

Middleware per SD

□ Standard

- **OMG OMA (Object Management Architecture)**
- **OMG CORBA (Common Object Request Broker Achitecture)**

◆ <http://www.omg.org>
◆ <http://www.acl.lanl.gov/OMG/CD/corba.html>

□ Tecnologia

- **Microsoft™ DCOM® (Distributed Component Object Model)**

◆ <http://www.microsoft.com/com/>

- **Famiglia Sun™**

◆ Java RMI, Jini, JDK ORB
◆ <http://java.sun.com>

Introduzione a OMA e CORBA



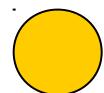
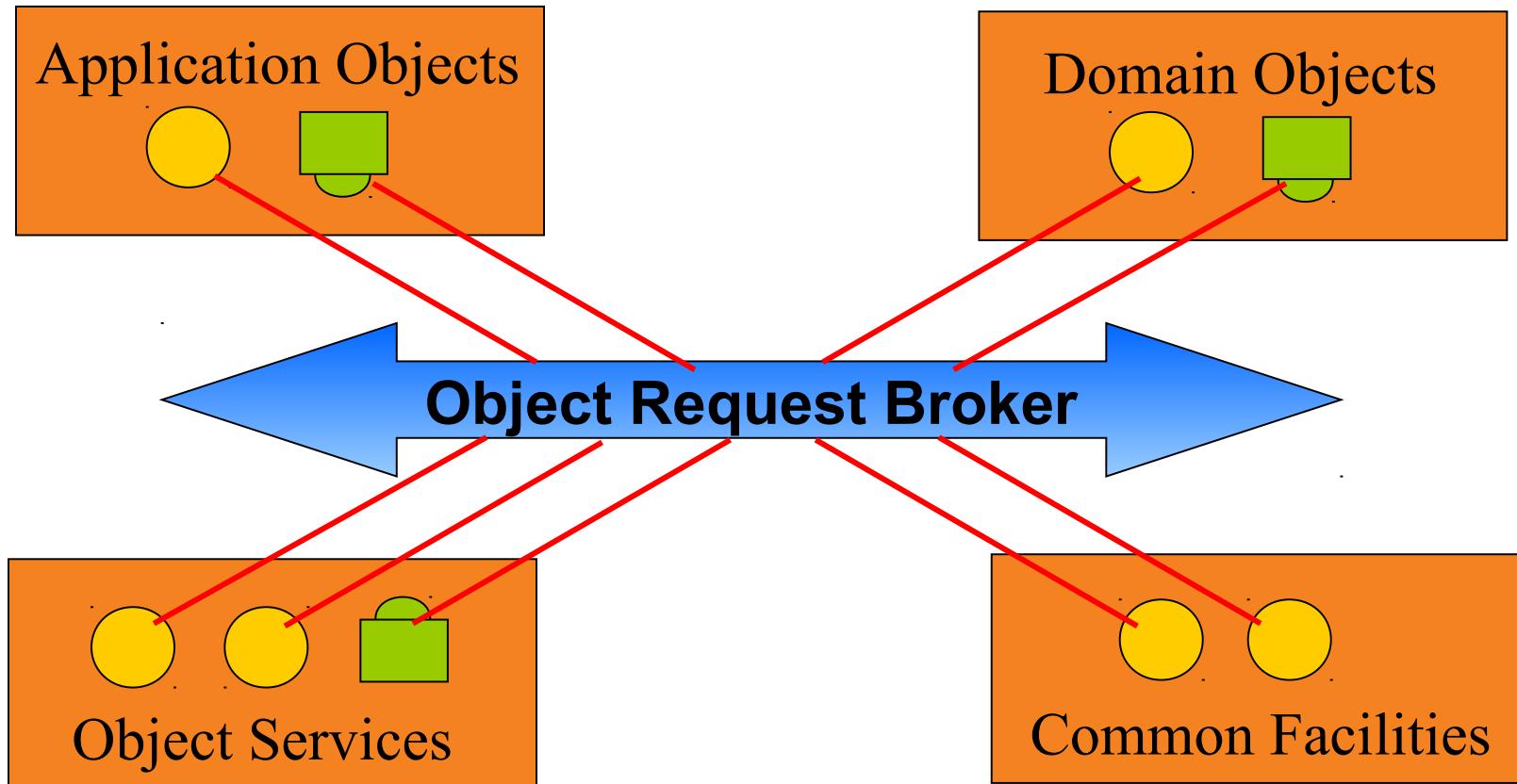
- ❑ sviluppare applicazioni distribuite, i cui componenti collaborino in modo
 - efficiente ed affidabile
 - trasparente
 - scalabile
- ❑ per aiutare a portare a termine questa sfida, l'OMG ha creato CORBA
 - OMG (Object Management Group) è un consorzio di aziende informatiche: Sun, HP, DEC, Iona, ...

L'OMA core Object Model



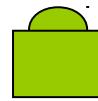
- definisce l'insieme dei concetti di base
- oggetti
- operazioni (signature, parametri e valori di ritorno)
- tipi (non oggetti)
- interfacce
 - sostituibilità
 - ereditarietà (*subtyping*)

L'OMA Reference Model



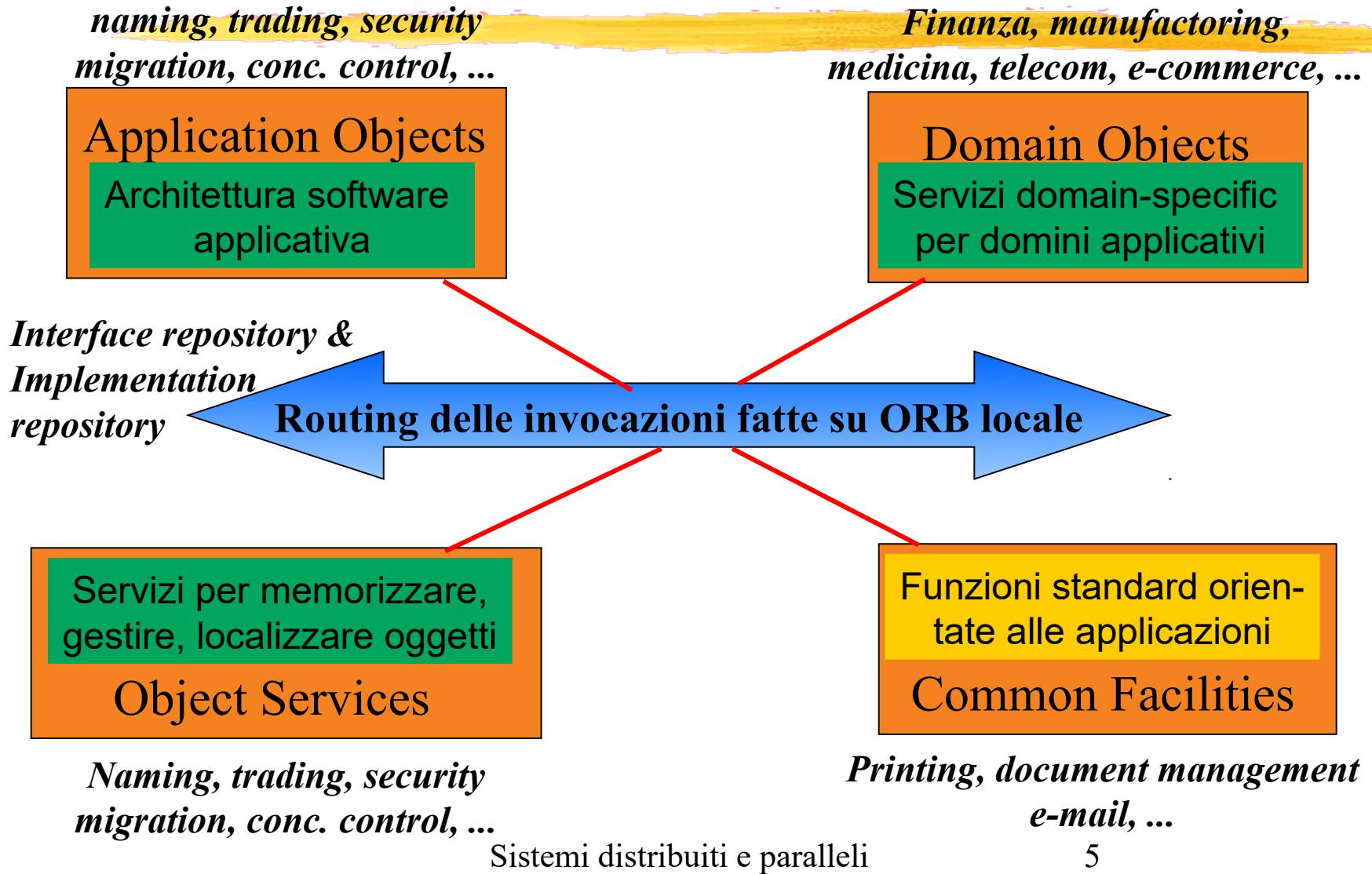
CORBA Object

Sistemi distribuiti e paralleli



Legacy Application Wrapper

L'OMA Reference Model



CORBA



- **Common ORB Architecture**
- si basa sul *Core Object Model* di OMA, e offre
 - la specifica delle funzionalità dell'ORB
 - la sintassi Interface Definition Language (IDL)
 - supporto all'interoperabilità con due protocolli (DII e DSI)
 - un insieme di mapping verso linguaggi di programmazione (C, C++, Smalltalk, Java, ADA95)

Contributi CORBA



- **CORBA lancia due sfide allo sviluppo di sistemi distribuiti:**
 - rendere lo sviluppo di applicazioni distribuite non più difficile dello sviluppo di applicazioni centralizzate (difficile anche solo per i problemi intrinseci)
 - fornire un'infrastruttura per integrare componenti software in un SD
 - ◆ in questo senso CORBA è una “*enabling technology*”

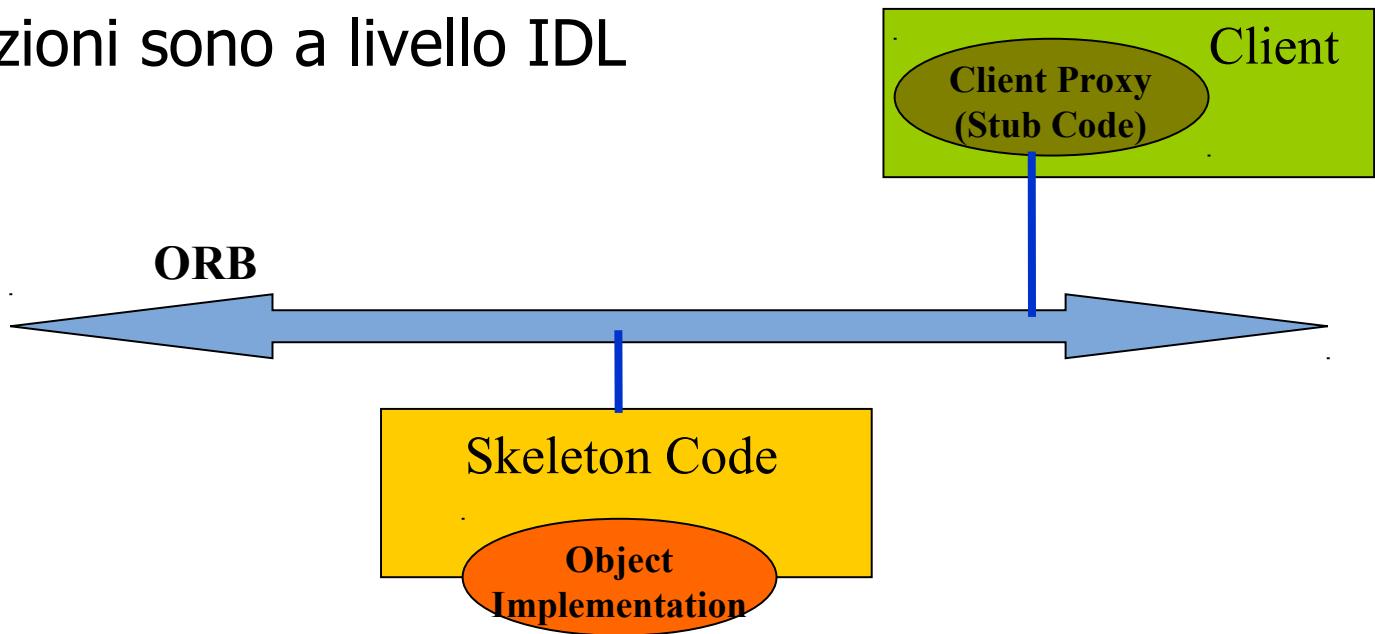
Principi di CORBA

□ trasparenza della distribuzione

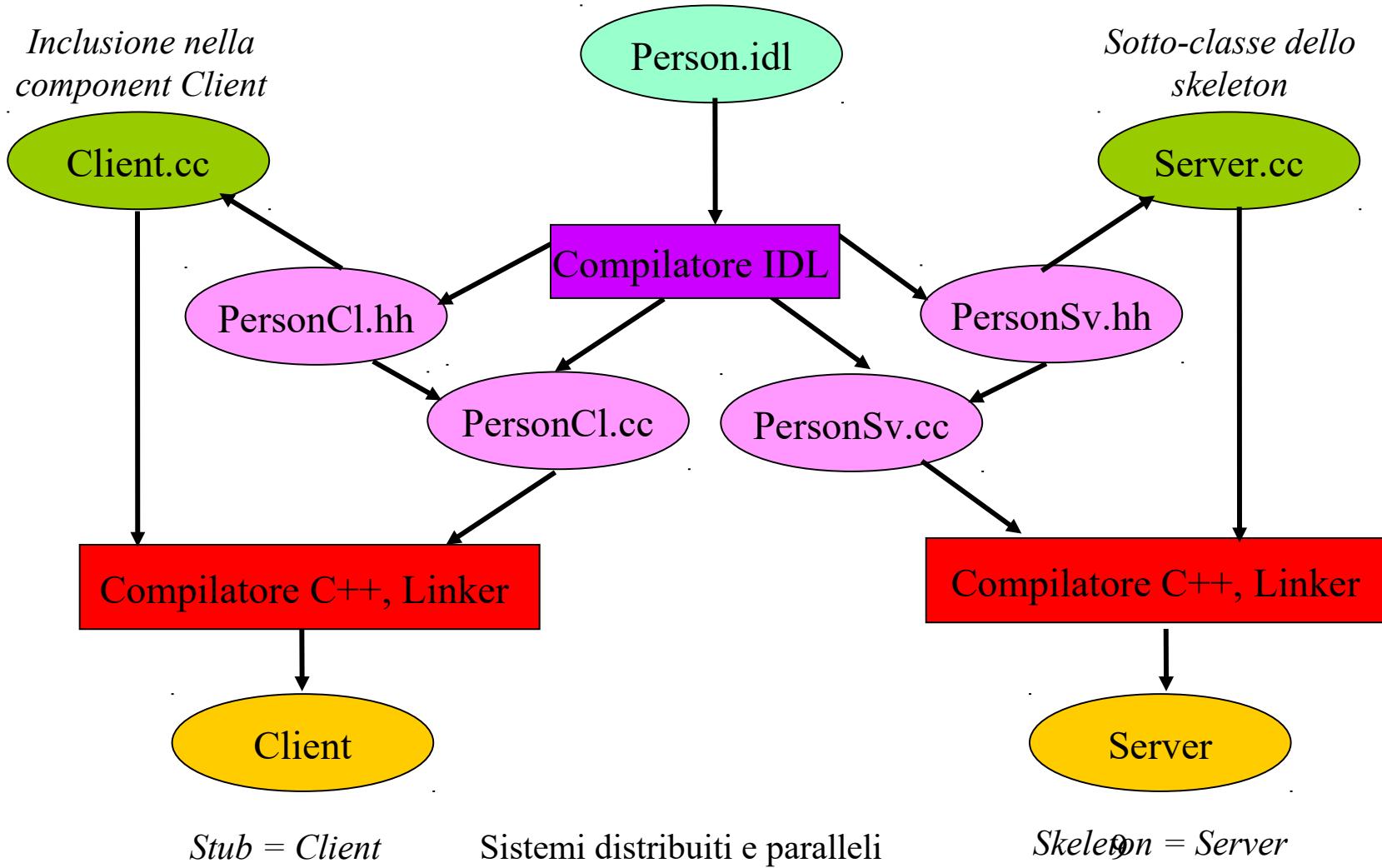
- gli oggetti CORBA invocano operazioni localmente sull'ORB

□ trasparenza dell'implementazione

- le invocazioni sono a livello IDL



Generazione di Stub e Skeleton



Interfaccia dell'ORB (Il Client)



