza coesione che collega macchine eterogenee. Le macchine devono solo poter comunicare tra di loro.

- Gli utenti sono coscienti del fatto che ci siano più macchine
- NOS permette di comunicare tra processi tramite socket
- NOS permette l'esecuzione concorrente di processi
- I servizi (come la migrazione dei processi o il load balancing) sono gestite dalle applicazioni
- L'accesso alle macchine del NOS è fatto esplicitamente

Middleware

Implementa servizi su un NOS per renderlo trasparente alle applicazioni. JavaRMI

Il middleware si mette a metà tra il NOS e le applicazioni.

Offre servizi come:

- naming
- trasparenza di accesso
- persistenza
- transazioni
- sicurezza

Il middleware in realtà è implementato come una applicazione che sta sopra il sistema operativo. Si connette agli altri componenti del sistema distribuito attraverso una comunicazione tramite rete parlando lo stesso protocollo.

Tipi di passaggi parametri che si possono realizzare in un sistema a oggetti distribuiti

I tipi primitivi e gli oggetti serializzabili per valore.

I riferimenti ad oggetti remoti vengono passati per valore, ma saranno usate per invocazioni remote.

Illustrare graficamente l'architettura 3 tier nel caso del paradigma Ajax. Discutendo dal punto di vista dell'evoluzione: dove si collocano la tecnologie Servlet e JavaScript?

MVC: Model View Controller?

Cosa si intende per RPC asincrono?

E' una chiamata di procedura remota nella quale non si blocca il programma che invia la richiesta per aspettare il risultato ma al massimo, dopo aver inviato la chiamata al metodo, si aspetta un ACK che conferma l'avvenuta chiamata del metodo. Il risultato sarà poi comunicato al chiamante successivamente.

Cosa è e a cosa serve la serializzazione

La serializzazione serve per rappresentare un oggetto come flusso di byte, in modo da poterlo inviare in remoto o salvarlo in locale.

Validare i dati insieriti dall'utente in un form web senza adottare una tecnologia di Scripting lato client

Lato client attraverso i tag e gli attributi HTML5. Lato server attraverso un linguaggi server side.

Le variabili in javascript sono

Non tipizzate.

Per qualsiasi richiesta HTTP (get o post) secondo il modello Servlet quale metodo sarà invocato per primo?

`service()`

Il modello detto hyper conytol dello stile architetturale REST significa che...

Tramite l'utilizzo di tecniche AJAX:

E' possibile rendere una pagina reattiva ad eventi generati dall'utente ma anche eseguire ciclicamente funzioni JavaScript indipendentemente da stimoli da parte dell'utente.

Domande di sistemi distribuiti

Caratteristiche

- mancanza di memoria condivisa: le informazioni vengono scambiate tramite messaggi
- non esiste uno stato globale dell'intero sistema

Ogni componente è autonomo * c'è bisogno di coordinare l'esecuzione concorrente dei diversi componenti del sistema * non esiste un clock globale * impossibilità di un fallimento completo del sistema, ma solo di fallimenti indipendenti dei singoli componenti.

Concetti fondamentali

Che caratteristiche dovrebbe avere un sistema distribuito?

Dovrebbe rendere le risorse facilmente accessibili, nascondere il fatto che queste siano distribuite sulla rete, essere aperto e scalabile.

Dare la definizione e descrivere le caratteristiche delle architetture NOS, DOS e Middleware. In particolare illustrando i 3 aspetti principali di cui ogni sistema distribuito deve occuparsi e discuterne come sono trattate nelle architetture citate

DOS Distributed operating system. Si prendono tante macchine omogenee e le si unisce in un unica entità. Le applicazioni usano il DOS senza sapere nulla di quello che sta sotto.

- L'utente non è aware della molteplicità di macchine
- Posso migrare i dati da un punto all'altro in base alle necessità
- Posso migrare la computazione da un punto all'altro
- Posso migrare processi interi tra i vari componenti

In particolare tutto ciò mi permette di realizzare:

- Load balancing
- Velocizzazione computazionale
- Trasparenza HW e SW.
- Esecuzione remota

NOS Network Operating System. Un sistema operativo senza coesione che collega macchine eterogenee. Le macchine devono solo poter comunicare tra di loro.

- Gli utenti sono coscienti del fatto che ci siano più macchine
- NOS permette di comunicare tra processi tramite socket
- NOS permette l'esecuzione concorrente di processi
- I servizi (come la migrazione dei processi o il load balancing) sono gestite dalle applicazioni
- L'accesso alle macchine del NOS è fatto esplicitamente

Middleware Implementa servizi su un NOS per renderlo trasparente alle applicazioni. JavaRMI

Il middleware si mette a metà tra il NOS e le applicazioni.

Offre servizi come:

- naming
- trasparenza di accesso
- persistenza
- transazioni
- sicurezza

Il middleware in realtà è implementato come una applicazione che sta sopra il sistema operativo. Si connette agli altri componenti del sistema distribuito attraverso una comunicazione tramite rete parlando lo stesso protocollo.

Tipi di trasparenza:

Accesso, ubicazione, migrazione, riposizionamento, replica, concorrenza, guasto.

Il termine "Failure Transparency" sta ad indicare che...

Un sistema è in grado di portare a termine un compito anche in presenza di fallimenti parziali

Il termine Access Transparency sta ad indicare che:

Accesso a risorse locali e remote con le stesse operazioni e con lo stesso formato dei dati.

Il termine Migration o Mobility Transparency sta ad indicare che:

Lo spostamento di dati e/o software non comporta modifiche nei programmi utente

Definizione corretta di middleware

Componente sopra i NOS che offre servizi alle applicazioni sovrastanti

A cosa servono i protocolli?

I protocolli definiscono il formato, l'ordine di invio e di ricezione dei messaggi tra i dispositivi, il tipo dei dati e le azioni da eseguire quando si riceve un messaggio

Problematiche fondamentali dei sistemi distribuiti

1. Individuare la controparte: come faccio a sapere con chi devo parlare? **Naming**
2. Accedere alla controparte: devo poter creare la connessione (**accesso**): come faccio a raggiungere la controparte? Mi serve un riferimento (access point). In Java si chiamano reference per questo!
3. **Comunicare 1**: bisogna comunicare: come faccio a inviare e ricevere messaggi? *Protocollo*
4. **Comunicare 2**: bisogna capire le comunicazioni e renderle comprensibili: come faccio a capire cosa mi ha inviato la controparte? *Formato dei dati (sintassi e semantica)*

Qual è la semantica in un programma Java e nello scambio di messaggio?

TIPIZZAZIONE: so che dati contiene, che operazioni posso fare, …

Stabilisce il formato dei dati e la loro semantica

Come si può "identificare" una risorsa nel Web?

- **Indirizzo IP** dell'host che ospita la risorsa
- **Numero di porta** (port number) – permette all'host ricevente di identificare il processo locale che gestisce la risorsa (e.g., il Web server)
- Cammino (**path**) che identifica la risorsa (e.g., file HTML) sull'host

protocollo://indirizzo_IP[:porta]/cammino/risorsa

Ci dà informazioni su naming, accesso e sulla comunicazione.

Fasi principali modello P2P

- **Boot**: permette a un peer di trovare una rete e connettersi ad essa
- **Lookup**: permette a un peer di trovare con chi interagire
- **Interazione**: due peer si scambiano informazioni

Messaggi e socket

L'invocazione da parte di un server di una accept su una socket determina:

La sospensione del server finché non arriva una richiesta di connessione.

Server multithread con 3 processi client, quante sono le socket aperte?

Più di 3 (precisamente 4 socket aperte)

Interfaccia di base a una coda in uno schema ad accodamento di messaggi

PUT: appende un messaggio alla coda

POLL: controlla se la coda ha qualcosa dentro; se sì, estrae il primo messaggio.