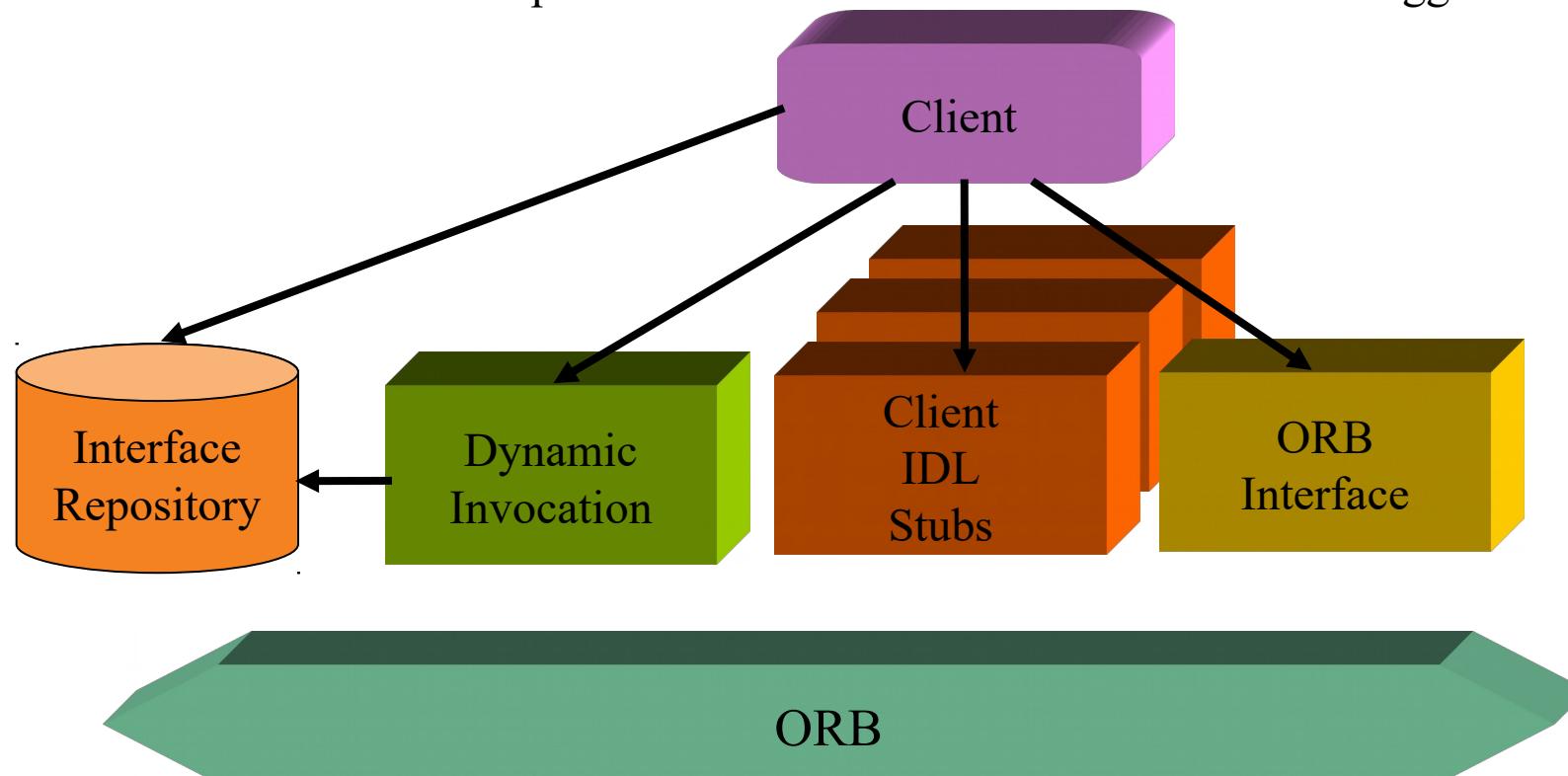
(DII) costruzione a runtime del metodo da invocare



Proxy per un server remoto (per ogni interfaccia del server)



API locale offerta al client per memorizzare e comunicare riferimenti a oggetti



Interfaccia dell'ORB (Il Server)



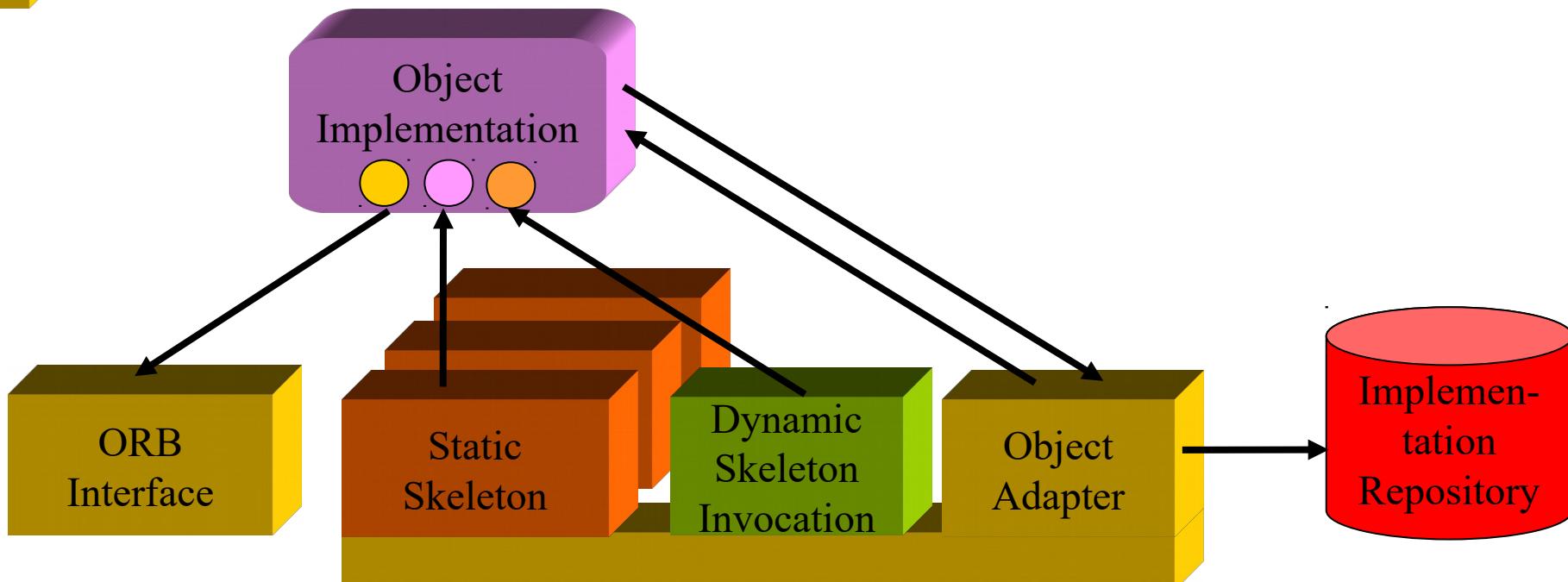
(DSI) mapping a runtime degli oggetti/metodi invocati



Interfaccia statica per ogni servizio esportato da un server

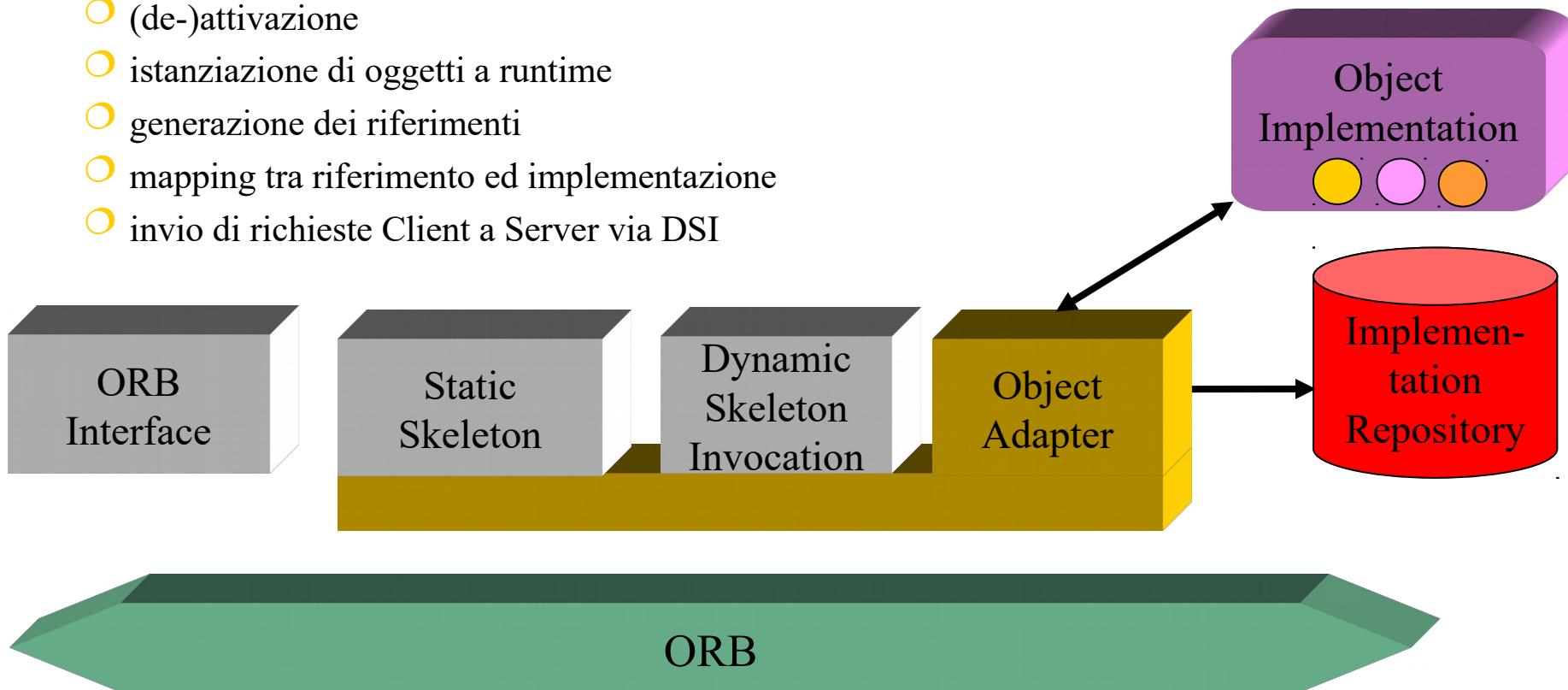


Accetta richieste per conto di un oggetto, e ne gestisce il ciclo di vita



L'Object Adapter

- Interfaccia tra ORB ed Object Implementation (server)
- Offre servizi di:
 - registrazione dei server
 - (de-)attivazione
 - istanziazione di oggetti a runtime
 - generazione dei riferimenti
 - mapping tra riferimento ed implementazione
 - invio di richieste Client a Server via DSI



Object Adapter: BOA



□ Basic Object Adapter (BOA)

- specifica formale comune a tutti gli ORB
- insieme di interfacce per gestire riferimenti a oggetti e relative implementazioni (***servant***)
 - ✧ mapping riferimento oggetto e servant relativo
 - ✧ attivazione trasparente degli oggetti
 - ✧ associa agli oggetti politiche per mapping tra invocazione e servant corrispondente
 - ✧ gestione della persistenza

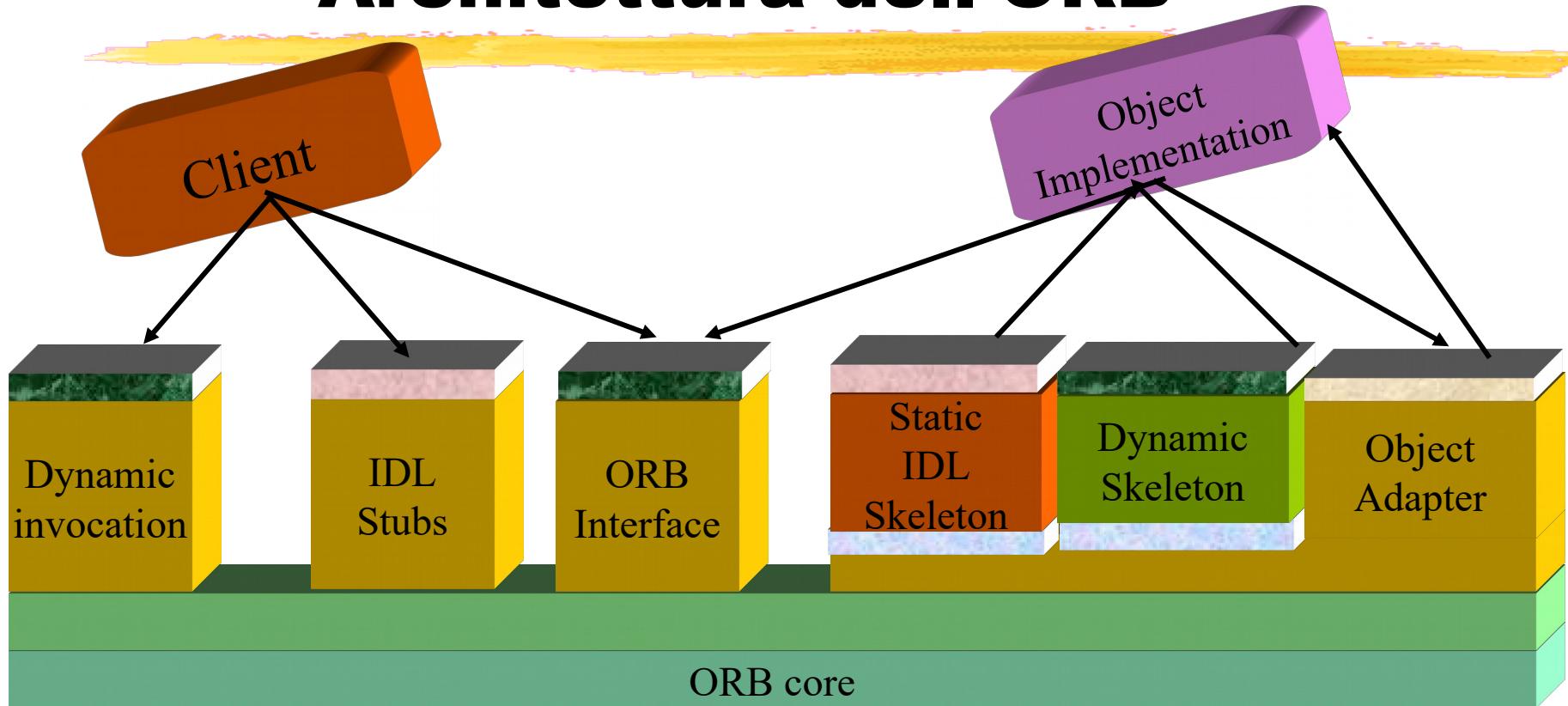
Object Adapter: POA



❑ Portable Object Adaptor (POA)

- l'implementazione di un oggetto deve esplicitamente registrare gli oggetti per essere attivata (sistema o administrator)
- gli oggetti ed i loro riferimenti sono noti all'ORB (no implementation repository)
- specificata da OMG in modo (troppo) vago ...

Architettura dell'ORB



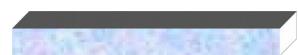
Multiple



Generated for each object type



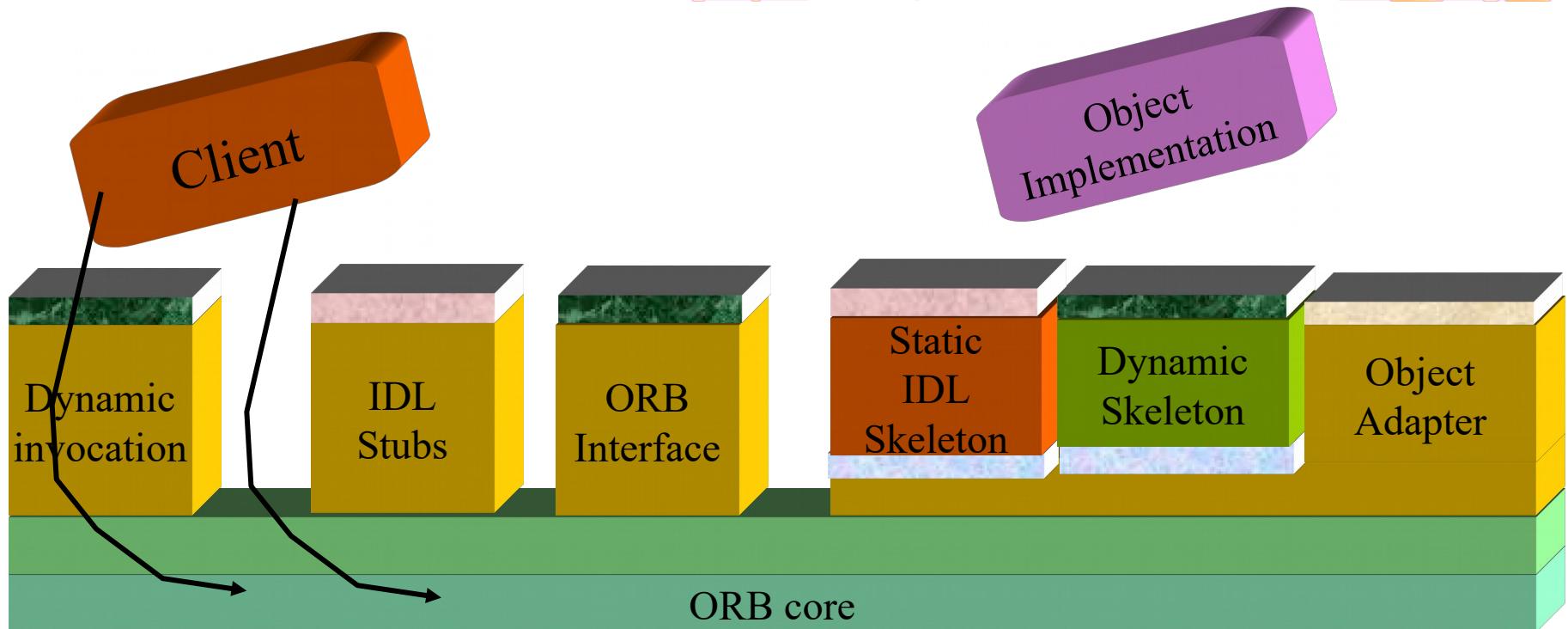
Identical for all ORB implem.