GET: controlla se la coda è piena. Se lo è, legge il primo messaggio. Se è vuota, si blocca e attende che venga inserito un messaggio

NOTIFY: installa un handler da chiamare quando un messaggio viene inserito (PUT) nella coda

Tipi di server

I server possono essere:

- **iterativi**: soddisfano una richiesta alla volta. Viene servito un client alla volta, non scalano
- **concorrenti processo singolo**: simulano la presenza di un server dedicato. Usano la `select()`. Se il canale per le `accept()` è pronto in lettura fa la accept, e per ogni canale pronto in lettura fa R/W
- **concorrenti multi-processo**: creano processi dedicati. System call `fork()`
- **concorrenti multi-thread**: creano thread dedicati. Diversi design pattern:
- Un thread per **client**
- Un thread per **richiesta**
- Un thread per **servente**
- Una **pool** di thread

Socket e canali

Ogni processo comunica attraverso canali

- Un canale gestisce flussi di dati in ingresso e in uscita (dati in formato binario o testuale)
  - Per esempio lo schermo, la tastiera e la rete sono "canali"
  - Dall'esterno ogni canale è identificato da un numero intero detto "porta"
- Le socket sono particolari canali per la comunicazione tra processi che non condividono memoria (per esempio perché risiedono su macchine diverse)
- Per potersi connettere o inviare dati ad un processo A, un processo B deve conoscere la macchina (host) che esegue A e la porta cui A è connesso (well-known port)

La trasmissione di messaggi tramite socket in Java avviene attraverso:

TCP/IP attraverso flussi di byte (byte stream) dopo una connessione esplicita, tramite normali system call read/write (sono sospensive/bloccanti e utilizzano buffer per garantire flessibilità).

La realizzazione di un server basato su socket TCP/IP può essere critica. Quale delle seguenti affermazioni è falsa:

L'invio di stream di byte limita il tipo di messaggio da inviare

L'invocazione da parte di un server di una accept su una socket determina:

La sospensione del server finché non arriva una richiesta di connessione.

Come può un processo "identificare" quello con cui intende comunicare?

[Naming e Accesso coincidono]

- Indirizzo IP
- Numero di porta (1 porta = 1 processo!)

Come avviene la comunicazione TCP/IP?

La comunicazione TCP/IP avviene attraverso flussi di byte (byte stream), dopo una connessione esplicita, tramite normali system call read/write

Come avviene lo scambio di messaggi fra due processi?

Con le primitive `send(receiving process, data)` e `receive(sending process, *buffer)`

Quali vantaggi o svantaggi comporta un'organizzazione a messaggi di lunghezza fissa?

Semplificazione della gestione dei buffer

Un server esegue l'istruzione write(socket, buffer, N) per scrivere sugli n byte del buffer. Che istruzione di read bisogna usare per leggere tutti questi dati?

Le socket trasportano stream di byte. Non esiste il concetto di messaggio e R/W avvengono per un numero variabile e arbitrario di byte.

Bisogna quindi prevedere un ciclo di lettura. Infatti i byte potrebbero arrivare "frammentati" e, dal momento che l'I/O è secondo la macchina di Von Neumann, i byte sono uno stream potenzialmente infinito.

RPC e RMI

RPC: vantaggi e svantaggi

Vantaggi:

- hanno una semantica nota
- sono facili da implementare

Svantaggi:

- sono esplicite (realizzate dal programmatore)
- sono statiche
- sono bloccanti

Primitive	Meaning
Socket	**Create a new communication endpoint**
Bind	**Attach a local address to a socket**
Listen	**Announce willingness to accept connections**
Accept	**Block caller until a connection request arrives**
Connect	**Actively attempt to establish a connection**
Write	**Send some data over the connection**
Read	**Receive some data over the connection**
Close	**Release the connection**

Figure 1: Primitive delle socket: cosa fanno

Comunicazione: differenze transiente/persistente, sincrona/asincrona

Sincrona: il client manda una richiesta al server e rimane in attesa della risposta di questo

Asincrona: il client attende solo una conferma di ricezione del messaggio, poi prosegue con la propria esecuzione. Quando arriveranno i risultati, il processo verrà notificato della ricezione

Transiente: se il client manda un messaggio e il server non è disponibile a ricevere, il messaggio viene scartato

Persistente: il client manda un messaggio e, nel caso il server non possa riceverlo in quel momento, il middleware si occupa di immagazzinarlo e consegnarlo al server quando questo sarà disponibile

Notare che la comunicazione avviene tramite system calls: le invocazioni sono bloccanti e c'è un **context switch** per passare dalla modalità user alla kernel.

Un host manda un messaggio, attraverso la sua rete locale, al communication server. Questo, sfruttando un programma di routing, inoltra il messaggio attraverso la rete Internet. Il messaggio arriva quindi al communication server sulla rete locale dell'host destinatario. Il programma di routing del communication server vede che il messaggio è per un host nella sua rete locale e inoltra il messaggio all'host di destinazione.

A cosa serve uno stub?

A simulare comportamenti locali per chiamate remote.

Il middleware RMI

Fornisce servizi:

- garbage collection di oggetti remoti con un meccanismo di conteggio dei reference esistenti
- caricamento e controllo con un class loader e un security manager
- gestione di oggetti replicati, persistenti
- attivazione automatica degli oggetti
- multi threading

Supporta l'invocazione di metodi tra oggetti in macchine virtuali distinte * le interfacce sono Java (non in un IDL generico) * vengono passati e ritornati oggetti Java * le classi vengono caricate dinamicamente

Si basa sulla portabilità del bytecode e sulla macchina virtuale: più sicuro perché non si deve tradurre nulla

Estende l'approccio OO al distribuito: supporta l'inheritance

Invocazioni: * statiche: interfaccia nota in compilazione * dinamiche: L'invocazione include informazioni logiche sull'identità dell'oggetto e del metodo. `Invoke(obj, method, input_parameters, output_parameters)`

Cosa è e a cosa serve la serializzazione

La serializzazione serve per rappresentare un oggetto come flusso di byte, in modo da poterlo inviare in remoto o salvarlo in locale.

Perché è necessario serializzare gli oggetti per poterli passare come parametro?

Per renderli compatibili al formato in byte di TCP/IP.

Cosa si intende per RPC asincrono?

E' una chiamata di procedura remota nella quale non si blocca il programma che invia la richiesta per aspettare il risultato ma al massimo, dopo aver inviato la chiamata al metodo, si aspetta un ACK che conferma l'avvenuta chiamata del metodo. Il risultato sarà poi comunicato al chiamante successivamente.

Perché non si usano puntatori nei sistemi distribuiti in Java-RMI?

Perché non si possono conoscere le allocazioni di memoria di un altro processo

Quali sono le informazioni *necessarie* contenute in un reference Java RMI?

- Indirizzo **IP**
- Numero di **Porta**
- **ID** univoco dell'oggetto

Dove risiede l'RMI Registry?

Sulla macchina dove risiedono gli oggetti che gestisce.

Tipi di passaggi parametri che si possono realizzare in un sistema a oggetti distribuiti

I tipi primitivi e gli oggetti serializzabili si passano generalmente per valore.

Con RMI è possibile l'invocazione di metodi tra oggetti in macchine virtuali differenti.

In particolare l'RMI per Java, chiamato JavaRMI, utilizza degli stub in maniera simile all'RPC per gestire i parametri a valore, consentendo però anche il passaggio di parametri per reference definendo ulteriori stub specifici per ogni oggetto.

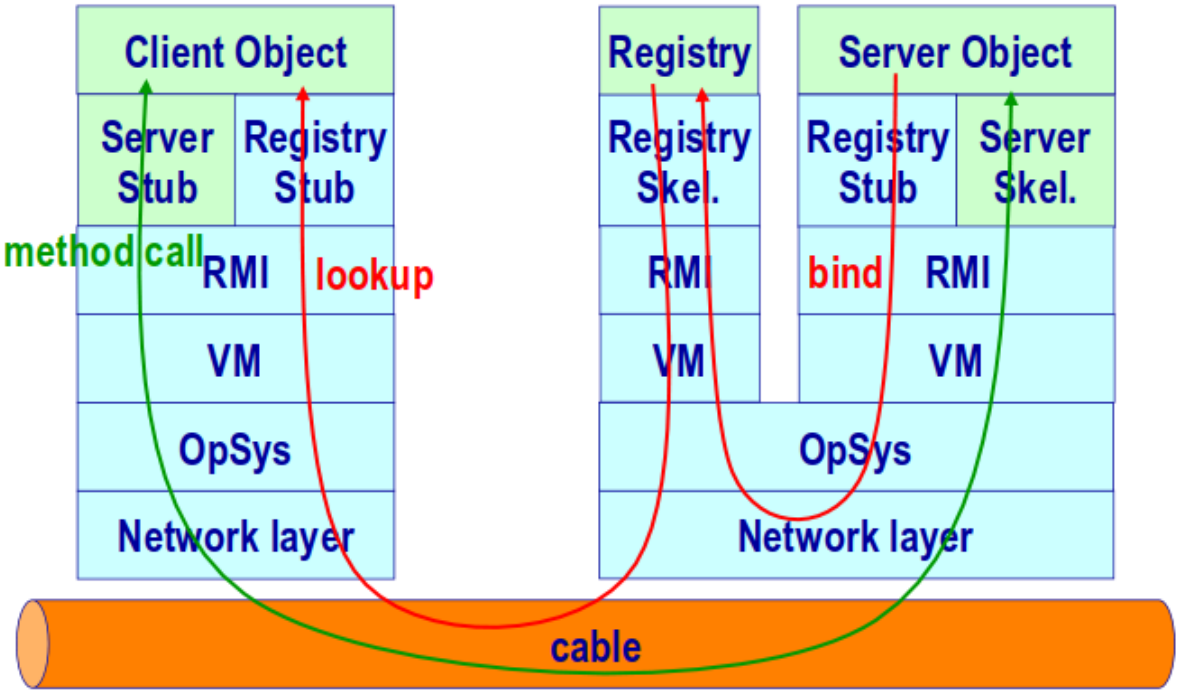


Figure 2: Illustrare graficamente RMI

HTML, DOM, CSS

Cos'è e a cosa serve HTML?

HTML è un linguaggio di markup e serve a dare struttura ai contenuti. NON si occupa di come i contenuti verranno presentati.

Cos'è e a cosa serve DOM?

DOM (Document Object Model) è una interfaccia neutrale rispetto al linguaggio di programmazione e alla piattaforma utilizzata per consentire ai programmi l'accesso e la modifica dinamica di contenuto, struttura e stile di un documento (web).

DOM è una API definita dal W3C (e implementata da ogni browser moderno) per l'accesso e la gestione dinamica di documenti XML e HTML.

Cosa fa un browser?

- Richiede documenti ad un server tramite il protocollo HTTP
- Il motore di rendering manda l'output su qualche dispositivo

Da cosa è costituita una pagina web?

Da vari oggetti (**risorse**) identificati da un URL

Cos'è un web server?

Un Web Server è una applicazione che si occupa di gestire e rendere disponibili le pagine web ai client

Validare i dati insieriti dall'utente in un form web senza adottare una tecnologia di scripting lato client

Lato client attraverso i tag e gli attributi HTML5.

Caratteristiche web 2.0

- Social web
- Innovation in assembly (mash-up)
- Combinazione di vecchie tecnologie e standard per farne di nuovi (come AJAX) che consentono lo sviluppo di RIA (Rich Internet Applications)

Quale affermazione è FALSA? I documenti CSS servono a...

Aggiungere contenuti extra in una pagina Web.

HTTP, Servlet, MVC

Caratteristiche di HTTP

- Protocollo di livello applicativo
- Modello client/server
- Usa TCP (porta 80)
- Stateless (il server non mantiene informazione sulle richieste precedenti del client, quindi ogni richiesta deve contenere tutte le informazioni necessarie per la sua esecuzione)
- Due tipi di messaggi: **request** e **response**

Messaggio HTTP request

1. Request line: METHOD URL VERSION (es. *GET /somedir/page.html HTTP/1.1*)
2. Header lines: HEADER_FIELD_NAME : VALUE (es. Accept: text/html, image/gif,image/jpeg)
3. Extra cr/lf
4. Body

Messaggio HTTP response

1. Status line: PROTOCOL STATUS_CODE STATUS_PHRASE (es. *HTTP/1.1 200 OK*)
2. Header lines: HEADER_FIELD_NAME : VALUEs (es. Content-Length: 6821 o Content-Type: text/html)
3. Extra cr/lf
4. Body

HTTP

- fornisce un modello tipo RPC basato sulle socket
- permette di fare invocqare programmi al server

CGI (Common Gateway Interface)

Permette al server di attivare un programma e di passargli le richieste e i parametri provenienti dal client

È una tecnologia standard usata dai webserver per interfacciarsi con applicazioni esterne generando contenuti web dinamici.

- Una CGI legge una HTTP REQUEST in arrivo al web server ed elabora una HTTP RESPONSE
- molto semplice da scrivere (può essere scritta in ogni linguaggio di programmazione)
- scarsa flessibilità (se viene cambiata la base dati di scambio bisogna riscrivere l'interfaccia)
- lentezza nell'interpretazione dei messaggi HTTP (per il casting del tipo di dato e per l'eventuale controllo di correttezza degli stessi)
- mancanza di uno stato dell'applicazione: ogni form è un messaggio autonomo sotto il punto di vista HTTP ed è quindi necessario l'inserimento di campi nascosti nella pagina o di cookie per il mantenimento delle informazioni di stato delle sessioni

Vantaggi utilizzo linguaggio interpretato:

- Portabilità
- Utilizzo di strutture ben definite
- Serve programmare solo le logiche delle applicazioni

Servlet

- piccole applicazioni Java residenti sul server
- servlet = componente gestito in modo automatico da un container o engine
- ha un'interfaccia che definisce il set di metodi (ri)definibili
- container controlla le servlet (attiva/disattiva) in base alle richieste dei client
- residenti in memoria:
  - hanno stato
  - consentono interazione con altra servlet
- cookies e HTTPSession consentono di essere stateful
- metodi `init()` `service()` `destroy()`
- javax.servlet.GenericServlet e javax.servlet.http.HTTPServlet
- HTTPServlet implementa `doGet` `doPost` ecc in `service`
- `HTTPServletRequest` riceve da `service` la richiesta del client
- `HTTPServletResponse` riceve da `service` la risposta per il client

Ciclo di vita servlet

- Una servlet viene creata dal container quando viene effettuata la prima chiamata: è condivisa da tutti i client e ogni richiesta genera un thread che esegue la doXXX appropriata
- Viene invocato il metodo `init()` per inizializzazioni specifiche
- Una servlet viene distrutta quando non ci sono servizi in esecuzione o quando scade un timeout predefinito
- `destroy` invocato per terminare correttamente la servlet. N.B.: potrebbe essere ancora in esecuzione `service`, quindi `destroy` deve notificare lo shutdown e attendere completamento `service`

Per qualsiasi richiesta HTTP (GET o POST) secondo il modello Servlet quale metodo sarà invocato per primo?

`service()`

Perché il ciclo di vita delle HTTPServlet può essere gestito da un engine?