(standard)

ORB-dependent

Richiesta locale (via ORB)



Multiple

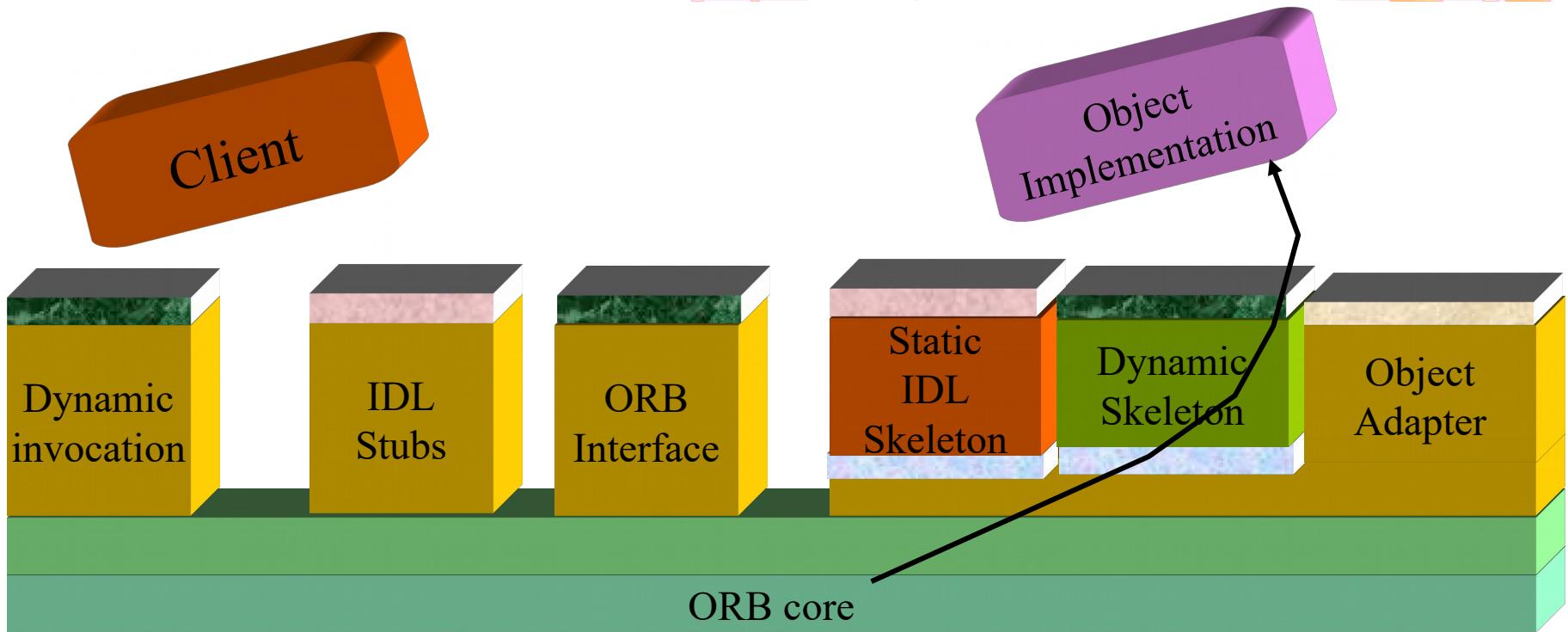
Generated for each object type

Identical for all ORB implem.

(standard)

ORB-dependent

Servizio remoto



Multiple



Generated for each object type



Identical for all ORB implem.



(standard)

ORB-dependent

Sommario



- **Lo standard CORBA comprende diverse parti:**
 - ORB (Object Request Broker)
 - BOA (Basic Object Adapter)
 - POA (Portable Object Adapter)
 - repository per le interfacce e per le implementazioni
 - IDL (Interface Definition Language)
 - SII (Static Invocation Interface)
 - DII (Dynamic Invocation Interface)
 - DSI (Dynamic Skeleton Interface)

CORBA IDL



- Lo sviluppo di applicazioni flessibili distribuite su piattaforme eterogenee richiede una rigida separazione fra interfaccia e implementazioni
- Benefici nell'uso di un IDL:
 - indipendenza dalla piattaforma
 - indipendenza dal linguaggio
 - obbligo alla modularità
 - aumento della robustezza

Altri IDL



- **La maggior parte degli IDL disponibili sono di tipo procedurale (DCE IDL, ...)**
 - complessi da estendere perchè non usano l'ereditarietà
- **L'IDL di CORBA:**
 - ha ereditarietà semplice e multipla
 - supporta molti linguaggi
 - si basa sul C++

L'OMG IDL compiler



- Genera ***client stub*** e ***server skeleton***
- **Stub e skeleton - in accordo con l'ORB**
-automatizzano le seguenti attività:
 - client proxy factories
 - conversione dei parametri (*marshalling*)
 - generazione dell'interfaccia per le classi
 - registrazione ed attivazione degli oggetti
 - localizzazione e *binding* degli oggetti

Caratteristiche dell'IDL OMG

- È un soprinsieme di un sottoinsieme del C++:
 - non è un linguaggio completo di programmazione, ma definisce solo le interfacce;
 - supporta le seguenti caratteristiche:
 - ◆ moduli, interfacce
 - ◆ operazioni, attributi, ereditarietà
 - ◆ tipi base, array, sequenze, struct, union
 - ◆ costanti
 - ◆ eccezioni

Static Invocation Interface (SII)

- ❑ Le operazioni IDL sono definite *prima* che un client sia implementato
 - è il modo più comune per usare l'IDL
 - tutti i metodi sono specificati in compilazione e sono resi noti al client e al server via stub
 - vantaggi principali:
 - ✧ semplicità
 - ✧ type-safety (controllata dal compilatore)
 - ✧ efficienza

Dynamic Invocation Interface (DII)

- **Le operazioni IDL del server non sono note quando il client è implementato**
 - modo meno comune di usare l'IDL
 - usa meta-dati immagazzinati nell'interface repository
 - **vantaggi principali:**
 - ✧ permette agli oggetti e ai loro metodi di essere specificati e invocati a run-time
 - **svantaggi:**
 - ✧ è più complicata
 - ✧ meno type-safe (controllo nel repository a runtime), più inefficiente della SII

Dynamic Skeleton Interface (DSI)



- **Fornisce al server una funzionalità analoga a quella che la DII fornisce al client**
- **Definita in CORBA 2.0 principalmente per costruire dei ponti con l'ORB**
- **Permette al codice del server di trattare metodi arbitrari su oggetti CORBA**

Riferimenti agli oggetti

- Un riferimento ad un oggetto è una *handle* opaca agganciata ad un oggetto
- I riferimenti agli oggetti possono essere passati a processi su altri nodi:
 - l'ORB li converte in una forma trasmissibile via rete (una stringa tipo URL)
- I riferimenti agli oggetti sono una potente caratteristica di CORBA

Attivazione di oggetti



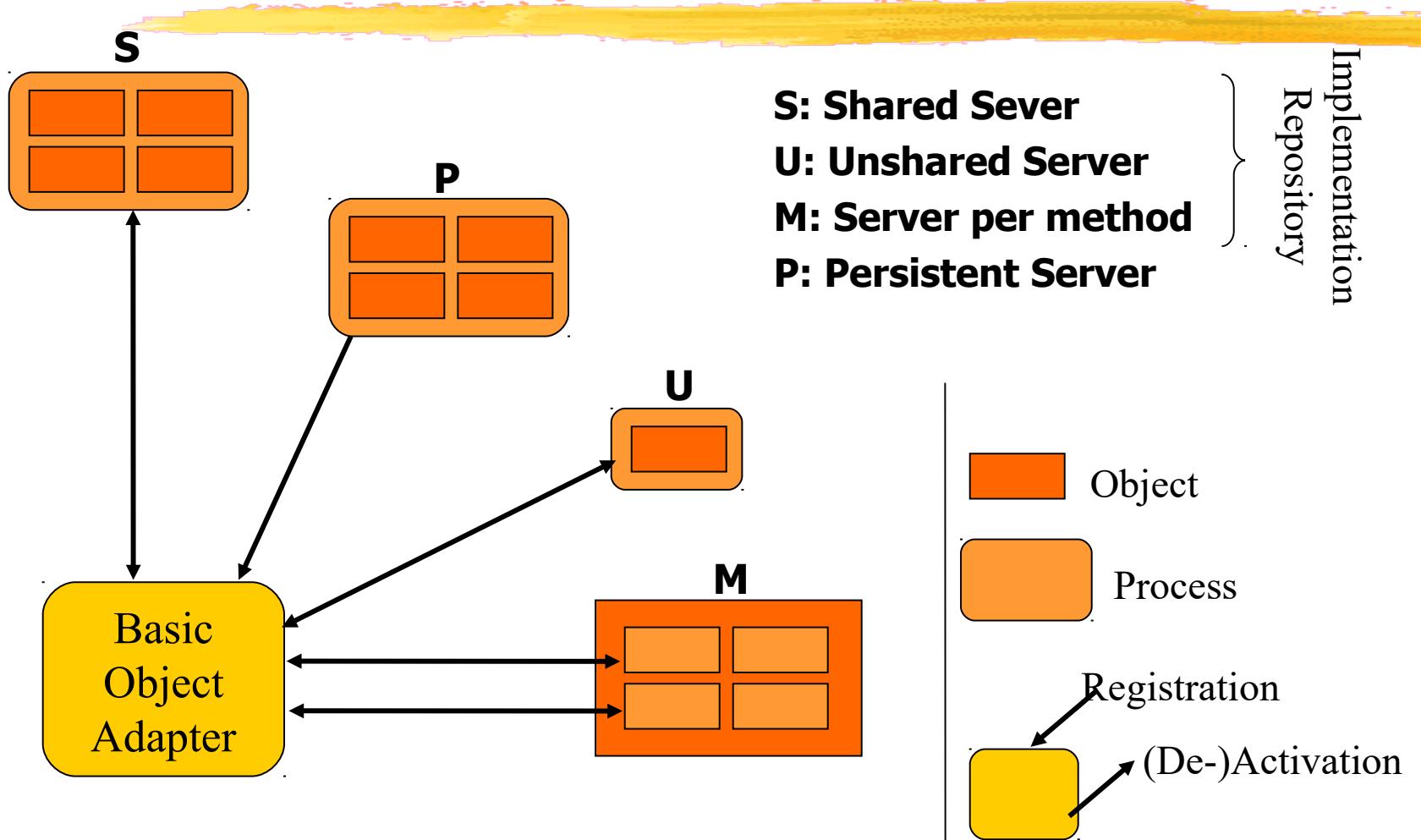
□ Concetto

○ se un oggetto non è attivo nel momento in cui un client ne invoca un metodo, l'ORB lo attiva

□ Gli oggetti devono essere registrati presso l'ORB nell'*implementation repository*

□ Gli oggetti possono essere installati su qualsiasi macchina

Modalità di attivazione



Valutazione di CORBA



□ ***Curva di apprendimento***

- dura, da ammortizzare su tanti progetti

□ ***Interoperabilità e portabilità***

- migliorata nella versione 2.0 ma non ancora perfetta (specie sui servizi ad alto livello)

CORBA 3 (Ago '99)

□ Quali estensioni?

○ Integrazione con Internet

- ◊ Inter Orb Protocols (CORBA 2)
- ◊ Firewall: port per IIOP e per IIOP over SSL
- ◊ Supporto a Callback e Notification
- ◊ Interoperatività con Name Service
 - riferimento in formato URL per identificare servizi predefiniti di cui non si conosce il REF (Incl. Naming Service)

CORBA 3 (cont.)

□ Controllo QoS (Quality of Service)

- Client e oggetti server possono
 - ✧ controllare l'ordering (by time, priority, or deadline)
 - ✧ set priority, deadlines, e time-to-live
 - ✧ set tempo di start/end per invocazioni time-sensitive
 - ✧ controllare politiche di routing e conteggio di network routing hop

□ Architettura

- Verso il *componentware* (integrazione con componenti Java, XML, ...)

Integrare le applicazioni: da clipboard a OLE

- Windows è nato con un obiettivo ben preciso: spostare l'integrazione fra funzioni applicative a livello di ambiente operativo: l'utente può quindi scegliere sul mercato gli applicativi specializzati migliori lasciando al sistema operativo il compito di integrarli. Per realizzare questo obiettivo deve mettere a disposizione meccanismi efficaci per la comunicazione fra applicazioni. L'evoluzione del sistema è strettamente legata all'introduzione di meccanismi sempre più potenti per consentire questa comunicazione
- Si è passati dalla clipboard, al DDE (Dynamic data exchange: un protocollo di comunicazione asincrono basato sui messaggi) a OLE
- OLE consente di inserire all'interno di un documento prodotto da un'applicazione (p.es. Word) un sottodocumento prodotto e gestito da un'altra applicazione (p.es una tabella Excel)
- La prima versione di OLE, risalente al '90, si basava su DDE ed era molto fragile e lenta. Era necessario trovare un meccanismo più robusto ed efficiente su cui poggiare OLE

Problemi del modello Windows

- Il modello di base dei sistemi windows è cresciuto molto a partire dalla versione 1.0 (1985). Questa crescita è avvenuta senza sostanziali adeguamenti di architettura
- L' API (Application Program Interface) è molto vasta (più di 1000 funzioni), cresciuta disordinatamente, con nomi di funzioni attribuiti senza una regola precisa
- Le DLL costituiscono una buona base per la modularità, ma presentano alcuni inconvenienti:
 - Dipendenza dalla collocazione fisica: le applicazioni carico le DLL facendo riferimento al path in cui si trovano e se una DLL viene spostata le applicazioni non riescono più ad accedere ai servizi
 - Gestione delle versioni: le DLL non possiedono meccanismi intrinseci di gestione delle versioni. Questa problematica viene lasciata alla buona volontà e alla disciplina degli sviluppatori con grossi rischi di compatibilità.
- Occorre un meccanismo migliore per comunicare fra le applicazioni e sistema operativo e in generale fra fruitori e fornitori di servizi

La storia si ripete

- Le singole applicazioni sono cresciute a dismisura mantenendo una struttura essenzialmente monolitica.
- La continua aggiunta di funzionalità le fa assomigliare sempre più ai pacchetti integrati della prima metà degli anni '80.
- Si hanno notevoli sovrapposizioni e duplicazioni di funzioni fra word processors, fogli elettronici, database ecc.
- Nasce quindi la necessità di scomporre questi applicativi in moduli che possano essere condivisi
- E' necessario procedere ad una "componentizzazione" delle applicazioni e ancora una volta deve essere il sistema operativo a fornire gli strumenti
- In tal modo un utente potrà scegliere di installare solo le funzionalità che gli sono necessarie attingendo ad un mercato di componenti sostituibili fra di loro

Un abbozzo di soluzione..

- Una tecnologia basata su oggetti e costruita su un modello di interazione client-server di tipo sincrono rappresenta una soluzione ideale per questo tipo di problemi. Infatti:
 - **Robustezza:** un meccanismo di interazione sincrono permette di costruire una forma di comunicazione intrinsecamente robusta
 - **Omogeneità dell'API:** un'impostazione object-based prevede la suddivisione dei servizi forniti dal sistema operativo in interfacce - (interfaccia=insieme dei metodi di un oggetto) e quindi fornisce una classificazione ordinata e coerente dei servizi stessi (cfr. Java)
 - **Indipendenza dalla collocazione:** un'interazione di tipo client/server si basa solo su un protocollo di comunicazione e su un meccanismo di indirizzamento: è quindi totalmente indipendente dalla collocazione fisica dei soggetti e dalla loro implementazione
 - **Componentizzazione:** il principio di encapsulamento è il cardine di qualunque tecnologia di componenti software

COM



- COM = Component Object Model (1992)
- È un modello binario di interazione fra processi, basato su componenti e su un meccanismo di comunicazione client/server di tipo sincrono.
- COM è un modello, cioè un insieme di specifiche, supportato da alcuni servizi di sistema (supporto runtime)
- Le specifiche definiscono uno standard binario per la creazione di componenti in grado di interagire fra di loro
- Trattandosi di uno standard binario c'è completa indipendenza dal linguaggio di programmazione usato per realizzare i componenti e le applicazioni che li utilizzano
- Il modello prevede sia un funzionamento locale che distribuito (DCOM)
- Il supporto runtime è costituito da una DLL di sistema che fornisce i servizi necessari per l'accesso ai componenti

Indirizzamento e GUID

- COM deve prevedere un meccanismo di indirizzamento per reperire i componenti
- L'indipendenza dalla collocazione fisica non consente di utilizzare un indirizzo fisico (pathname)
- Si utilizzano invece degli identificatori globali (GUID=globally unique identifiers)
- Il concetto di GUID è stato introdotto, con un nome leggermente diverso (UUID=universally unique id.), dall'OSF (Open Software foundation) nelle specifiche DCE (Distributed computing environment).
- In DCE gli UUID vengono utilizzati per identificare i destinatari delle chiamate di procedura remota (RPC)
- Un GUID è un numero di 128 bit (16 byte) assegnato in modo da garantire l'unicità nello spazio (48 bit) e nel tempo (60 bit).
- Viene rappresentato così: {32bb8320-b41b-11cf-a6bb-0080c7b2d682}
- COM utilizza diversi tipi di GUID.

CLSID

- Il primo utilizzo dei GUID è nell'identificazione delle classi di componenti: ogni classe di componenti COM è caratterizzata da un proprio identificatore che viene chiamato CLSID (Class Identifier). Disponendo di un CLSID si può chiedere a COMPOBJ di creare un'istanza e restituire un riferimento.
- Il database di sistema di Windows - la registry - mantiene una corrispondenza fra CLSID e le entità fisiche (EXE, DLL) che contengono l'implementazione dei componenti (server).
- La funzione CoCreateInstance (contenuta in COMPOBJ) provvede a:
 - reperire il server tramite la registry
 - caricarlo in memoria (se non è già presente)
 - chiamare una funzione che crea un'istanza e restituisce un riferimento
- La registry fornisce anche un servizio di naming, ovvero una corrispondenza fra nomi di classi (ProgID) e CLSID.
- I ProgID hanno un formato
`<nome>.<componente>.<versione>`
 - P.es. Word.WordBasic.5

Comunicare con i componenti: le interfacce

- Per definizione un oggetto COM è un oggetto identificato univocamente da un GUID in grado di esporre una o più interfacce.
- Un'interfaccia è insieme di funzioni (metodi) che permettono di interagire con l'oggetto che la espone. In virtù dell'incapsulamento, le interfacce sono l'unico modo per interagire con l'oggetto.
- COM è stato pensato in C++ e il meccanismo delle interfacce nasce dalla tecnica abitualmente usata per accedere ad oggetti definiti e istanziati nelle DLL.
- Un'interfaccia non è altro che un puntatore ad una porzione della "virtual method table" (*vtable* o *VMT*) di un oggetto. La VMT è in sostanza una tabella di puntatori a funzione.
- Usando una terminologia "Java" un oggetto COM è un oggetto che implementa un certo numero di interfacce.
- Anche le interfacce sono identificate da GUID. I GUID che svolgono questo ruolo vengono chiamati IID (Interface Identifiers).
- Per convenzione i nomi delle interfacce iniziano con la lettera I.