Perché hanno un'interfaccia nota

JSP Java Server Pages

- specificano l'interazione tra un contenitore/server ed un insieme di "pagine" che presentano informazioni all'utente
- pagine costituite da tag tradizionali (HTML, XML, WML, …) e da tag applicativi che controllano la generazione del contenuto
- rispetto ai servlet, facilitano la separazione tra logica applicativa e presentazione
- il codice JSP va incluso in tag speciali, delimitati da `<%` e `%>`
- la pagina viene convertita automaticamente in una servlet java la prima volta che viene richiesta
- in entrambe le modalità, il file JSP viene prima compilato
- la versione compilata viene tenuta in memoria per rendere più veloce una successiva richiesta della pagina
- commenti: `<%-- ..... -->`
- direttive: `<%@ ... %>`:
  - `page`: liste di attributi/valore
  - `include`: include pagine HTML/JSP in compilazione
  - `taglib`: tag definiti dall'utente implementando opportune classi
- direttive JSP `<jsp:direttiva ... >`
  - `<jsp:forward ... >`: determina invio richiesta concorrente
  - `<jsp:include ... >`: invia dinamicamente la richiesta ad una data URL e ne include il risultato
  - `<jsp:useBean ... >`: localizza ed istanzia (se necessario) un javaBean nel contesto specificato
- elementi di scripting:
  - `<%! declaration; %>`
  - `<%= expression %>`
  - <% codice (*scriptlet*) %>: le variabili valgono per la singola esecuzione e ciò che viene scritto sullo stream di output sostituisce lo scriptlet
- gli oggetti possono essere creati:
  - implicitamente usando le direttive JSP
  - esplicitamente con le azioni
  - direttamente usando uno script (raro)
- gli oggetti hanno un attributo che ne definisce lo scope: dal più visibile al meno visibile
  - application
  - session
  - request
  - page

Esecuzione di JSP

- il client richiede via HTTP un file .jsp
- il file .jsp viene interpretato e accede a componenti lato server (Java Beans e Servlet) che generano contenuti dinamici
- il risultato viene spedito al client sotto forma di pagina html

Il pattern MVC: Modern View Controller

1. La richiesta viene inviata ad una Java Servlet che:
   - genera i dati dinamici richiesti dall'utente
   - li mette a disposizione della JSP come Java Beans
2. La servlet richiama una pagina .jsp che:
   - legge i dati dai Beans
   - organizza la presentazione in HTML che invia all'utente

In generale:

- l'utente manda una richiesta al controller;
- il controller si affida al model per soddisfare la richiesta (es. eseguire una query su un DB);
- il controller riceve dal model i risultati e li inoltra alla view, che li dispone come output

Nel caso specifico di servlet/JSP/JavaBeans, la servlet inoltra alla JSP il JavaBeans, ma non sempre è così.

JavaBeans

Un bean è una classe che segue regole precise:

- deve avere costruttori senza parametri
- attributi private
- gli unici metodi sono getters e setters

		cache	safe	idempotent
OPTIONS	represents a request for information about the communication options available on the request/response chain identified by the Request-URI			✓
GET	means retrieve whatever information (in the form of an entity) is identified by the Request-URI	✓	✓	
HEAD	identical to GET except that the server MUST NOT return a message-body in the response	✓	✓	
POST	is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line			
PUT	requests that the enclosed entity be stored under the supplied Request-URI			✓
DELETE	requests that the origin server delete the resource identified by the Request-URI			✓
TRACE	is used to invoke a remote, application-layer loop- back of the request message			✓

Figure 3: Metodi HTTP

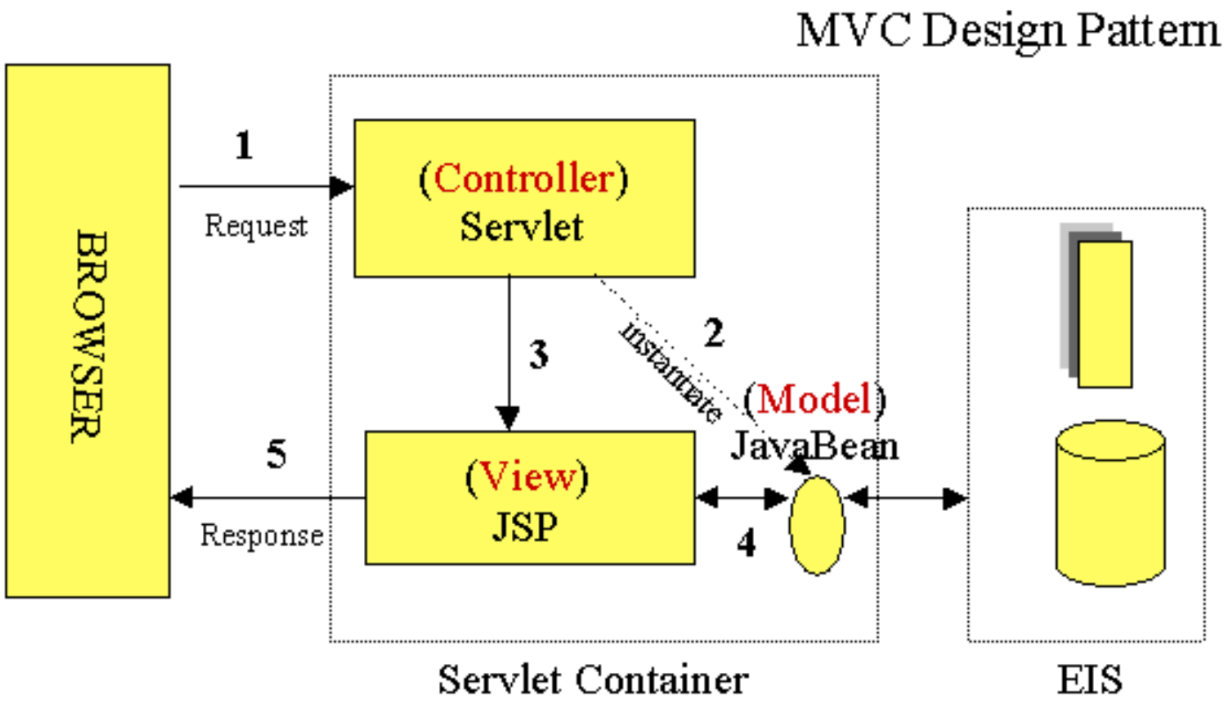


Figure 4: MVC tramite JSP, Servlet e JavaBeans

AJAX [Asynchronous JavaScript and XML]

Introduzione

AJAX è un **pattern** che consente di programmare il client.

- **Asynchronous**: rende asincrona la comunicazione tra browser (interfaccia) e server
- **JavaScript**: è il cuore del codice tramite cui funzionano le applicazioni AJAX e supporta la comunicazione con le applicazioni server
- **XML**: usato per trasferire i dati per costruire pagine web identificando i campi per il successivo uso nel resto dell'applicazione (adesso si usa di più **JSON**)
- **DOM**: utilizzato tramite JS per manipolare la struttura della pagina HTML e le risposte XML restituite dal server
- **XMLHttpRequest**: oggetto utilizzato per scambiare dati in modo asincrono con il server (sarebbe meglio chiamarlo *HTTPMessage*). È possibile aggiornare parti della pagina senza ricaricarla.

Il modello prevede di **ottenere dati XML da applicazioni Web**. Questi dati permettono di cambiare certe parti della pagina già caricata utilizzando le capacità di Javascript di modificare gli elementi del DOM

Per cosa si può usare?

- Real time form validation
- Auto completion
- Ottenere informazioni più dettagliate sui dati senza refresh della pagina
- Controlli UI sofisticati

Tramite l'utilizzo di tecniche AJAX:

È possibile rendere una pagina reattiva ad eventi generati dall'utente ma anche eseguire ciclicamente funzioni JavaScript indipendentemente da stimoli da parte dell'utente.

Le applicazioni AJAX-JSON sono più efficienti di quelle AJAX-XML?

Sì, perché i dati JSON sono in un formato compatibile JavaScript

Trasmissione dei dati tra server e applicazione RIA

- SOAP (Simple Object Access Protocol)
- XML-RPC (RPC con XML per l'encoding)
- JSON (JavaScript Object Notation)
- AMF (Action Message Format, basato su SOAP)

Un formato richiede tempo per :

- predisporre i dati lato server
- trasferire i dati
- fare il parsing dei dati
- fare il rendering dell'interfaccia

Invocazione di funzioni

- al verificarsi di un **evento**
- da codice JS
- automaticamente (si invoca da sola)

XMLHttpRequest

- `var xhttp = new XMLHttpRequest();`
- `xhttp.onreadystatechange = function(){ ... };`
- `xhttp.open(method, URL, async)` [es. xhttp.open("GET", "file.html", true)]. Se async = false, non fare la `onreadystatechange()` ma scrivi direttamente dopo la `send()`
- `xhttp.send()` [send() usato per GET, send(*string*) usato per POST]

Per metodo POST aggiungo un header HTTP con `setRequestHeader(header, value)`

Tre proprietà:

- **onreadystatechange**:
  - Stores a function (or the name of a function) to be called automatically each time the readyState property changes
- **readyState**:
  - 0: UNSENT Client has been created. open() not called yet.
  - 1: OPENED open() has been called.
  - 2: HEADERS_RECEIVED send() has been called, and response headers and status are available.
  - 3: LOADING Downloading; responseText holds partial data.
  - 4: DONE Operation is complete, and responseText is ready
- **status**:
  - 200 OK
  - 404 NOT FOUND

Risposta del server:

- xhttp.responseText
- xhttp.responseXML

Comunicare con servlet:

- `xhttp.open("GET", "GetHint?entry="+str, true);`
  - `GetHint` è la servlet
  - `entry` è il parametro

Illustrare graficamente l'architettura 3 tier nel caso del paradigma Ajax. Discutendo dal punto di vista dell'evoluzione: dove si collocano la tecnologie Servlet/JSP e JavaScript?

- JavaScript e JSP sono identificabili come View del pattern MVC.
- Le servlet sono identificabili come Controller del pattern MVC

Web Services, REST, Web API

Introduzione

- I web services stabiliscono un metodo per standardizzare la comunicazione
- La comunicazione è basata su documenti [XML]
- I servizi offerti da un operatore devono essere:
  - **descritti** per essere individuabili
  - **invocabili** in modo noto
  - **componibili**
  - pronti a **soddisfare le esigenze** del cliente e del fornitore

Cos'è un web service?

[servizio = entità software che può essere trovata e invocata da altri sistemi software]

Un web service è un'applicazione software identificata da un URI, con interfacce e binding (associazione interfaccia-protocollo-formato dei dati) che possono essere definiti descritti e scoperti da artefatti XML. Un web service supporta l'interazione diretta con altri software tramite lo scambio di messaggi XML tramite protocolli basati su Internet.

Caratteristiche chiave

- **Software as a service**:
  - si può accedere ovunque, da qualsiasi piattaforma
  - i componenti possono essere isolati in modo da esporre solo i servizi di livello business
  - disaccoppiamento tra componenti e sistemi (più stabili e flessibili)
- **Dynamic business interoperability**:
  - nuove partnership possono essere costruite dinamicamente e automaticamente
- **Accessibility**:
  - i servizi internet possono essere decentralizzati, distribuiti in giro per il mondo e vi si può accedere con molti device
- **Efficiencies**:
  - i business possono concentrarsi sui task cruciali e sul valore aggiunto dei loro prodotti
  - i web services costruiti da applicazioni per usi interni possono essere facilmente esposti per l'utilizzo esterno
  - si usa lo sviluppo incrementale
  - è più facile fare bugfixing visto che i web services sono implementati in formato leggibile da tutti
  - meno rischi, deploy più facile
- **Specifiche universali**:
  - scambio di dati, messaggi, scoperta di servizi, descrizione delle interfacce, orchestrazione dei processi business
- **Integrazione con sistemi legacy**
- **Nuove opportunità di mercato**
- I **servizi** sono **componenti indipendenti**:
  - interfaccia nota: linguaggio di descrizione standard (es. WSDL), possibile gestione automatica da parte del middleware
  - punto di accesso unico: uso di URI (URL/URL), possibile sviluppo di name services (es. UDDI)

  - scambio di dati basato su documenti: formato standard (es. XML), si supera il classico scambio di parametri

SOA: Service Oriented Architecture

I web services sono consistenti con i principi della SOA.

- i service requestor ed i service provider interagiscono
- le discovery agencies pubblicano i loro servizi sui service provider
- discovery agencies e service requestor interagiscono trovandosi a vicenda

Organismi di standardizzazione

- W3C
- OASIS
- WS-I

Stack concettuale

1. Business process: **BPEL** (Business Processes Execution Language)
2. Search and find: **UDDI** (Universal Discovery Description and Integration)
3. Description: **WSDL** (Web Services Description Language)
4. Messaging: **SOAP** (Simple Object Access Protocol)
5. Transport: **HTTP, SMTP, ...** (Internet Protocols)

Peculiarità

- Componenti pubblici ("scopribili", interfacce pubbliche)
- Componibilità (*orchestrazione* e *coreografia* di servizi composti e coordinati)
- Descrizioni semantiche (scoperta, composizione, raccomandazione di servizi, …)
- QoS (sicurezza, disponibilità, performance, *context awareness*)
- Organizzazione dei sistemi:
  - p2p (application managed)
  - ESB (middleware managed)
  - grid (given model)

Problemi

- Descrizione funzionale e QoS
- Composizione, orchestrazione, coreografia
- Semantica
- Infrastruttura (naming, gestione dei servizi, …)
- Ingegneria del software (sviluppo e deploy)
- Servizi per servizi (servizio di scoperta globale, reputazione, sicurezza, …)
- Business (modelli, processi, …)

Semantic web

- cooperazione fra siti web, risorse e servizi
- linguaggi semantici:
  - RDF: Resource Description Framework
  - OWL: Web Ontology Language
- metaservizi per supportare la scoperta di servizi:
  - servizi per i metadati
  - matchmaking

Processi business

Un insieme di attività logicamente collegate per raggiungere un obiettivo ben definito

- può prevedere particolari output al completamento e l'avvio a particolari condizioni
- può includere interazione fra i partecipanti
- durata variabile, in genere long term
- varie attività manuali e/o automatiche
- distribuito e personalizzato per le varie aziende

Workflows

Sequenza di step a cui sono sottoposti oggetti fisici e informazioni.

Includono attività, punti di decisione, varie strade, regole, ruoli

Web Processes

"Nuovi workflow": creati dalla composizione di web services scoperti

Comporre servizi web

- I service provider forniscono un'interfaccia pubblica in formato WSDL
- Composizione = set di servizi interconnessi
- Le operazioni devono essere compatibili

Orchestrazione e coreografia

- **Orchestrazione**: descrive come i WS interagiscano a livello di messaggi, includendo la *business logic*
- **Coreografia**: descrive la sequenza di messaggi che potrebbe coinvolgere più parti. Definisce lo stato condiviso delle interazioni fra businesses

BPEL

Linguaggio XML-based per la specifica formale di processi business e protocolli di interazione.

Molte feature per facilitare modeling e esecuzione di processi business basati sulla composizione di WS.