IUnknown: il punto di partenza

- Un Oggetto COM deve esporre almeno una interfaccia: IUnknown
- IUnknown rappresenta una sorta di punto di ingresso in quanto consente di accedere a tutte le altre interfacce esposte dall'oggetto. In pratica un oggetto COM si identifica con la sua IUnknown
- IUnknown comprende 3 metodi: QueryInterface, AddRef e Release
- AddRef e Release implementano un meccanismo di “reference counting”
- QueryInterface permette di accedere alle altre interfacce esposte dall'oggetto: possiamo chiedere all'oggetto se implementa una determinata interfaccia passando come parametro il relativo IID. In caso positivo il parametro Obj contiene un riferimento all'interfaccia richiesta
- La sintassi è:

```
function QueryInterface(const IID: TGUID; out Obj): HRESULT;
```
- HRESULT è un intero che ci dice se l'interfaccia richiesta è disponibile (S_OK) oppure no (E_NOINTERFACE)
- Anche IUnknown è identificata da un IID:
`'{00000000-0000-0000-C000-000000000046}'`

IUnknown: la madre di tutte le interfacce



- COM supporta l'ereditarietà delle interfacce, quindi un meccanismo per il polimorfismo ma non per il riuso
- Tutte le interfacce COM discendono da IUnknown
- Questo significa che tutte le VMT corrispondenti hanno nei primi 3 slot i metodi QueryInterface, AddRef e Release
- La presenza di QueryInterface consente di passare da un'interfaccia all'altra senza dover ritornare sempre indietro ad IUnknown
- L'ereditarietà delle interfacce è usata molto limitatamente: in pratica tutte le interfacce discendono da IUnknown o da IDispatch (che discende a sua volta da IUnknown)

Reference counting

- Come abbiamo detto gli oggetti COM implementano un meccanismo di reference counting per le interfacce
- In pratica ogni interfaccia ha un suo contatore
- Quando QueryInterface restituisce un'interfaccia incrementa anche il contatore
- Ogni chiamata ad AddRef incrementa a sua volta il contatore
- Ogni chiamata a Release lo decrementa e quando il conteggio scende a zero l'interfaccia può essere liberata
- Quando i contatori di tutte le interfacce implementate da un oggetto sono a zero l'oggetto può essere distrutto
- E' una sorta di "garbage collection manuale"
- In pratica quando l'applicazione che utilizza l'oggetto (il client) assegna il riferimento ad una nuova variabile deve provvedere a chiamare AddRef. Inoltre deve chiamare Release ogni volta che alla variabile viene assegnato un nuovo riferimento oppure quando la variabile stessa esce dallo scope

Schema di utilizzo di un oggetto COM

- Vediamo in pratica come funzionano le cose per un'applicazione che usa un oggetto COM:
 - Inizializza il sistema chiamando CoInitialize
 - Chiama la funzione CoCreateInstance, esportata da COMPOBJ.DLL, passando come parametro il CLSID dell'oggetto che ci interessa
 - function **CoCreateInstance**(const **clsid**: **TCLSID**; **unkOuter**: **IUnknown**;
 dwClContext: **Longint**; const **iid**: **TIID**; out **pv**): **HRESULT**;
 - CoCreateInstance procede così:
 - ✧ usa la registry per risalire al server che implementa la classe richiesta
 - ✧ se la classe è registrata attiva il server (se non è già attivo)
 - ✧ chiede al server di creare un'istanza
 - ✧ riceve dal server un riferimento all'interfaccia IUnknown dell'istanza
 - ✧ restituisce IUnknown all'applicazione
 - L'applicazione usa IUnknown.QueryInterface per accedere all'interfaccia voluta
 - Può invocare tutti i metodi disponibili e accedere ad altre interfacce
 - Usa AddRef e Release per gestire il tempo di vita dell'oggetto
 - Alla fine di tutto chiama CoUninitialize

Creazione degli oggetti: Class Factory

- Consideriamo il caso di un server implementato in una DLL:
 - La DLL esporta due funzioni per registrare e deregistrare il server nella registry.
 - Si usa un utility di sistema (regsvr32) per eseguire la registrazione
 - La DLL esporta una funzione per la creazione degli oggetti:
`function DllGetClassObject(const CLSID,IID:GUID;var Obj):HRESULT;`
 - Il meccanismo di creazione è indiretto: la funzione non crea un'istanza della classe richiesta
 - Crea invece un'istanza di una “Class Factory” e ne restituisce l'interfaccia IClassFactory
 - Chiamando il metodo IClassFactory.CreateInstance si ottiene finalmente la creazione dell'istanza e la restituzione di IUnknown
 - Il meccanismo delle Class Factory consente di encapsulare le problematiche relative alla creazione di un oggetto e di mantenere un elevato grado di isolamento fra client e server
 - CoCreateInstance chiama prima la funzione CoGetClassObject per ottenere la class factory e quindi invoca IClassFactory.CreateInstance

Ereditarietà e aggregazione

- Come abbiamo detto, COM supporta l'ereditarietà delle interfacce ma non quella delle implementazioni.
- Nella visione dei progettisti di COM l'ereditarietà provoca un'accoppiamento troppo stretto fra gli oggetti
- In particolare l'ereditarietà è vista come una pericolosa breccia nell'incapsulamento: una classe derivata può accedere allo stato interno della classe base
- Questo può provocare interazioni non desiderate e creare problemi di compatibilità fra versioni
- In alternativa COM propone un meccanismo di riuso chiamato aggregazione (una forma di delega): un oggetto espone anche le interfacce di un altro oggetto (aggregato)
- E' molto meno elegante e più complessa dell'ereditarietà (poco più di una specifica di implementazione)

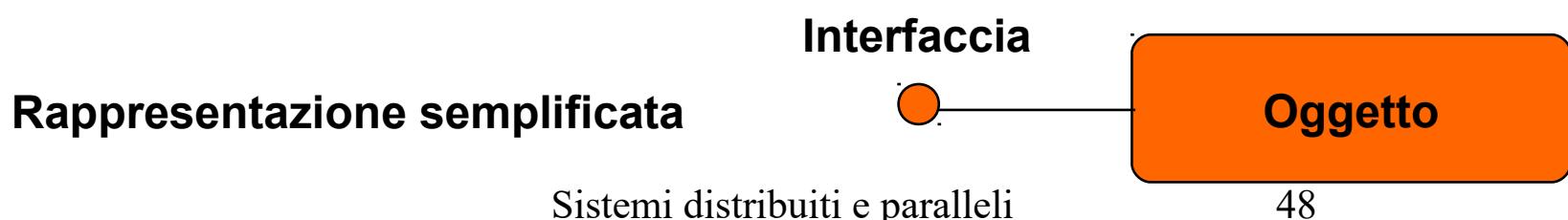
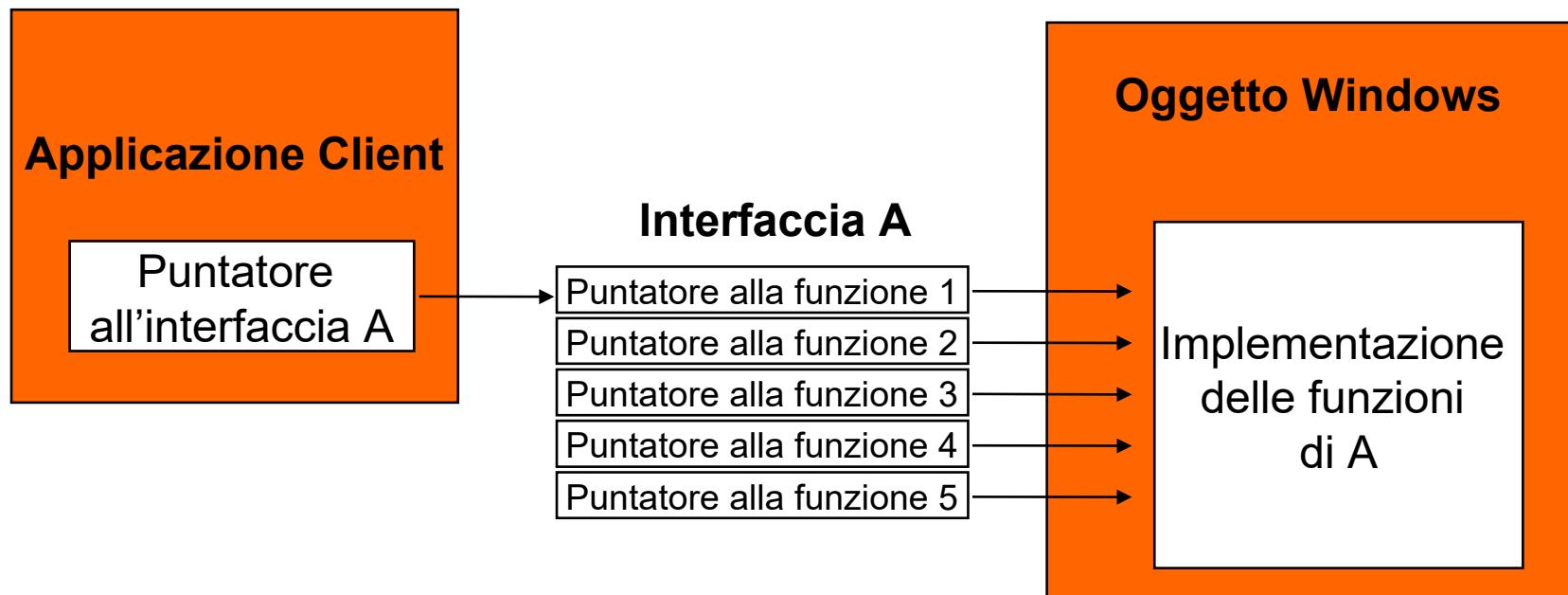
Oggetti COM e server

- ❑ Indipendenza dalla collocazione fisica: l'applicazione che usa un oggetto COM (client) non sa dove risiede effettivamente l'oggetto
- ❑ Server: entità in cui risiede un oggetto COM
- ❑ Tre tipi di server
 - In-process server: all'interno dello stesso spazio del processo utente, in una DLL
 - Local server: fuori dal processo ma nella stessa macchina (in un EXE)
 - Remote server: in un'altra macchina: DCOM (Distributed COM)
- ❑ I server locali e remoti vengono detti "out-of-process".

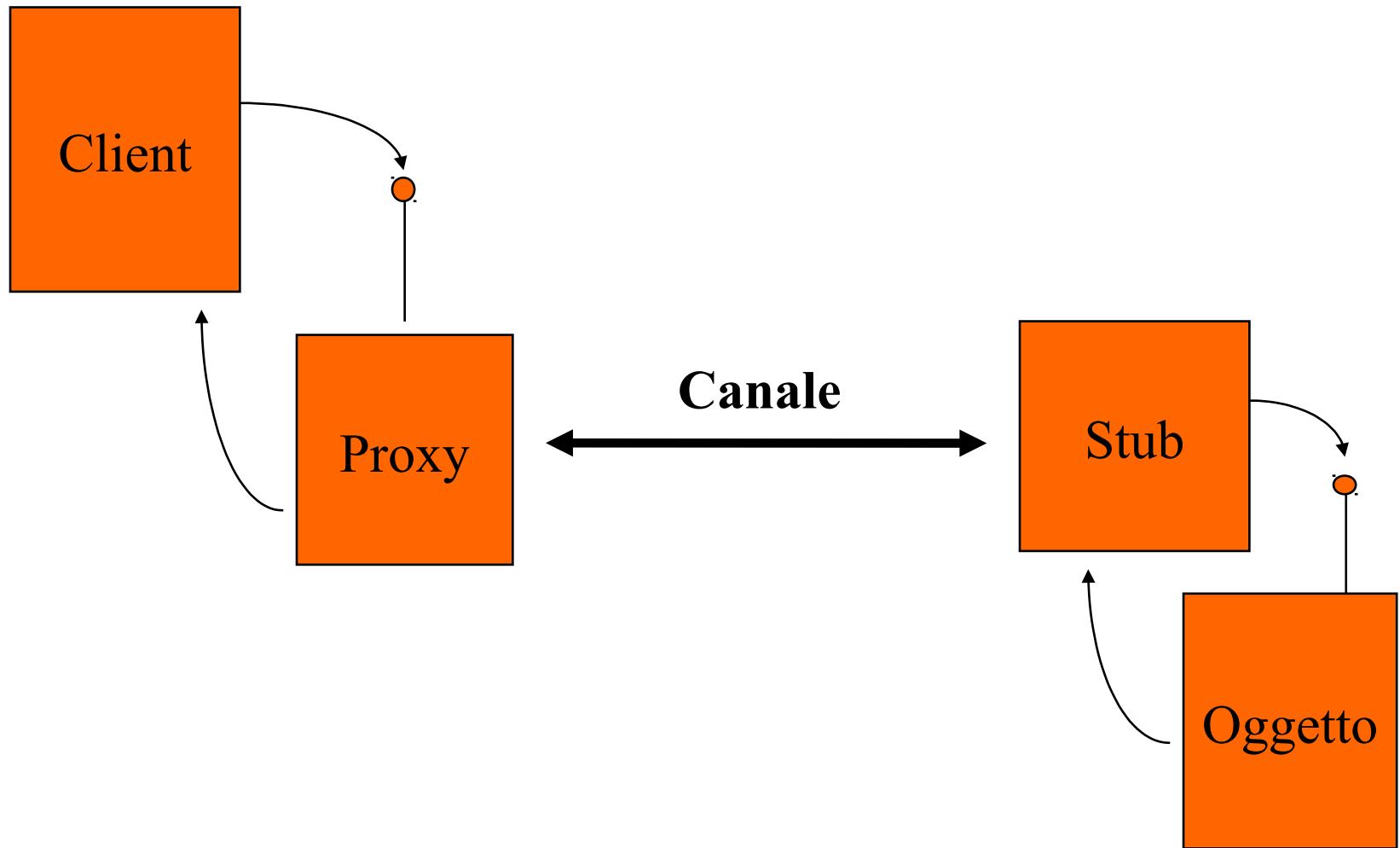
Marshalling

- Marshalling: meccanismo per il passaggio delle chiamate e dei parametri fra il client e il server
- Il marshalling trasforma una chiamata di funzione in un pacchetto di dati (PDU: protocol data unit), contiene un ID della funzione e i parametri,
- Questo pacchetto può essere trasmesso con un protocollo di rete
- Il ricevente utilizza un meccanismo simmetrico (unmarshalling) per trasformare il PDU in una chiamata effettiva di funzione.
- L'applicazione chiama un'immagine locale (proxy) dei metodi di un'interfaccia e il marshalling provvede a trasmettere chiamate e parametri all'oggetto effettivo
- In-process server: l'immagine locale coincide con l'oggetto e quindi non c'è marshalling
- Local server: forma semplificata di RPC, chiamata LRPC (Lightweight Remote Procedure Call) o LPC(Local Procedure Call) , basata sui messaggi Windows.
- Remote server: RPC standard

Schema del modello COM



Schema di implementazione di COM



In-process server



- Server in-process: DLL che esporta 4 funzioni standard: DllGetClassObject, DllCanUnloadNow, DllRegisterServer, DllUnregisterServer
- L'applicazione client richiede un oggetto al supporto runtime (COMOBJ) sulla base di un GUID
- COMOBJ cerca nella registry il nome della DLL associata al GUID e la carica in memoria
- Chiama una funzione della DLL e ottiene l'interfaccia IClassFactory (vtable di un oggetto ClassFactory)
- Chiede a ClassFactory di instanziare l'oggetto richiesto e di restituirne l'interfaccia IUnknown (IFClassFactory.CreateInstance)
- L'interfaccia viene passata al client
- Il client invoca i metodi dell'oggetto nella DLL

Local server: attivazione



- Il client richiede a COMPOBJ un oggetto con un GUID
- COMPOBJ ricava il nome dell'applicazione (EXE) in cui risiede l'oggetto e se necessario la avvia
- L'applicazione chiama una funzione di COMPOBJ a cui passa un puntatore alla propria interfaccia IUnknown
- COMPOBJ attiva il proxy (è un in-process server) e richiede a quest'ultimo l'interfaccia IUnknown.
- Restituisce al client l'interfaccia IUnknown del proxy
- Attiva lo stub (DLL che risiede nello spazio di memoria del server) e gli passa il puntatore all'interfaccia IUnknown del server

Local server: chiamata



- L'applicazione client chiama un metodo dell'interfaccia: in realtà è un metodo del proxy
- Il proxy esegue il “marshalling”: trasforma la chiamata e i parametri in un messaggio Windows e lo invia allo stub
- Il messaggio arriva allo stub che estrae l'identificatore della chiamata e i parametri (“demarshalling”)
- Lo stub chiama il metodo opportuno dell'interfaccia effettiva (è in pratica un callback)
- Il server esegue il metodo

Local server: ritorno

- ❑ Il metodo termina e restituisce i risultati (valore di ritorno della funzione + parametri passati per riferimento) allo stub
- ❑ Lo stub trasforma i risultati in un messaggio Windows (marshalling) e lo invia al proxy
- ❑ Il proxy riceve il messaggio ed estrae i risultati
- ❑ Il metodo del proxy termina e restituisce i risultati al client

Remote server

- Il meccanismo è lo stesso del local server
- Il passaggio dei messaggi avviene però su una rete attraverso un meccanismo RPC standard (DCE)
- Il marshalling consiste nella creazione di un PDU (Protocol Data Unit) ovvero un pacchetto di dati da trasmettere sulla rete, per esempio come pacchetto TCP/IP
- Nel proxy e nello stub abbiamo cicli di attesa che rendono sincrona l'operazione
- L'aspetto più complesso è garantire che, pur in presenza di un meccanismo sincrono, le applicazioni non si blocchino (deadlock)