- modella il controllo dell'esecuzione
- separa definizione astratta da legame concreto
- rappresenta ruoli dei partecipanti e interazioni fra questi
- compensation support (???)

Struttura BPEL

BPEL usa WSDL per specificare le attività

- ogni processo BPEL è esposto come web service usando WSDL [entry e exit points]
- i tipi di dati WSDL sono usati per descrivere l'informazione che passa fra richieste
- WSDL può essere usato per referenziare servizi esterni richiesti dal processo business

Cinque sezioni documento BPEL

- message flow
- control flow
- data flow
- process orchestration
- fault and exception handling

Cos'è WSDL?

- Web Service Description Language: *come e dove accedere al servizio*
- Linguaggio basato su XML per descrivere i web services e come accedervi
- Descrive 4 pezzi critici di dati:
  - interfaccia che informa su tutte le operazioni disponibili pubblicamente
  - dichiarazione dei tipi di dato per tutti i messaggi di richiesta e risposta
  - informazione sul binding a livello di trasporto
  - informazioni sull'addressing per localizzare il servizio

WSDL 2.0 - concetti

Parte astratta:

- descrizione di un web service in termini di messaggi che manda/riceve attraverso (tipicamente tipizzati secondo XML Schema)
- patterns di scambio di messaggi definiscono sequenza e cardinalità dei messaggi
- operazione associa un pattern di scambio di messaggi a uno o più messaggi
- interfaccia raccoglie le operazioni in un formato per il trasporto

Parte concreta:

- i binding specificano il formato di trasporto delle interfacce
- l'endpoint di un servizio associa indirizzi network con un binding
- un servizio raggruppa gli endpoint che implementano un'interfaccia comune

Tipi di operazioni WSDL

- **one-way**: operazione riceve un messaggio e non risponde
- **request-response**: operazione riceve una richiesta e risponde
- **solicit-response**: operazione può mandare una richiesta e attende una risposta
- **notification**: operazione può mandare una richiesta e non attende una risposta

Regole per la propagazione dei fallimenti

- **Gestione dei fallimenti**:
  - WSDL specifica la propagazione dei fallimenti, non la loro generazione
  - propagazione = tentativo best effort di trasmettere il messaggio di errore al giusto destinatario
  - la generazione di un fallimento termina lo scambio
- **Il fallimento sostituisce un messaggio**:
  - qualsiasi messaggio dopo il primo del pattern può essere sostituito con un messaggio di fallimento, che DEVE avere la stessa direzione e lo stesso destinatario
  - se non c'è un modo di arrivare al destinatario, il fault viene scartato
- **Messaggio scatena l'errore**:
  - qualsiasi messaggio, primo incluso, può scatenare un messaggio di fallimento in risposta
  - ogni destinatario può propagare l'errore e deve propagare al massimo un fallimento per messaggio di errore
  - la direzione del messaggio di errore è l'inversa di quella del messaggio di arrivo
  - il messaggio di errore DEVE essere mandato a colui che ha mandato il messaggio che ha scatenato l'errore
  - se non c'è un modo di arrivare al destinatario, il fault viene scartato
- **No fallimenti**:
  - nessun fallimento viene propagato

SOAP

- **protocollo basato su XML** per consentire a componenti software e applicazioni di **comunicare tramite protocolli internet standard** (HTTP, SMTP, …)
  - **Stateless**
  - **one-way** (richiesta/risposta, può passare per intermediari, no multicast)
  - ignora la semantica dei messaggi scambiati
  - l'effettiva interazione deve essere scritta nel documento **SOAP**
  - ogni pattern di comunicazione deve essere implementato dai sistemi sottostanti
- modo standard di strutturare messaggi XML
  - applicazione delle specifiche XML
  - si affida a XML Schema e XML Namespaces
- slegato da linguaggi/piattaforme
- semplice e estensibile
- passa dati strutturati e tipizzati
- serve ad accedere a registri ed invocare servizi
- **procedure remote vs. document oriented** —> *gli endpoint definiscono la semantica*

Componenti messaggio SOAP

- **SOAP envelope**:
  - definisce contenuti messaggio (cos'è, a chi è indirizzato, opzionale/obbligatorio)
- **SOAP header (opzionale)**:
  - flessibile, può essere processato dai nodi intermediari
  - contiene informazioni su come processare il messaggio: routing, autorizzazioni, …
- **SOAP body**:
  - messaggio effettivo
  - informazioni su "chiamata e risposta"

UDDI

- Nasce per risolvere il problema del **trovare i web services**
- Universal Description, Discovery and Integration
- Serve ai business per scoprirsi facilmente, velocemente, dinamicamente e fare transazioni
- UDDI permette a un business di:
  - descrivere se stesso e i propri servizi
  - scoprire altri business
  - integrarsi con altri business
- Due parti: specifiche tecniche e UDDI business registry

Approcci per scoprire servizi

- Registro:
  - servizio deve essere pubblicato dall'entità che lo possiede
  - il proprietario decide tutto: cosa mettere, cosa aggiornare, …
  - esempio: UDDI (anche se è anche index)
- Indice:
  - ognuno può creare il proprio
  - pubblicazione passiva
  - può includere informazioni di terze parti
  - esempio: Google
- P2P:
  - i servizi si scoprono a vicenda dinamicamente
  - mando richiesta ai vicini; se non mi sanno aiutare chiedono ai loro vicini e così via

Accedere a UDDI via API

SOAP e XML Schema sono le basi

REST vs SOAP

SOAP: Client -> SOAP+XML[+HTTP o altro protocollo] -> Server [WSDL]

REST: Client -> JSON/PO-XML/RSS su HTTP -> Server [WADL]

Protocollo di trasporto:

- REST: HTTP
- SOAP: vari protocolli

Formato dei messaggi:

- REST: vari formati (XML-SOAP, JSON, RSS)
- SOAP: XML-SOAP (envelope, header, body)

Identificatori di servizi:

- REST: URI
- SOAP: URI e WS-addressing

Descrizione del servizio:

- REST: documentazione testuale o WADL (Web Application Description Language)
- SOAP: WSDL

Composizione di servizi:

- REST: mashup
- SOAP: BPEL

Scoperta di servizi:

- REST: no standard
- SOAP: UDDI

REST

- semplificare gli accordi
- nomi = le risorse a cui ci si riferisce
- verbi = operazioni sulle risorse nominate
- content types = che rappresentazioni delle informazioni sono disponibili
- REpresentational State Transfer
- stile architetturale per sistemi distribuiti
- risorse definite da URI
- risorse manipolate tramite rappresentazioni (desiderate: campo **Accept**; restituite: campo **Content-Type**; specificate con MIME)
- più rappresentazioni per una risorsa
- messaggi autodescrittivi e stateless
- stato dell'applicazione guidato dalla manipolazione di risorse
- hypermedia per controllare il comportamento dell'applicazione:
- transizioni di stato del client avvengono tramite links
- link determinati e forniti dal server
- i principi REST si addicono ai componenti architetturali del web: URI e HTTP
- si usano GET, POST, DELETE, PUT, HEAD, OPTIONS di HTTP
- stateless: no stato, no sessione. Client mantiene lo stato attraverso i link
- vantaggi:
- scalabile

- basso accoppiamento tra le risorse: no informazioni di sessione condivise
- caching migliora le prestazioni
- load balancing più facile: meno stati di comunicazione
- è semplice: meno software specializzato
- identificazione tramite meccanismi standard
- uso completo e corretto protocollo HTTP
- meccanismo leggero e stratificato per integrazione di dati e servizi
- piattaforma distribuita e guidata dagli hypermedia
- si può usare per riscrivere applicazioni CRUD:
- POST = CREATE [Location: URI/nuova/risorsa; Messaggio: 201 Created]
- GET = RETRIEVE
- PUT = UPDATE
- DELETE = DELETE
- [PATCH aggiorna parzialmente una risorsa; idempotente]
- [OPTIONS per ricevere info su una risorsa o sul server]

Il web come sistema

- sistema hypermedia distribuito, composto principalmente da HTML, URI e HTTP
- aperto, scalabile, estendibile, facile da capire

Web servers

- pagine possono essere richieste da un client/caricate sul server/rimosse dal server
- client può inviare risorse (HTTP POST)
- per interazioni più complesse sono richieste più operazioni
- **WebDAV**: Web Distributed Authoring and Versioning, definisce metodi HTTP addizionali (sarebbe meglio utilizzarli per più tipi di dati)

Tecnologie web

- devono funzionare RESTful:
- risorse navigabili e identificate da URI: NON POSSONO ESSERE ACCEDUTE/MODIFICATE DIRETTAMENTE, si lavora con rappresentazioni
- se non puoi nominare qualcosa, non lo puoi usare
- accesso alle risorse via HTTP, hanno stato
- transizioni di stato modellate come accessi a risorse
- se le applicazioni funzionano diversamente, non sono tecnologie web
- plugin piuttosto che risorse "single-presentation": no PDF e applet Java
- documenti = concetti astratti di risorse descrittive

Stato

- parte del contenuto trasferito
- il client può switchare fra vari server
- i client possono immagazzinare differenti rappresentazioni per salvare lo stato
- trasferimento di stato rende il sistema scalabile
- non è state-specific
- stato trasferito fra client e server
- loose coupling (client e server sono disaccoppiati tramite il trasferimento di stato)

Nomi

- nomi di risorse [URI]

- qualsiasi cosa di interesse dovrebbe avere un nome
- separare i nomi da verbi e rappresentazioni migliora l'estendibilità

Verbi

- operazioni sulle risorse
- **core idea** di REST: usare solo verbi universali (possono essere applicati a tutti i nomi)
- spesso i metodi HTTP (GET POST PUT DELETE) sono sufficienti

Content type

- le rappresentazioni dovrebbero essere processabili dalla macchina
- le risorse sono astrazioni, REST passa rappresentazioni
- le risorse hanno varie rappresentazioni (ovvero content types)
- aggiungere o cambiare content types non cambia l'architettura: client e server differenti supportano differenti content type
- content negotiation fa sì che i content types vengano negoziati dinamicamente

Regole per progettare applicazioni RESTful

1. Identificare le risorse = progettare le URI
2. Scegliere le rappresentazioni da utilizzare (preesistenti o nuove)
3. Definire la semantica dei metodi (rispettando la natura dei metodi stessi)
4. Scegliere i codici di risposta:

- 2xx Success
- 4xx Client error
- 5xx Server error

Perché i servizi REST sono più conformi al modello del Web di quelli SOAP?

Perché usano HTTP in modo nativo.

Cosa significa il termine stateless nel contesto della Service Oriented Architecture?

Che lo stato di un servizio non dipende dallo stato di un altro servizio

Architettura a 3 livelli nel modello client-server

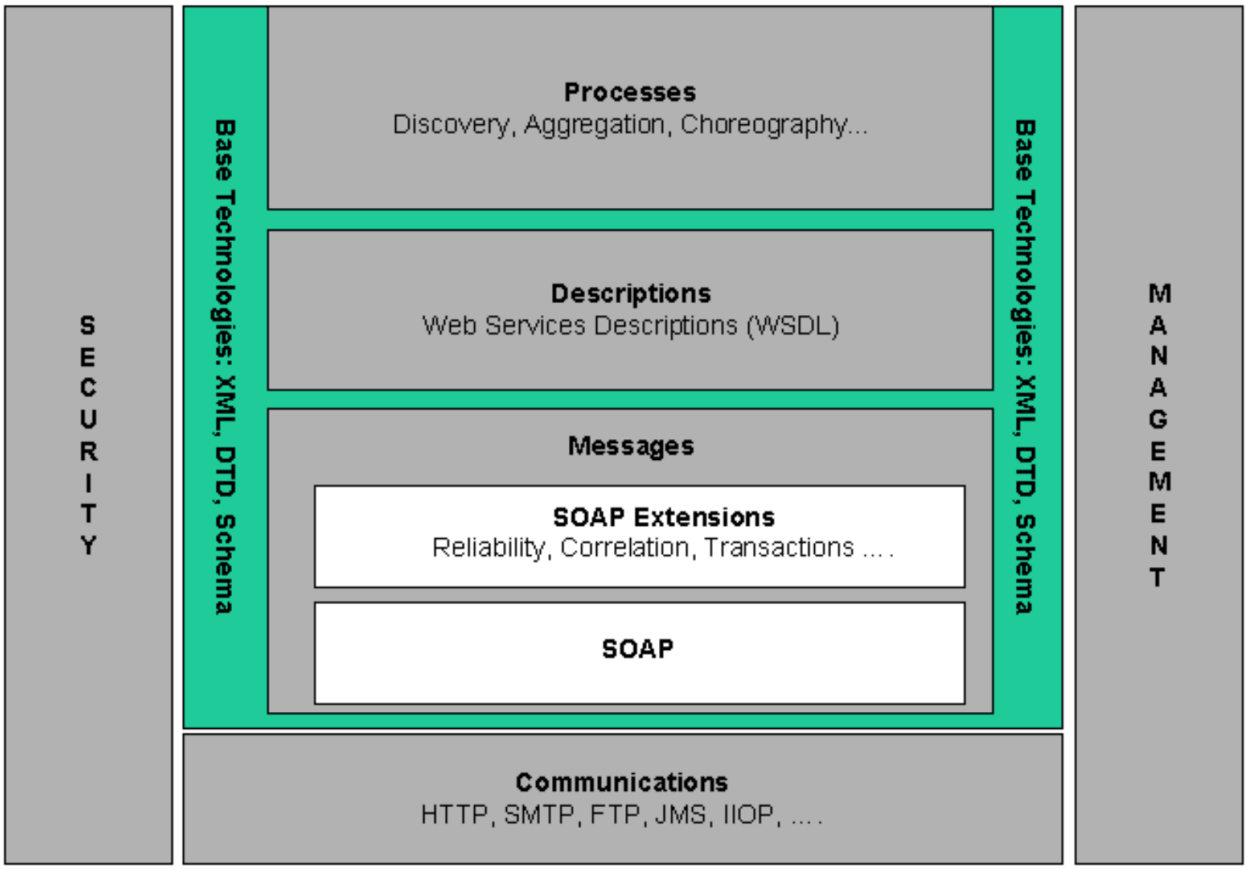
Si separano **interfaccia utente**, **gestione della logica funzionale** (business logic) e **gestione dei dati persistenti**.