Generalizzazione del marshalling

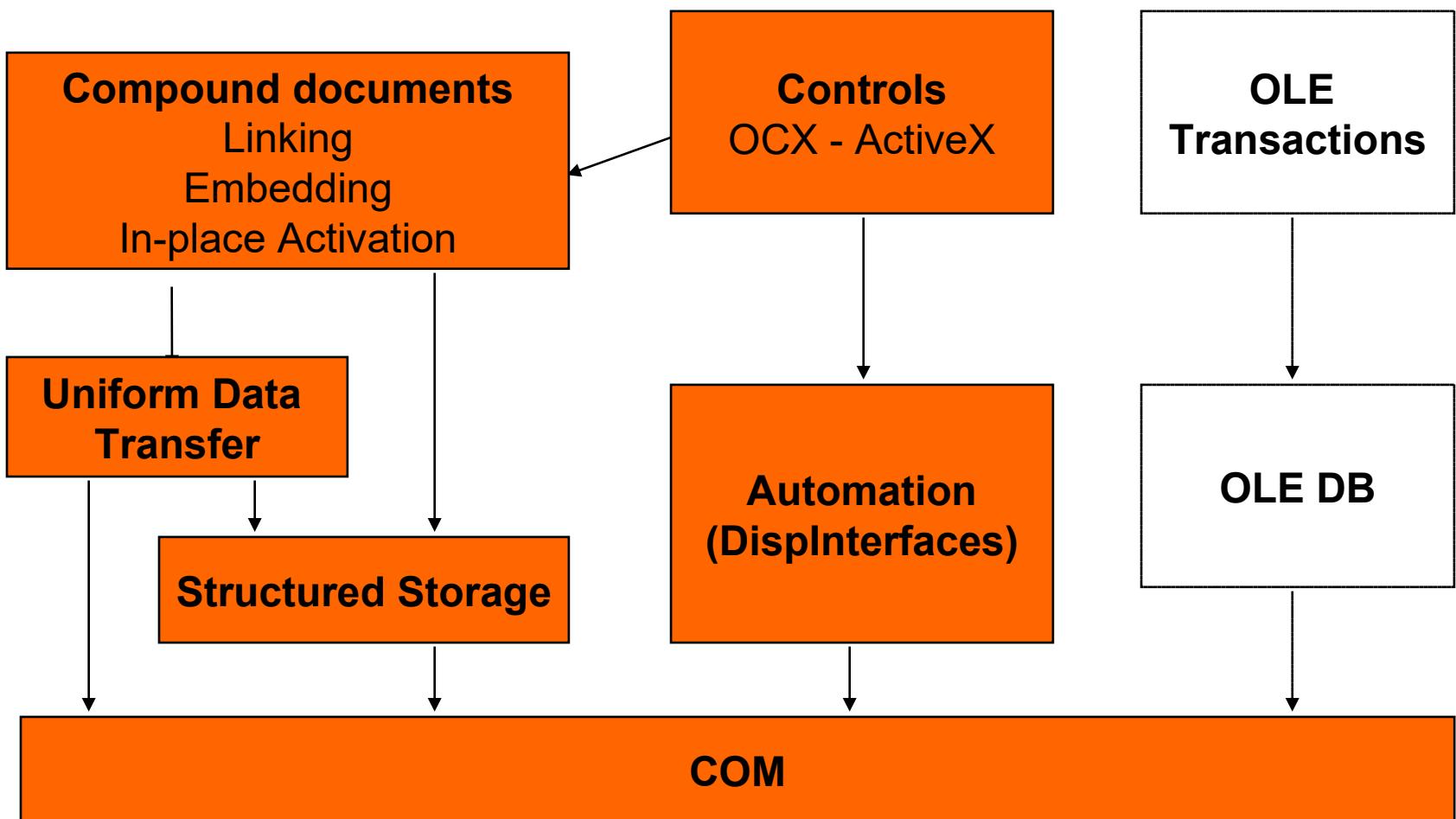
- Ogni interfaccia o insieme di interfacce richiede una coppia proxy/stub in grado di eseguire marshalling/demarshalling
- In una prima fase COM metteva a disposizione questo servizio solo per le interfacce standard
- La Microsoft sconsigliava la definizione di interfacce non standard
- L'uso di un'interfaccia non standard richiedeva la scrittura manuale di un proxy e di uno stub dedicati
- Recentemente è stato introdotto un meccanismo generalizzato

MIDL



- Le interfacce vengono descritte mediante un apposito linguaggio (MIDL)
- MIDL = Microsoft Interface Definition Language
- Il compilatore MIDL genera automaticamente la coppia proxy/stub a partire da una di queste descrizioni
- MIDL è un linguaggio con una sintassi simile a quella del C ed è compatibile con lo standard DCE: IDL (Interface Definition Language)
- Incorpora un meccanismo precedente (ODL=Object Definition Language) che forniva una soluzione parziale legata ad OLE Automation

Le tecnologie ActiveX



Documenti composti (OLE)

- ❑ Integrazione di componenti provenienti da fonti diverse all'interno di un documento composto
- ❑ Container: applicazione che gestisce il documento
- ❑ Server: applicazioni specializzate che gestiscono i componenti
- ❑ Embedding: il componente viene incorporato nel documento
- ❑ Linking: il documento contiene solo un riferimento al componente
- ❑ In-place activation: l'applicazione server si attiva nello spazio visivo (finestra) del container

Structured storage



- ❑ Consente di creare un intero file system all'interno di un singolo file
- ❑ Gestione di concorrenza e diritti di accesso
- ❑ Gestione delle transazioni
- ❑ Nato come supporto per il salvataggio dei documenti composti
- ❑ Base per il futuro file system ad oggetti (OFS) di Cairo

Uniform data transfer



- Unifica i vari sistemi di scambio dati fra processi: clipboard, drag and drop, DDE
- Indipendenza dal mezzo fisico usato per lo scambio (memoria, file, protocolli di rete ...)
- Separazione netta fra impostazione del trasferimento (protocollo) ed effettivo scambio dati
- Lo scambio avviene mediante un particolare oggetto COM denominato Data Object
- Il protocollo è costituito da una serie di funzioni API che hanno il compito di scambiare un Data Object fra due processi

Automation

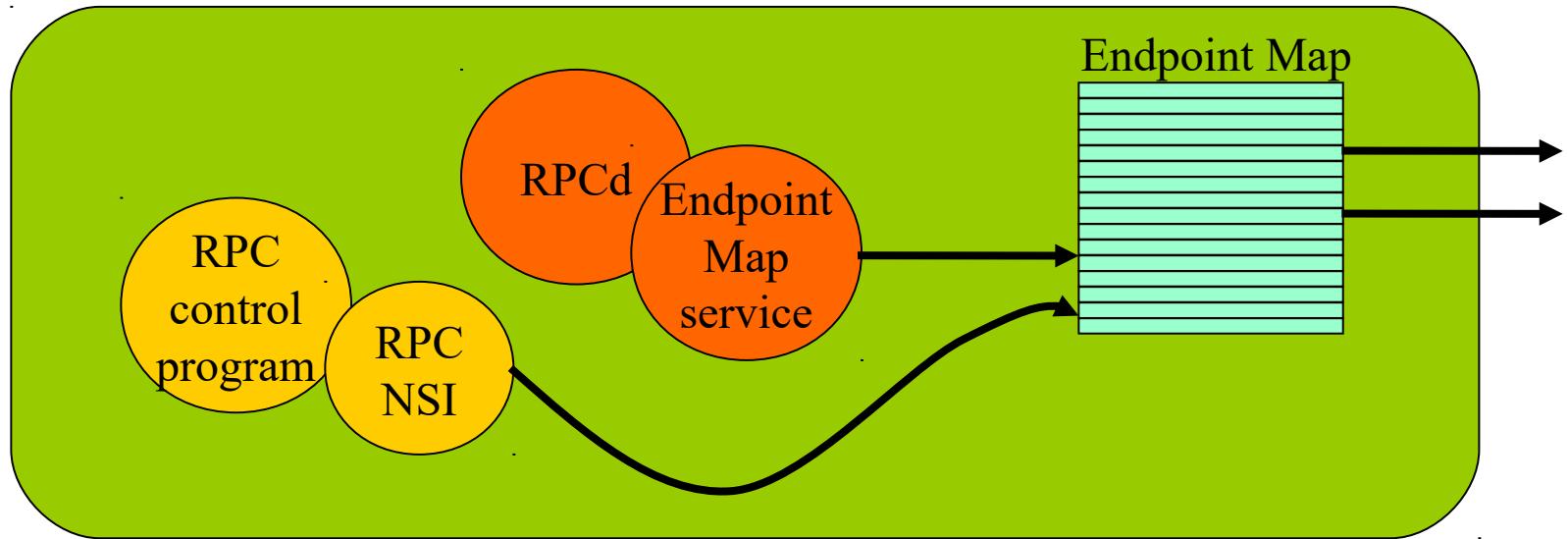
- Consente ad un'applicazione (controller o client) di comandare dall'esterno un'altra applicazione (server)
- E' in pratica un metodo di scripting generalizzato indipendente dal linguaggio.
- E' la tecnologia più usata con DCOM (Remote automation)
- Consente di incapsulare e riutilizzare applicazioni legacy
- Fornisce un metodo semplificato per creare interfacce non standard senza ricorrere a proxy/stub

Introduzione a DCE

- ❑ **Distributed Computing Environment**
- ❑ **è un middleware per sistemi distribuiti eterogenei:**
 - per programmazione (procedurale)
 - per gestione sistemistica
- ❑ **supportato dai maggiori fornitori (Compaq, Fujitsu, HP, Hitachi, IBM, HP, ...), software side (Tivoli), e user-side (VISA)**
- ❑ **Open Software Foundation**
 - Fornitori ed utenti di Sistemi Informativi
 - Collabora con IETF, W3C, TMF, OMG

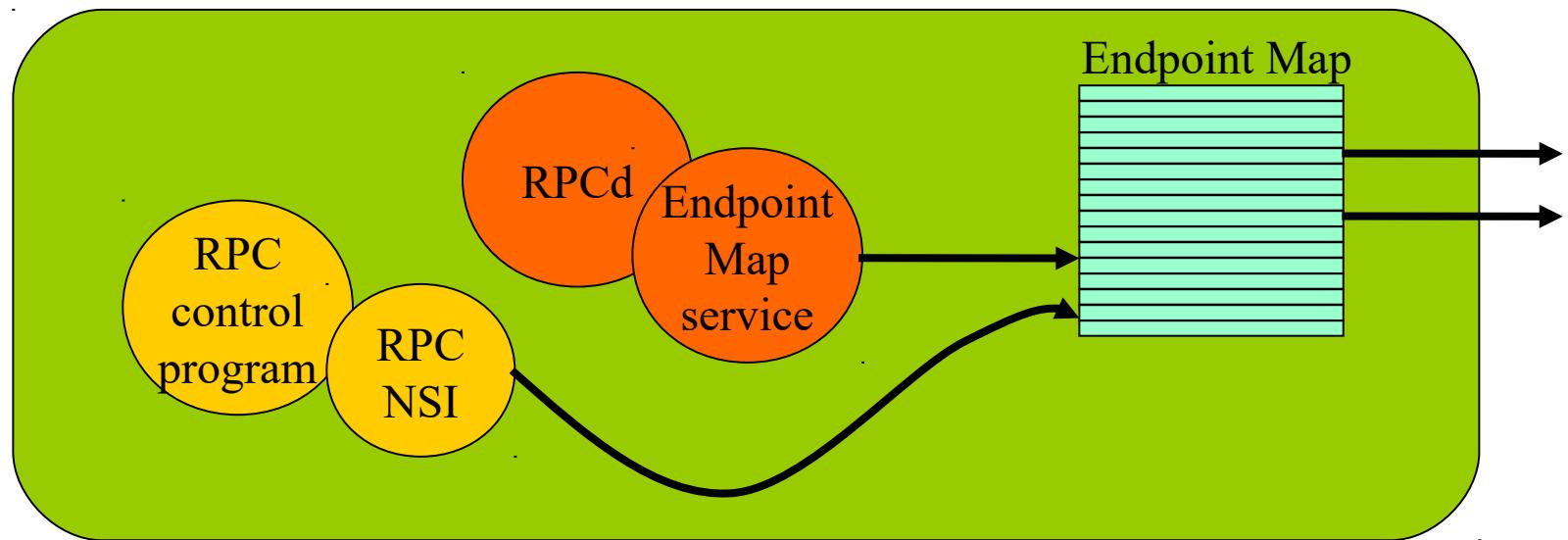
THE *Open* GROUP

DCE RPC



- ❑ **RPCd - RPC daemon**
- ❑ **Endpoint Map Service**
 - mantiene la Endpoint Map per i server RPC locali
 - fa il lookup degli endpoint per i client RPC

DCE RPC (cont.)



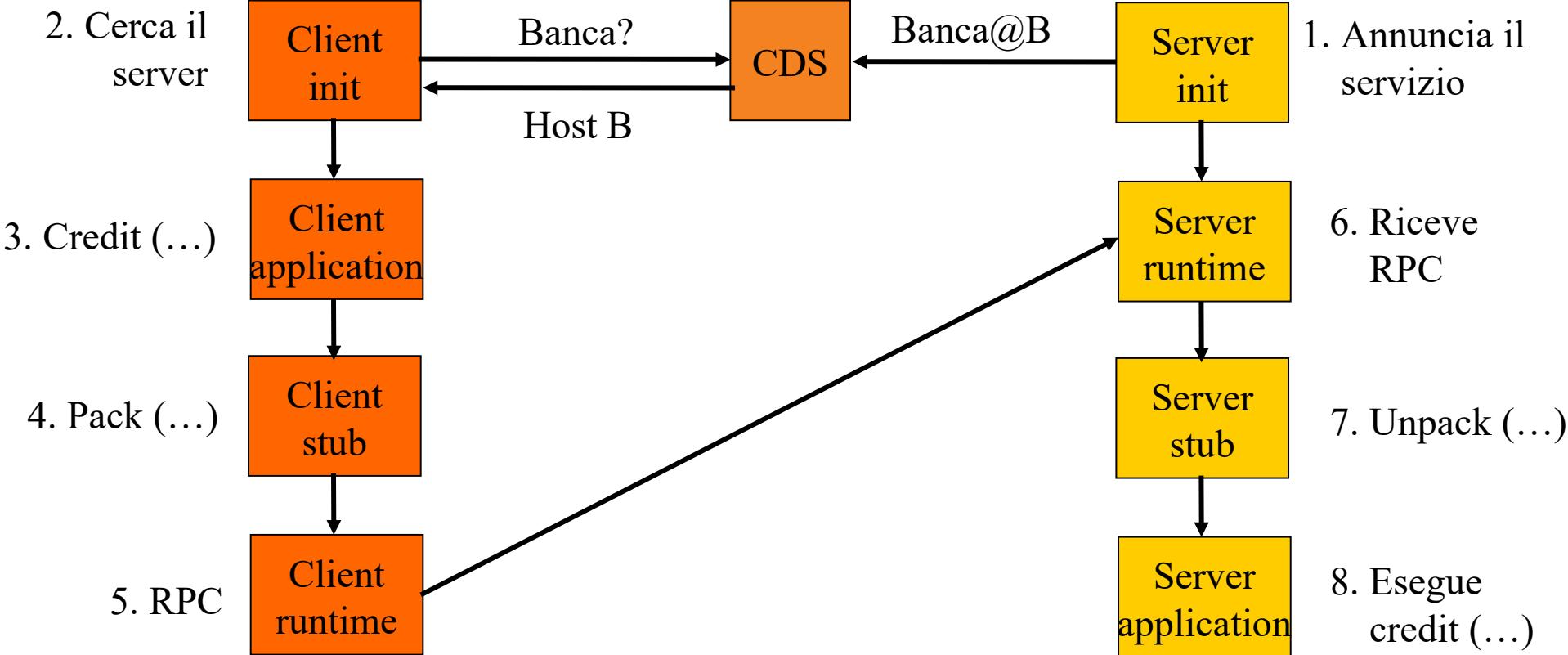
□ **RPC cp**

- comandi per accedere a RPC name service interface (NSI)

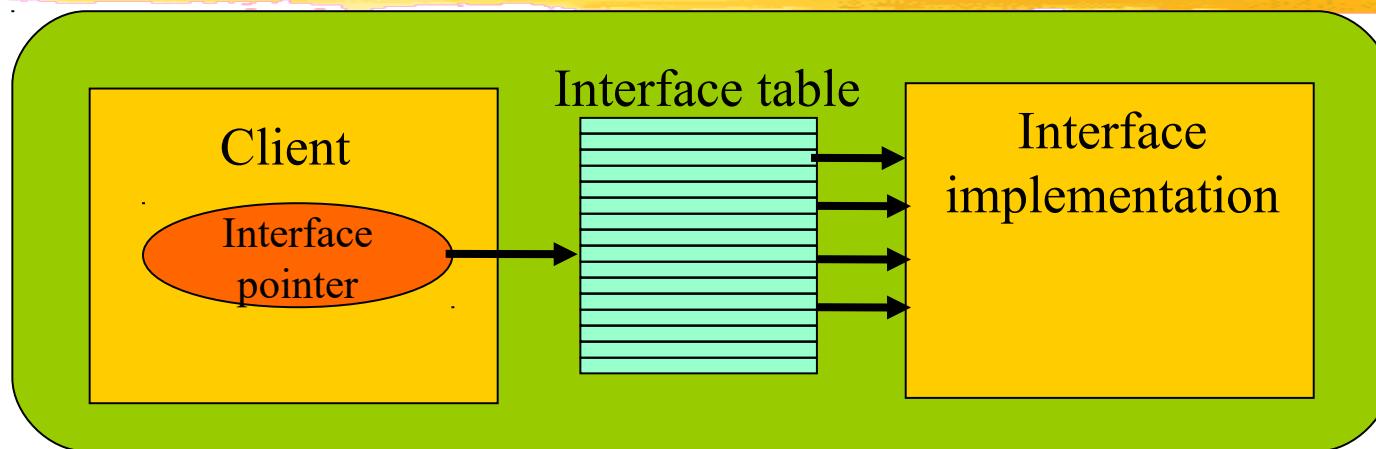
□ **RPC NSI**

- offre operazioni di show e delete delle endpoint locali e remote

Funzionamento di RPC



Component Object Model

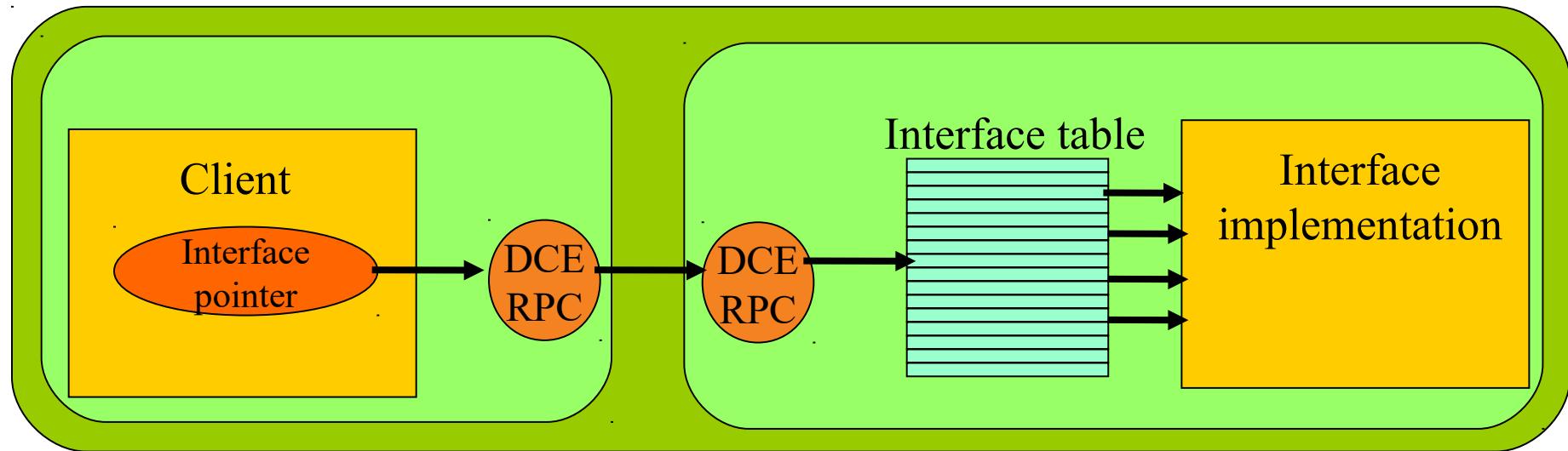


- **Ottenimento del puntatore a un'interfaccia**
 - COM: libreria delle funzioni di base con prefisso “Co” (Es. CoCreateInstance)
 - DCOM: dare l'indicazione del sito in cui trovare l'eseguibile del server
- **Creazione di istanze multiple: class Factory**
 - il client invoca IClassFactory::CreateInstance
 - la class factory crea l'oggetto e restituisce il puntatore all'interfaccia (IID)

Oggetti COM

- **Ogni oggetto supporta l'interfaccia IUnknown che offre**
- **QueryInterface()** - restituisce le interfacce offerte da un oggetto
- **AddRef()**- mantiene il numero dei processi (contatore) che stanno usando l'oggetto (chiamato ad ogni richiesta)
- **Release()** - decrementa il contatore quando un client rilascia l'interfaccia. Quando == 0 viene chiamato il distruttore della classe che implementa l'oggetto
- **Usa aggregazione piuttosto che ereditarietà**
- **Definisce le interfacce in IDL (Microsoft!)**

DCOM



- Usato principalmente in ambiente Windows
- è un'estensione di COM con l'uso di DCE RPC
- ... OMG ha definito l'interoperabilità tra DCOM e CORBA
(es. Iona OrbixCOMet)

Java RMI



- **Remote Method Invocation**
- **Java java.rmi**
 - implementazione del *server* (**myInterface.java** e **MyServer.java**)

- - ◊ dichiarare un'interfaccia remota (simile a IDL)
 - ◊ implementarne le operazioni
 - ◊ registrare la presenza di ogni oggetto presso il *runtime RMI*
 - ◊ registrare il *Security Manager*
 - ◊ registrare ogni oggetto nel Name Server di Java, il *Registry*
 - ◊ generare *stub* e *skeleton* (compilatore rmic), resp.
MyServer_Stub.class e *MyServer_Skel.class*

```
public interface myInterface extends java.rmi.Remote
{
    public void myMethod () throws java.rmi.RemoteException;
}
```

```
import java.rmi.server.*;
import java.rmi.*;
public class MyServer implements myInterface {
    public MyServer () throws java.rmi.RemoteException{ super();
    }

    public void myMethod () throws java.rmi.RemoteException {
        System.out.println("WOW!! SUCCESS!!");
    }
}
```

extends
java.rmi.server.UnicastRemoteObject

```
public void main(String args[]) {
    System.setSecurityManager(new RMISecurityManager());
    try {
        MyServer obj = new MyServer();
        Naming.rebind("///MyInterfaceServer", obj);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Java RMI (cont.)

□ Implementazione del *client* (**MyClient.java**)

- usare il *Registry* per recuperare il riferimento al *server* (specificando l'host su cui esso si trova)
- invocare l'operazione remota

□ Compilazione

- **rmic MyServer**
- **javac MyClient.java**

□ Esecuzione

- **start rmiregistry // start the registry**
- **start java MyServer // server start**
- **java MyClient // client start**

Esempio (MyClient.java)

```
import java.rmi.*;
import java.rmi.server.*;
import java.io.*;
public class MyClient {
    public static void main(String args[]) {
        try {myInterface obj= (myInterface)
        Naming.lookup("//corolla.polito.it/MyInterfaceServer");
        obj.myMethod();
    }
    catch (Exception e) {e.printStackTrace();}
}
}
```

Java RMI e CORBA

□ RMI

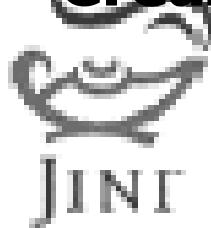
- ambiente omogeneo e Java puro
- Java Object Model
- il client deve conoscere la locazione fisica del server RMI
- il server RMI deve essere in esecuzione e registrato presso Registry e Security Manager

□ CORBA

- ambiente eterogeneo e multi-linguaggio
- IDL
- il client non si occupa della locazione fisica dell'oggetto server
- l'oggetto server può anche non essere in esecuzione al momento della richiesta, ma deve essere registrato almeno presso l'Interface Repository

Cos'è Jini?

Creazione spontanea di sotto-reti di servizi ed utenti



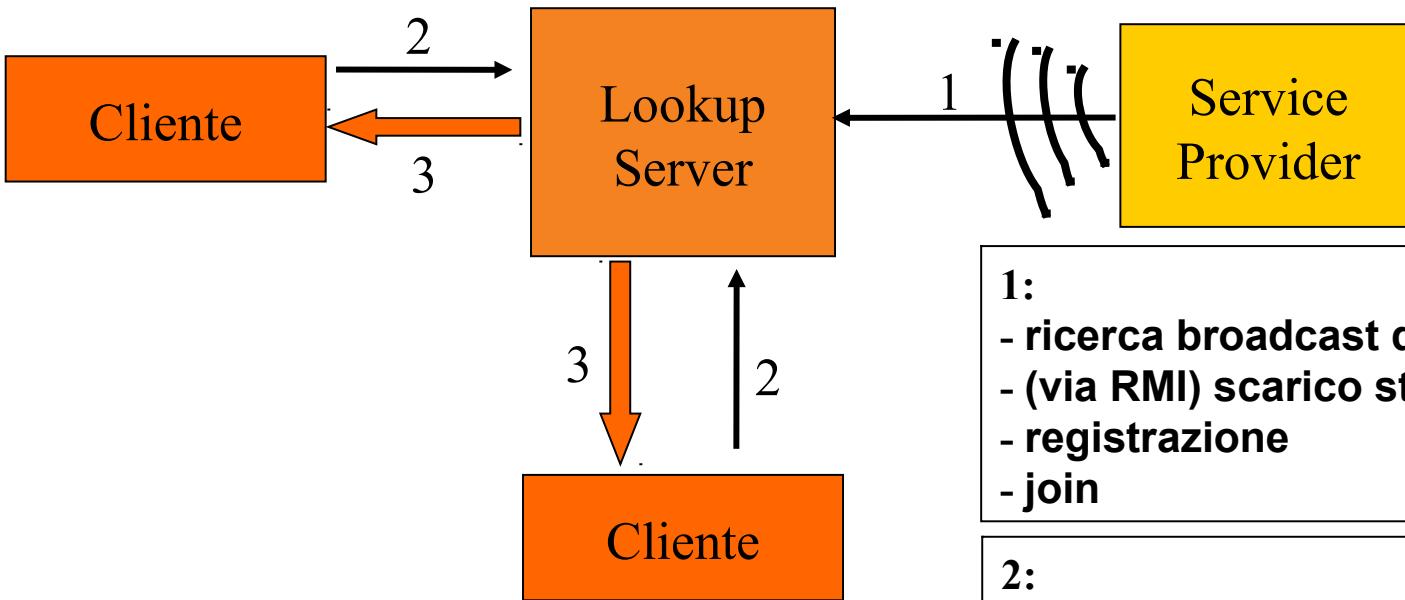
- ❑ **Costruito su Java e RMI**
- ❑ **35,000 LOC (~10K)**
- ❑ **Più leggero di CORBA**
 - Event broadcast
- ❑ **Java 1.3**
 - RMI broadcast
 - Security



Funzionamento (I)

- Architettura basata sul Lookup Server
 - Modello “Cliente cerca Servizi”
-
- Fase 1: Join
 - ✧ broadcast del *Discovery package*
 - Fase 2: Discovery
 - ✧ broadcast del *Request package*

Funzionamento (II)



1:
- ricerca broadcast del lookup server
- (via RMI) scarico stub dei servizi
- registrazione
- join

2:
- ricerca broadcast del lookup server
- e dei servizi di un certo tipo
(operazione e attributi)

3:
- un lookup server risponde
scaricandogli il service stub
- il cliente può interagire direttamente
con il Provider

Servizi in Jini:

○ oggetto (interfaccia)

◊ esecuzione locale consumer

○ riferimento (stub)

◊ esecuzione remota provider

Lookup Communication Protocol



□ Tre protocolli di Discovery

○ Multicast request protocol

○ Multicast announcement protocol

- ◆ Indirizzo multicast 224.0.1.84/85, Port 4160
- ◆ Uso in sottoreti (e nel dominio)

○ Unicast discovery protocol

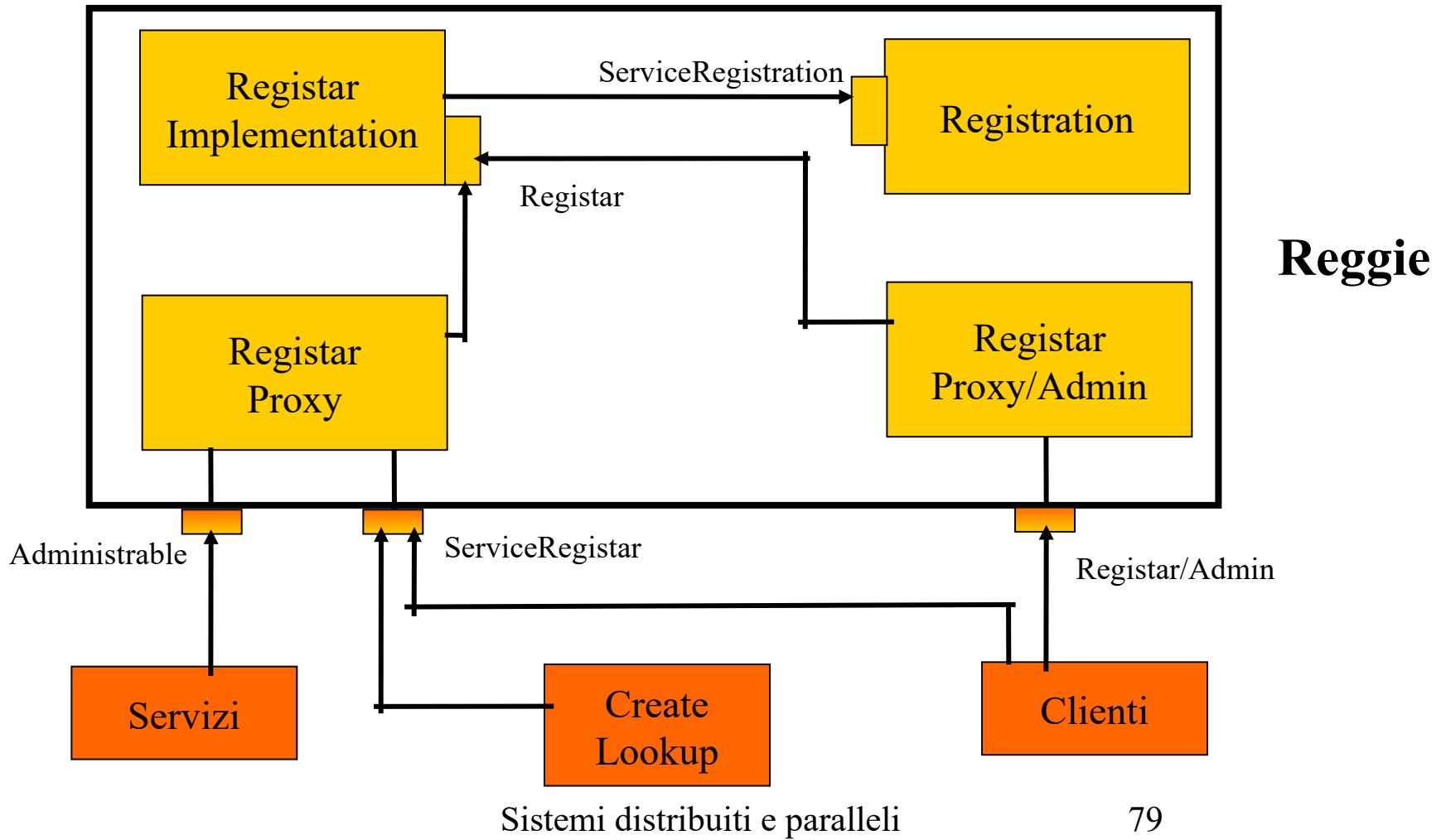
- ◆ Serve l'indirizzo IP del Lookup Server
- ◆ Uso in ambienti non-locali e distribuiti

Service Matching



- Si usano ***Entries*** e ***Templates***
- Un Service Item (i) fa matching con un Template (t) se:
 - $i.\text{ServiceID} = t.\text{ServiceID}$ (o $t.\text{ServiceID}$ è null) &
 - $i.\text{service}$ è una istanza di $t.\text{serviceTypes}$ &
 - $i.\text{attributeSet}$ ha almeno una *matching entry* per $t.\text{attributeSetTemplate}$

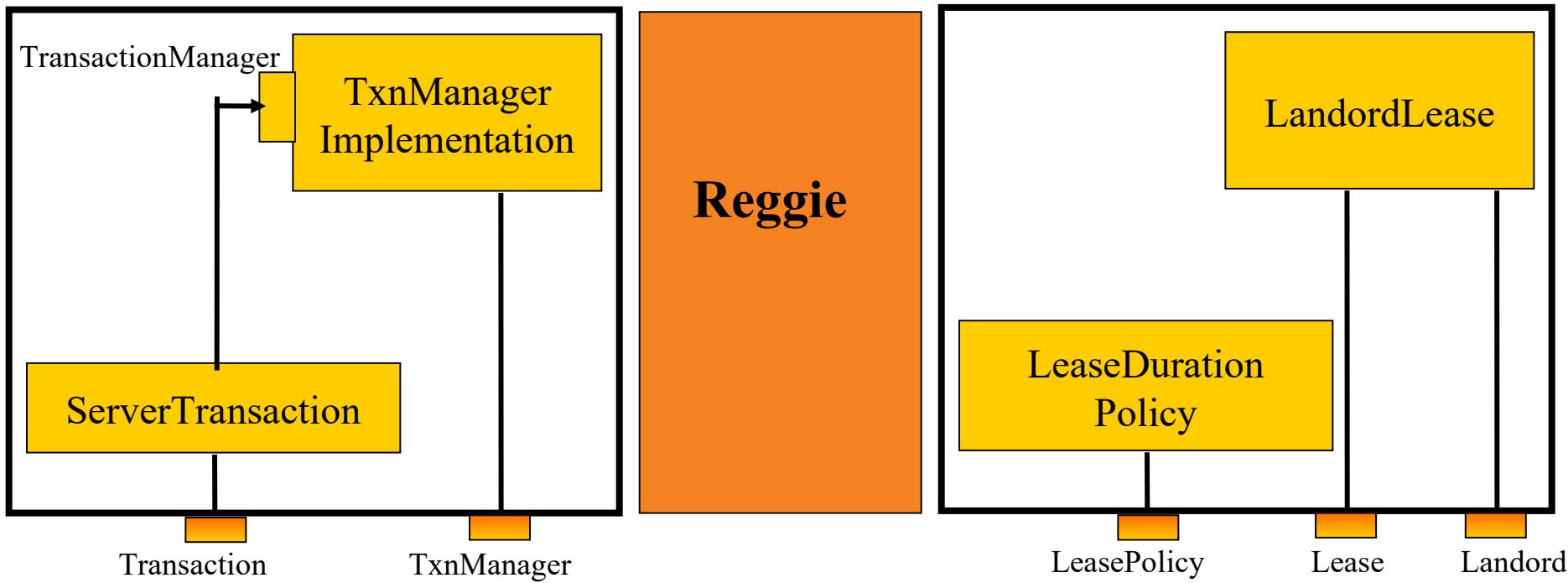
Architettura del Lookup Server



Architettura di Jini

Mahalo

Lease



□ Risorse Plug&Play

- Kodak, Sony, Ticino, ...
- Architetture software di servizi?

Introduzione a XML



- XML è un nuovo linguaggio creato a partire dallo standard SGML al fine di garantire maggior flessibilità e generalità di utilizzo per implementare diversi tipi di dati.
- È stato sviluppato dallo XML Working Group (originariamente noto come SGML Editorial Review Board) costituitosi all'interno del W3C nel 1996.
- XML risponde alla necessità di mettere in comunicazione sistemi eterogenei.
- Attualmente esiste una notevole quantità di formati per rappresentare informazioni testuali o strutturate, proprietari o meno, che sono difficilmente gestibili all'infuori dell'applicativo originario (ex Microsoft Word, Adobe PDF).

Obiettivi di XML



- Compatibilità con applicazioni differenti.
- Creare un linguaggio utilizzabile con facilità in Internet, ma più flessibile di HTML.
- Facilità di progettazione ed elaborazione dei documenti.
- Necessità di disporre di un linguaggio che permettesse di descrivere e strutturare i dati.
- Comprensibilità elevata dei documenti grazie all'utilizzo di elementi aventi nomi non arbitrari o convenzionali ma che rispecchino il loro contenuto (ad esempio si potrebbe definire un elemento author in cui vada inserito il nome dell'autore di un libro, nel contesto di un sito creato per la vendita o la consultazione di libri), rendendo anche più facile la catalogazione dei documenti per eventuali ricerche.

Obiettivi di XML



- Permettere la pubblicazione online di documenti indipendenti dal tipo di dispositivo che vi avrà accesso.
- Permettere alle industrie di sviluppare protocolli indipendenti dalle piattaforme per lo scambio dei dati.
- Permettere agli sviluppatori di visualizzare l'informazione nel modo desiderato, grazie al supporto dei fogli di stile.

XML: definizione della sintassi



- La sintassi di XML può essere definita dall'utente a seconda del tipo di documento da creare.

- E' possibile definire delle classi di documenti significative (chiamate DTD – Document Type Definition ossia definizione di tipo del documento) oppure degli schemi di documento (XMLSchema) e associarvi particolari proprietà mediante un foglio di stile esterno, che può essere realizzato in formato CSS o XSL (XSLT per trasformazione, XSL-FO per formattazione a stampa).

XML



- L'utente può scegliere se utilizzare delle DTD già create (ad esempio sono state create delle DTD per uniformare i documenti creati per particolari settori, come MathML-Mathematical Markup Language).

Struttura di un documento XML

- Un documento XML deve innanzitutto essere ben formato, ossia rispettare tutte le regole di sintassi previste dal linguaggio:
- Presenza di un unico elemento radice che contiene tutti gli altri.
- Chiusura di tutti gli elementi aperti (a meno che siano elementi vuoti, che vanno comunque chiusi con sintassi del tipo
)
- Differenza tra maiuscolo e minuscolo.
- Controllo dell'indentamento dei tag.



XML Document

Sistemi distribuiti e paralleli

Struttura di un documento XML



- Un documento XML è valido se possiede una DTD a cui corrisponde.
- Dichiarazione iniziale:
`<?XML version="1.0" encoding="UTF.8"?>`
- Encoding permette di definire il tipo di codifica utilizzato per presentare i dati. Sono ammesse la maggior parte delle codifiche esistenti.

Struttura di un documento XML

- Un documento XML:
 - E' basato su una struttura gerarchica.
 - Inizia con una "radice", ossia un elemento padre che contiene e delimita tutti gli altri e ricorre una sola volta all'interno del documento.
 - E' composto da entità, elementi e commenti.
- Tutti gli elementi vanno annidati in modo preciso perché il documento sia accettabile.



XML Document

Analisi e Visualizzazione del documento



- Un documento XML viene interpretato da una specifica applicazione, costituita da due parti fondamentali:
 - **Parser**
 - **Processore**

Analisi e Visualizzazione del documento

- Il **parser** esegue il controllo sintattico del documento e si occupa della gestione degli errori.
- Il controllo avviene su due livelli:
 - ✧ Sulla validità del documento, se esiste una DTD.
 - ✧ Sulla forma del documento (se è ben formato oppure no).
- Un **processore**, che si occupa di visualizzare il documento utilizzando un apposito foglio di stile.
- Per ottenere una visualizzazione della pagina web è necessario aggiungere valore semantico agli elementi dichiarati, ossia applicare al documento un foglio di stile (CSS o XSL) tramite una dichiarazione posta subito dopo l'iniziale dichiarazione del linguaggio:

```
<?xml:stylesheet href="xml.css" title="Stile"  
type="text/css"?>
```

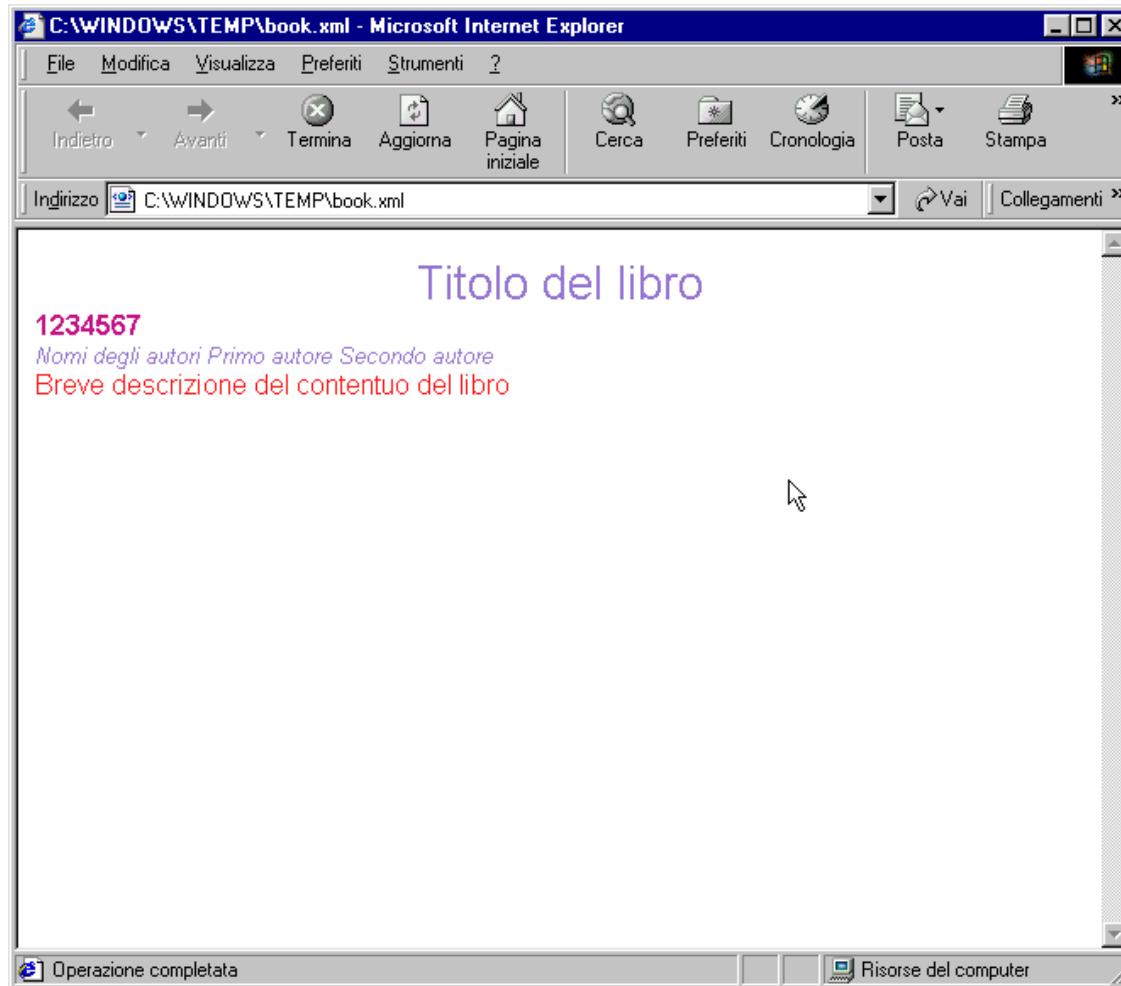


XML Document



Documento CSS

Analisi e Visualizzazione del documento



XML: namespace

- Lo spazio dei nomi è il nome attribuito alla lista degli elementi contenuti nel documento.
- Modo per eliminare ambiguità nell'uso di elementi, marcatori e attributi.
- Ognuno degli elementi è preceduto da un prefisso, separato dal nome dal carattere ":". es fo:, utilizzato da XSL.

curriculum.fo

XML: namespace



- ❑ XML utilizza l'attributo predefinito "xmlns" (eXtensible Markup Language Name Space) per introdurre i prefissi utilizzati.
- ❑ Il valore di quest'attributo è un **URI** (Uniform Resource Identifier), ad esempio <http://www.w3.org/1999/XSL/Transform>
- ❑ Ogni spazio dei nomi deve essere dichiarato prima di poter essere usato.

XML: namespace



- La sintassi per la dichiarazione è:

`xmlns:[prefisso] = "[URI]"`

- Esempio di dichiarazione per xsl:fo

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

XML: sintassi



- Forma degli elementi:

```
<nome  
 (attributo:valore) ></nome>
```

- Elementi vuoti :

```
<nome />
```

- I tag possono iniziare con una lettera, un underscore (_), oppure ":". Non è possibile iniziare nessun tag con la combinazione di lettere "xml", né maiuscole né minuscole.

Microsoft e XML



- ❑ Spazio dei nomi XSLT:
 - Explorer richiede lo spazio dei nomi relativo alla versione precedente di XSL (<http://www.w3.org/TR/WD-xsl>) mentre le specifiche W3C richiedono <http://www.w3.org/1999/XSL/Tranform>
- ❑ Tipo MIME XSLT: "text/xsl" invece che "text/xml".
- ❑ Template di default: non supportato.

XML e Internet



- In un documento web dinamico i componenti vengono elaborati e composti solo nel momento in cui arriva una richiesta esplicita. Questo tipo di documento è utilizzato nei casi in cui sia necessario generare dei contenuti in modo automatico o in risposta ad un'operazione interattiva effettuata dall'utente.

- XML è una ottima soluzione per la gestione dei contenuti dinamici.

XML e Internet



- L'approccio ai contenuti dinamici più in voga attualmente è quello multimodale:
 - Una risorsa web è disponibile a più media differenti.
 - Ogni medium riceve la risorsa web secondo modalità differenti.
 - Non è necessario scrivere diverse versioni dello stesso testo per medium diversi.

XML e Internet



- Le tecnologie disponibili possono essere divise in due gruppi principali:
 - Tecnologie dal lato client.
 - Tecnologie dal lato server.

XML e Internet



- La scelta dell'una o dell'altra avverrà in base a:
 - Considerazioni sul carico di lavoro da affidare a client e server.
 - Considerazione sull'affidabilità della tecnologia.
 - In alcuni casi, inoltre, si sceglierà una tecnologia client side, con una sorta di "soluzione di riserva".
 - Le tecnologie client side (JavaScript, CSS o Java) sono soggette alle impostazioni dell'utente, e possono quindi avere limitazioni o essere addirittura disattivate.
- Comunemente, non si utilizzano tecnologie client side per compiti critici.

XML e Internet



- E' possibile effettuare la trasformazione del codice XML sul lato server mediante:
 - Codice ASP.
 - PHP.
 - Java servlet.
 - CGI (ex Perl, Python etc.).
 - Appositi processori (ex. Sablotron, Xalax, Saxon).
- Per un utilizzo in Internet è sempre preferibile trasformare XML dal lato server.

XML e Internet



- Prima di realizzare un applicativo basato su XML valutare:
 - Il codice disponibile (se applicabile).
 - Le prestazioni necessarie.
 - Il grado di conoscenza della piattaforma di sviluppo.
 - Le risorse hardware e software.
 - Le risorse economiche.

XML e Internet



- In generale per soluzioni complesse di e-commerce si preferisce usare Java su piattaforma UNIX.
- Per la maggior parte dei casi non è necessaria una piattaforma, un linguaggio specifico o un particolare hardware.
- Si possono realizzare siti web basati su XML anche con Perl!

DTD



- Una DTD descrive lo "schema" del documento.
- In particolare descrive quali sono i costituenti "legali" di un documento XML.
- Definizione dell'elemento principale, dei figli, degli attributi (con il dominio di valori assumibili).
- E' possibile anche dichiarare entità, ossia riferimenti a blocchi di dati, interni o esterni al documento.
- Un elemento XML può avere una struttura predefinita tramite la DTD in modo da vincolarne il contenuto.

DTD: dichiarazione



- Una DTD può essere dichiarata internamente o esternamente al documento XML:
- Se viene dichiarata internamente si deve utilizzare, all'interno del file XML, la seguente sintassi:
`<!DOCTYPE root-element [element-declarations]>`

DTD : dichiarazione esterna



- Una DTD può essere esterna al file XML.
- In questo caso la DTD viene salvata in un file avente estensione .dtd e richiamata all' interno del documento XML con la sintassi:
`<!DOCTYPE root-element SYSTEM "filename">`

DTD: elementi

- Gli elementi possono essere dichiarati con la sintassi:
 - <!ELEMENT element-name category>, nel caso di elemento che non ha figli
 - <!ELEMENT element-name (child-element-name)>, per un elemento con figli: i nomi dei figli vengono dichiarati all'interno delle parentesi.
- Le virgole, a separare gli elementi (ad esempio i figli), indicano un ordine di successione obbligatorio.

DTD: elementi



- Gli elementi vuoti vengono dichiarati con la sintassi: <!ELEMENT element-name EMPTY>
- Gli elementi di tipo carattere sono dichiarati con la sintassi: <!ELEMENT element-name (#PCDATA)>
- Gli elementi di tipo misto sono dichiarati con la sintassi: <!ELEMENT element-name ANY>.

DTD: elementi

- Gli elementi dichiarati possono essere:
 - Obbligatori, ma presenti una sola volta <!ELEMENT element-name (child-name)>
 - Obbligatori, e presenti almeno una volta <!ELEMENT element-name (child-name+)>
 - Facoltativi e presenti n volte: <!ELEMENT element-name (child-name*)>
 - Facoltativi e, se presenti, una volta sola: <!ELEMENT element-name (child-name?)>
- Per dichiarare due elementi presenti in alternativa si usa la sintassi:

<!ELEMENT note (to,from,header,(message|body))>

DTD: attributi

- Per ogni elemento possono essere indicati una serie di attributi (ATTLIST), definiti in base al tipo di carattere (PCDATA o CDATA) e alla obbligatorietà o meno della loro presenza.
- 4 casi base:
 - #DEFAULT, valore di default dell' attributo
 - #REQUIRED, la presenza dell'attributo è obbligatoria.
 - #FIXED, il valore dell'attributo è fissa
 - #IMPLIED, l'attributo è previsto ma non obbligatorio e non ha un valore di default.

DTD: attributi



□ Esempi di valori degli attributi:

- PCDATA (valore di tipo carattere)
- ID (valore di tipo ID)
- IDREF (valore che indica riferimento a un altro ID)
- ENTITY (entità)

DTD: attributi - default



- Sintassi:

```
<!ATTLIST element-name attribute-name attribute-type  
"default-value">
```

- DTD:

```
<!ATTLIST payment type CDATA "check">
```

- XML:

```
<payment type="check" />
```

DTD: attributi - implied



- Sintassi:

```
<!ATTLIST element-name attribute-name attribute-type  
#IMPLIED>
```

- DTD:

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

- XML:

```
<contact fax="555-667788" />
```

DTD: attributi - required



- Syntassi:

```
<!ATTLIST element-name attribute_name attribute-type  
#REQUIRED>
```

- DTD:

```
<!ATTLIST person number CDATA #REQUIRED>
```

- XML:

```
<person number="5677" />
```

DTD: attributi - fixed



□ Syntassi:

```
<!ATTLIST element-name attribute-name attribute-type  
#FIXED "value">
```

□ DTD:

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

□ XML :

```
<sender company="Microsoft" />
```

DTD: lista attributi



- Sintassi:

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

- DTD:

```
<!ATTLIST payment type (check|cash) "cash">
```

- XML:

```
<payment type="check" /> or <payment type="cash" />
```

DTD: entità



- Le entità servono per dichiarare dei testi o delle iscrizioni fisse che vengono inserite ripetutamente all'interno dei documenti.
- Possono essere interne o esterne.

DTD: entità interne



- Sintassi:

```
<!ENTITY entity-name "entity-value">
```

- DTD:

```
<!ENTITY writer "Donald Duck.">
```

```
<!ENTITY copyright "Copyright XXXX.">
```

- XML:

```
<author>&writer;&copyright;</author>
```

DTD: entità esterne

□ Sintassi:

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

□ DTD:

```
<!ENTITY writer SYSTEM  
"http://www.xxxx.com/entities/entities.xml">  
<!ENTITY copyright SYSTEM  
"http://www.xxxx.com/entities/entities.dtd">
```

□ XML:

```
<author>&writer;&copyright;</author>
```

Esercizio



- Creare una DTD che descriva un curriculum con:
 - Dati personali
 - Esperienze formative
 - Esperienze Professionali
- Creare un documento XML contenente il proprio curriculum e validarlo rispetto alla DTD (utilizzare XMLSpy)

XML Schema



- XML Schema è un'alternativa alla DTD, basata su XML, per la descrizione della struttura di un documento XML.
- Il linguaggio utilizzato nell'XML Schema è descritto nell'XML Schema Definition (XSD).
- Il prefisso per qualsiasi elemento di un XML Schema è xsd:

XML Schema: caratteristiche



- Definisce gli elementi che appaiono in un documento
- Definisce gli attributi possibili
- Definisce le relazioni di parentela tra gli elementi
- Definisce l' ordine degli elementi figli
- Definisce il numero degli elementi figli
- Definisce se un elemento è vuoto o contiene testo
- Definisce i tipi di dati per gli elementi e gli attributi
- Definisce I valori di default e fissi per gli elementi e gli attributi

XML Schema vs DTD



- XML Schema è facilmente estendibile
- XML Schema è più espressivo
- XML Schema è scritto in XML
- XML Schema supporta la descrizione di tipi di dati
- XML Schema supporta i namespace

XML Schema: dichiarazione

- Un XML Schema è contenuto in un file, avente estensione .xsd, che viene richiamato all' interno del file XML con la sintassi:

```
<note xmlns="http://www.w3schools.com"  
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xsi:schemaLocation=  
      "http://www.w3schools.com/schema/note.xsd">
```



W3C Schema



XML Document

XML Schema: elemento root

- L'elemento di base di un documento XML Schema è:

```
<xsd:schema>
```

- Esempio:

```
<xsd:schema
```

```
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
    targetNamespace="http://www.xxxx.com"
```