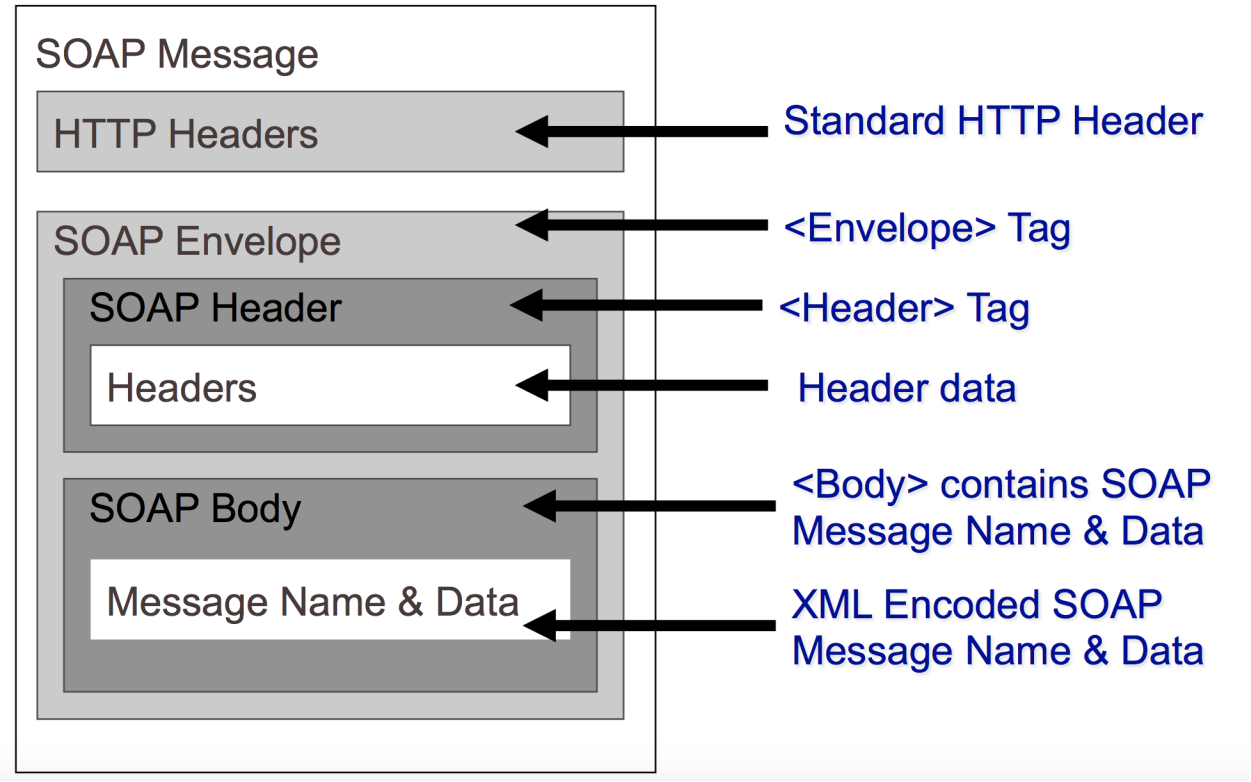
Nel caso del modello client-server:

- l'interfaccia verso l'utente è rappresentata da un Web server e da eventuali contenuti statici (es. pagine HTML);
- la business logic corrisponde a una serie di moduli integrati in un application server per la generazione di contenuti dinamici (per esempio, moduli Java EE su JBoss)
- i dati (acceduti dalla business logic) sono depositati in maniera persistente su un DBMS (data layer). Può risiedere sulla stessa macchina host dell'application server oppure su macchina host dedicata e separata.

Il modello detto hypermedia control dello stile architetturale REST significa che...

- Gli hypermedia sono il motore dello stato dell'applicazione. Stato mantenuto **dal client**; il server mantiene lo stato delle risorse.
- Hypermedia = link e form
- Seguire un link = transizione di stato
- Il server guida il client verso nuovi stati presentando di volta in volta links diversi
- Rappresentazione degli hypermedia = interfaccia





JavaScript e jQuery

Linguaggio di scripting

Linguaggio di programmazione per l'automazione di compiti altrimenti eseguibili da un utente umano all'interno di un ambiente software.

Spesso interpretato e molto dinamico

Cos'è JavaScript?

È un linguaggio di scripting interpretato da un engine. Ha la capacità di effettuare richieste HTTP al server in modo trasparente all'utente e rende asincrona la comunicazione tra browser e web server

- JavaScript è un linguaggio di programmazione interpretato inizialmente progettato per permettere l'esecuzione di script all'interno di browser web, lato client, per l'interazione con l'utente, la validazione di dati all'interno di form, la modifica di documenti web senza effetto 'pagina bianca'
- JavaScript è dinamico, debolmente tipizzato, la cui sintassi è stata influenzata dal C e da Java

Esempio di jQuery

$(document).ready(function() { $('body').hide().fadeIn('slow); });

Operatori di confronto

- === e !== NON FANNO conversione di tipo
- == e != FANNO conversione di tipo

Arrays

var mioArray = [1, ‘ciao’, [1, 2]];

Strutture dati generalizzate

var myauthor = {name: ‘Ernest Hemingway’, “born”: 1899, died: 1961 };

Molto simile a JSON!

var cat = {colour: “grey”, name: “Spot”, size: 46}; cat.size = 47;

delete cat.size; // Now (cat.size == undefined)

cat.weight = 5.5;

null e undefined

Attenzione null e undefined sono oggetti veri e propri, il primo rappresenta una variabile definita ma con valore ‘vuoto’, la seconda un variabile non definita; inoltre: null == undefined ma null !== undefined

Dato l’oggetto javascript definito come segue: var cat={colour:”grey”, name:”Spot”,size: ecc...}

Per accedere al campo name si deve indicare cat.name

In Javascript è possibile modificare dinamicamente il contenuto e la struttura di una pagina HTML, per selezionare un elemento della pagina...

Si possono utilizzare funzioni Javascript di base offerte dal browser quali ad esempio document.getElement oppure funzionalità offerte da framework come ad esempio i selettori di jQuery

Dato l’array javascript definito come segue: var myarr = {“Primo”,“Secondo”,“Terzo”}

Per aggiungere un valore numerico 4 in coda un comando adatto è myarr[myarr.length]=4

Come gestisce il tipo di dato delle variabili javascript?

Il tipo cambia in base a ogni assegnamento

Le variabili in javascript sono

Non tipizzate.

In JavaScript è possibile modificare dinamicamente il contenuto e la struttura di una pagina HTML; per selezionare un elemento della pagina...

Si possono utilizzare funzioni JavaScript di base offerte dal browser quali ad esempio document.getElementById() oppure funzionalità offerte da framework come ad esempio i selettori jQuery

jQuery

jQuery è un framework JavaScript che rende molto semplice la scrittura di applicazioni web offrendo funzionalità quali - manipolazione di HTML/DOM e CSS - metodi per eventi HTML - effetti e animazioni - supporto a programmazione AJAX - varie altre utilità (anche tramite plugin)

Sintassi `$(selector).action()`

Funzione each

$(document).ready(function() { ($'span.pq').each(function(){var quote =(this).clone(); quote.removeClass('pq'); quote.addClass('pullquote'); $(this).before(quote); }); // end each }); // end ready

`before` serve ad aggiungere la variabile al documento, appena prima dello span stesso

La funzione `toggle(function1, function2)` permette di associare a un elemento due diverse funzioni, da chiamare rispettivamente per ogni click dispari e pari

Eventi jQuery

- `.click`
- `.focus`
- `.dblclick`
- `.mouseover`

Varie jQuery

All'atto del caricamento voglio che il cursore sia posizionato nella prima text area di input

$(document).ready(function() { $(':text:first').focus(); }

- `.attr(attributo, newValue)`
- `.css(attributo, newValue)`
- `.slideUp()`
- `.slideDown()`
- `.fadeIn()`
- `.fadeOut()`
- `.setInterval(funzione, millisecondi)`

Altro

Le applicazioni native in Android in che formato sono?

Sono in .apk

Quando un Thread invoca il metodo wait() succede che:

L'esecuzione viene sospesa a tempo indeterminato e il lock rilasciato.

Il quantificatore synchronized applicato ad un blocco di codice Java significa che viene eseguito in mutua esclusione rispetto:

Agli altri metodi o blocchi dello stesso oggetto.

Un vantaggio della virtualizzazione è che

Viene garantita l'integrità del sistema anche a fronte di fallimenti software

Quando un thread, in un programma Java, riceve una notifica attraverso il metodo notifyAll, in quale stato viene spostato?

Stato di Ready