```
    xmlns="http://www.xxxx.com" >
```

- Può contenere attributi che fanno riferimento alla localizzazione del namespace:

- `xmlns:xs=http://www.w3.org/2001/XMLSchema`, fa riferimento al suo namespace

- `targetNamespace=http://www.xxxx.com`, fa riferimento al namespace utilizzato nel documento XML

- `xmlns=http://www.xxxx.com`, indica il namespace di default

XML Schema: elementi semplici

- Un elemento semplice può contenere solo testo.
- Il testo può essere però di differenti tipi: booleano, stringa, data etc.
- Tipi di dati:
 - xsd:string
 - xsd:decimal
 - xsd:integer
 - xsd:boolean
 - xsd:date
 - xsd:time

XML Schema: elementi semplici



□ Codice XML

```
<lastname>Refsnes</lastname>
<age>34</age>
<dateborn>1968-03-27</dateborn>
```

□ XML Schema

```
<xsd:element name="lastname" type="xsd:string"/>
<xsd:element name="age" type="xsd:integer"/>
<xsd:element name="dateborn" type="xsd:date"/>
```

XML Schema: attributi

- Qualsiasi elemento possieda un attributo è considerato un elemento complesso.
- La sintassi per definire gli attributi è:

```
<xsd:attribute name="xxx" type="yyy"/>
```
- Esempio di codice XML:

```
<lastname lang="EN">Smith</lastname>
```
- Corrispondente dichiarazione di attributo:

```
<xsd:attribute name="lang" type="xsd:string"/>
```

XML Schema: attributi



- XML Schema permette di specificare un valore fisso oppure un valore di default per gli attributi:

```
<xsd:element name="color" type="xsd:string" default="red"/>
```

```
<xsd:element name="color" type="xsd:string" fixed="red"/>
```

- Tutti gli attributi sono opzionali ma si può comunque forzarne la dichiarazione:

```
<xsd:attribute name="lang" type="xsd:string"  
use="optional"/>
```

- E' anche possibile dichiarare un attributo obbligatorio:

```
<xsd:attribute name="lang" type="xsd:string"  
use="required"/>
```

XML Schema: vincoli



- XML Schema permette di stabilire dei vincoli per i valori assumibili da un attributo o da un elemento.
- Il vincolo si stabilisce tramite l'elemento `xsd:restriction`
- All'interno di questo elemento vengono inseriti i valori assumibili dall'attributo o da un elemento.
- Tramite il tag `<xsd:sequence>` si indica la sequenza obbligatoria degli elementi

XML Schema: vincoli



- Si possono avere vincoli di:
 - Enumerazione (xsd:enumeration)
 - Range di valori (xsd:pattern)
 - Range numerico (maxExclusive, maxInclusive, minExclusive, minInclusive)
 - Rispetto degli spazi (xsd:whiteSpace)
 - Lunghezza (xsd:length, xsd:minLength, xsd:maxLength)

Vincolo di enumerazione



```
<xsd:element name="car">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Audi"/>
      <xsd:enumeration value="Golf"/>
      <xsd:enumeration value="BMW"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Range di valori



```
<xsd:element name="letter">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:string">  
      <xsd:pattern value="[a-z]" />  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

Range numerico



```
<xsd:element name="age">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Rispetto degli spazi

```
<xsd:element name="address">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:string">  
      <xsd:whiteSpace value="preserve"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

- Possibili valori dell'attributo `xsd:whiteSpace`:
 - Preserve - mantiene tutti gli spazi così come scritti
 - Replace – rimpiazza tutti gli spazi con uno spazio singolo
 - Collapse – elimina tutti gli spazi bianchi

Vincolo di lunghezza



- ❑ xsd:length definisce la lunghezza di un attributo:

```
<xsd:element name="password">  
  <xsd:simpleType>  
    <xsd:restriction base="xsd:string">  
      <xsd:length value="8"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

- ❑ xsd:minLength e xsd:maxLength definiscono la lunghezza minima e massima che l'attributo deve avere.

XML Schema: elementi complessi

- Un elemento complesso è un elemento che può contenere altri elementi e/o attributi.
- Può:
 - Essere vuoto
 - Contenere altri elementi
 - Contenere solo testo
 - Contenere sia altri elementi che testo
- E' sempre introdotto dal tag <xsd:element>, che conterrà questa volta un tag <xsd:complexType>

XML Schema: elementi complessi



- E' possibile definire un elemento complesso creando direttamente un elemento:

```
<xsd:element name="employee">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element name="firstname" type="xsd:string"/>  
      <xsd:element name="lastname" type="xsd:string"/>  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>
```

XML Schema: estensioni di elementi complessi

- E' anche possibile estendere degli elementi complessi, ereditando le loro proprietà e aggiungendone altre, tramite il tag <xsd:extension> e l'attributo base, che indica l'elemento che si vuole estendere:

```
<xsd:element name="employee" type="fullpersoninfo"/>
<xsd:complexType name="personinfo">
  <xsd:sequence>
    <xsd:element name="firstname" type="xsd:string"/>
    <xsd:element name="lastname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

XML Schema: estensioni di elementi complessi

```
<xsd:complexType name="fullpersoninfo">
  <xsd:complexContent>
    <xsd:extension base="personinfo">
      <xsd:sequence>
        <xsd:element name="address" type="xsd:string"/>
        <xsd:element name="city" type="xsd:string"/>
        <xsd:element name="country" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

XML Schema: elementi complessi vuoti

- Per definire un elemento vuoto, ossia senza contenuto, è necessario definire un elemento di tipo complesso che possa contenere solo elementi al suo interno (e non testo!) ma non dichiarare poi alcun elemento contenuto.

```
<xsd:element name="product">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:integer">
        <xsd:attribute name="prodid" type="xsd:positiveInteger"/>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

XML Schema: elementi complessi solo testo

- In XML Schema è possibile definire degli elementi complessi che contengano solo testo.

```
<xsd:element name="shoesize">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:integer">
        <xsd:attribute name="country" type="xsd:string" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

XML Schema: elementi complessi misti

- Tramite l'attributo mixed="true" è possibile indicare che il valore di un elemento è sia testo che altri elementi figli.

```
<xsd:element name="letter">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="orderid" type="xsd:positiveInteger"/>
      <xsd:element name="shipdate" type="xsd:date"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

XML Schema: elementi complessi misti

- Tramite l'attributo mixed="true" è possibile indicare che il valore di un elemento è sia testo che altri elementi figli.

```
<xsd:element name="letter">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="orderid" type="xsd:positiveInteger"/>
      <xsd:element name="shipdate" type="xsd:date"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Esercizio

- Creare un XML Schema che descriva la struttura di un documento “reminder”, contenente:
 - To
 - From
 - Heading
 - Body
- Creare un documento XML basato sullo schema, che sia valido.

XML DOM

- **DOM XML (Document Object Model) è un'interfaccia standard per la programmazione dei documenti XML.**
- **Definisce il modo in cui si può avere accesso ai documenti XML e manipolarli.**
- **E' possibile creare documenti XML, navigare al loro struttura ad albero, aggiungere, eliminare, modificare gli elementi.**

XML DOM



- Per accedere al modello a oggetti di un documento è necessario utilizzare un XML parser.
- E' possibile utilizzare svariati linguaggi di programmazione/scripting
- Si basa sull'esistenza di:
 - Proprietà, leggibili e modificabili
 - Metodi, invocabili

DOM: nodi - proprietà

- E' possibile accedere ai nodi di un documento XML per leggerne il nome, il valore, modificarne i parametri etc.

`attributes` Restituisce tutti gli attributi del nodo

`ChildNodes` Restituisce la lista dei figli del nodo

`firstChild` Restituisce il primo figlio

`lastChild` Restituisce l'ultimo figlio

`nextSibling` Restituisce il fratello

DOM: nodi - proprietà

nodeName	Restituisce il nome del nodo
nodeType	Restituisce il tipo del nodo
nodeValue	Restituisce il valore del nodo
ownerDocument	Restituisce il nodo root
parentNode	Restituisce il nodo padre
previousSibling	Restituisce il precedente nodo con lo stesso padre

DOM: nodi - metodi

appendChild(newChild)

Appende il nodo

cloneNode(boolean)
del

Restituisce una copia
nodo

hasChildNodes()
nodo

Restituisce true se il
ha figli

insertBefore(newNode,refNode)

Inserisce un nuovo nodo
prima di un determinato
nodo esistente

replaceChild(newNode,oldNode)

Sostituisce un nodo
esistente con uno nuovo
Rimuove un nodo

removeChild(nodeName)

DOM: lista nodi proprietà e metodi



Proprietà

length Restituisce il numero di nodi di una lista

Metodi

item Restituisce uno specifico nodo all'interno di una lista.

DOM: elementi - proprietà



documentElement

Restituisce l'elemento root

doctype

Restituisce la DTD o lo Schema del
documento

DOM: elementi - metodi



createAttribute(attributeName)	Crea un attributo
createCDATASection(text)	Crea una sezione CDATA
createComment(text)	Crea un nodo commento
createElement(tagName)	Crea un elemento
createProcessingInstruction (target, text)	Crea una processing-instruction
getElementsByName(tagName) partire dal	Ritorna una lista di nodi a nodo selezionato
createTextNode(text)	Crea un nodo testuale

DOM: attributi - metodi



name	Specifica o restituisce il nome dell'attributo
value	Specifica o restituisce il valore dell'attributo
specified nella	Restituisce true se il valore di un attributo è settato nel documento, false se è settato DTD o Schema.

DOM: testo - metodi



`splitText(number)`

Restituisce il testo a partire dal carattere inserito.

Modifica di un documento XML



Per modificare un documento XML è innanzitutto necessario creare un'istanza del parser e caricare il documento:

```
<script type="text/javascript">
//istanza parser
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
//istruzione di attesa
xmlDoc.async="false"
// caricamento documento
XmlDoc.load("note.xml")
</script>
```

Creazione di un documento XML



Per creare un documento XML utilizzando Dom è necessario innanzitutto creare un nuovo oggetto di tipo documento XML

Es. JavaScript:

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
```

Es. VBScript in ASP:

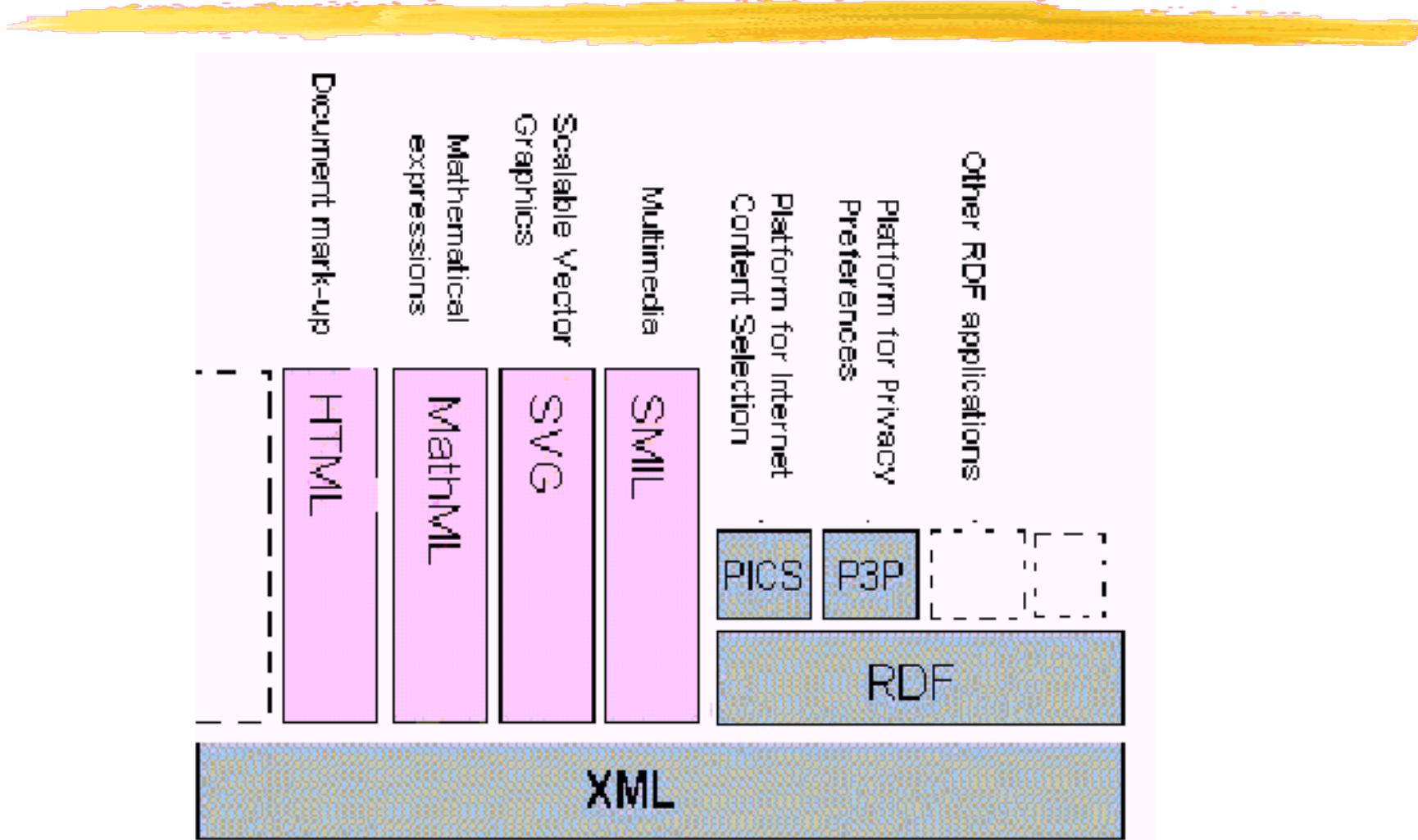
```
set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")
```

XML e dialetti



XML può essere considerato come la base di nuovi linguaggi, tra cui MathML, SMIL, RDF che hanno una sintassi che si fonda sulle regole e i costrutti base stabiliti da XML.

XML e dialetti



Web services definizione

- Definizione ufficiale del W3C:
 - "A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols"
- I web services sono definibili come una soluzione per permettere la comunicazione tra applicazioni in ambito internet.
- L'idea su cui si basano è quella di fornire un linguaggio e una piattaforma di accesso comune a sistemi differenti.

I servizi Web

- **Un servizio Web è un elemento funzionale disponibile in un punto qualsiasi di Internet e accessibile tramite protocolli comuni come HTTP e SMTP.**
- **Caratteristiche specifiche di un servizio Web:**
 - E' basato su XML
 - ✧ il ricorso a XML (al posto di altri linguaggi binari proprietari) come linguaggio di rappresentazione dei dati per tutti i protocolli e le tecnologie dei servizi Web disponibili realizza di fatto la loro completa interoperabilità, eliminando tutte le specificità dovute alla rete, al sistema operativo o alla piattaforma.
 - Non è strettamente correlato (loosely coupled)
 - ✧ L'utilizzatore (client) non è legato in modo diretto a un determinato servizio Web, la sua interfaccia può essere modificata nel tempo senza per questo compromettere la capacità del client di interagire con il servizio stesso. Ciò consente ai sistemi software manutenzione e integrazione più semplici.

Caratteristiche dei servizi Web

- E' " a grana grossa" (coarse grained)
 - ✧ le tecnologie orientate agli oggetti come Java rendono disponibili i loro servizi tramite singoli metodi. Dal punto di vista funzionale, nel creare un programma Java da zero occorre creare i vari metodi "a grana fine" da assemblare in un servizio "a grana grossa" utilizzabile dal client o da un altro servizio. La tecnologia dei servizi Web fornisce un metodo naturale per la definizione di servizi "a grana grossa" che accedono ad una quantità di logica funzionale.
- Può essere indifferentemente sincrono o asincrono
 - ✧ Il client può attendere in maniera bloccante o no il completamento delle operazioni del server.
- Supporta le chiamate RPC (Remote Procedure Calls)
 - ✧ Un servizio Web supporta RPC fornendo servizi di un componente tradizionale oppure traducendo le chiamate in ingresso in altrettante chiamate a componenti EJB (Enterprise JavaBeans) o .NET.
- Supporta lo scambio di documenti
 - ✧ Uno dei vantaggi principali di XML è la capacità di rappresentare in maniera generica non soltanto dati, ma anche documenti semplici (un indirizzo) o complessi (un libro intero).

Tecnologie per i servizi Web

Negli ultimi anni si sono affermate tre importanti tecnologie considerate standard mondiali:

- Il protocollo SOAP (Simple Object Access Protocol)
 - Fornisce una struttura standard di incapsulamento per il trasporto dei documenti XML attraverso un certo numero di protocolli Internet standard tra cui SMTP, HTTP, FTP. Definisce anche gli standard e i collegamenti per chiamate RPC per il trasporto su XML.
- Il linguaggio WSDL (Web Service Description Language)
 - WSDL è una tecnologia XML che descrive in modo standardizzato l'interfaccia di un servizio Web: la presentazione da parte del servizio dei parametri d'ingresso e d'uscita di una chiamata a procedura e il suo tipo (se produce o no risultati), la struttura di una funzione ed i collegamenti ad un protocollo di trasporto imposti dal servizio stesso. WSDL rende possibile a qualsiasi client l'interazione con un servizio Web.
- I servizi UDDI (Universal Description, Discovery and Integration)
 - UDDI è un organismo che fornisce un registro mondiale per promuovere la pubblicità, la scoperta e l'integrazione dei servizi Web.

Web service stack

Web Service Stack

UDDI

WSDL

SOAP

XML

HTTP, TCP/IP...

○ XML (Extensible Markup Language)

World Wide Web Consortium, XML, w3.org/XML/

- ◆ Formato universale dei documenti e dati strutturati pubblicati su Web.

○ SOAP(Simple Object Access Protocol)

World Wide Web Consortium, XML Protocol Activity, w3.org/2002/ws/

- ◆ Protocollo per lo scambio di informazioni in ambiente distribuito.

○ WSDL (Web Services Definition Language)

World Wide Web Consortium Technical Report, w3.org/tr/wsdl.

- ◆ Formato XML per la descrizione dei servizi di rete.

○ UDDI (Universal Description, Discovery and Integration)

UDDI.ORG, uddi.org/about.html.

- ◆ Iniziativa di settore mirante a consentire il reperimento e l'utilizzo dei Web Service in modo rapido, semplice e dinamico.
Sistemi distribuiti e paralleli

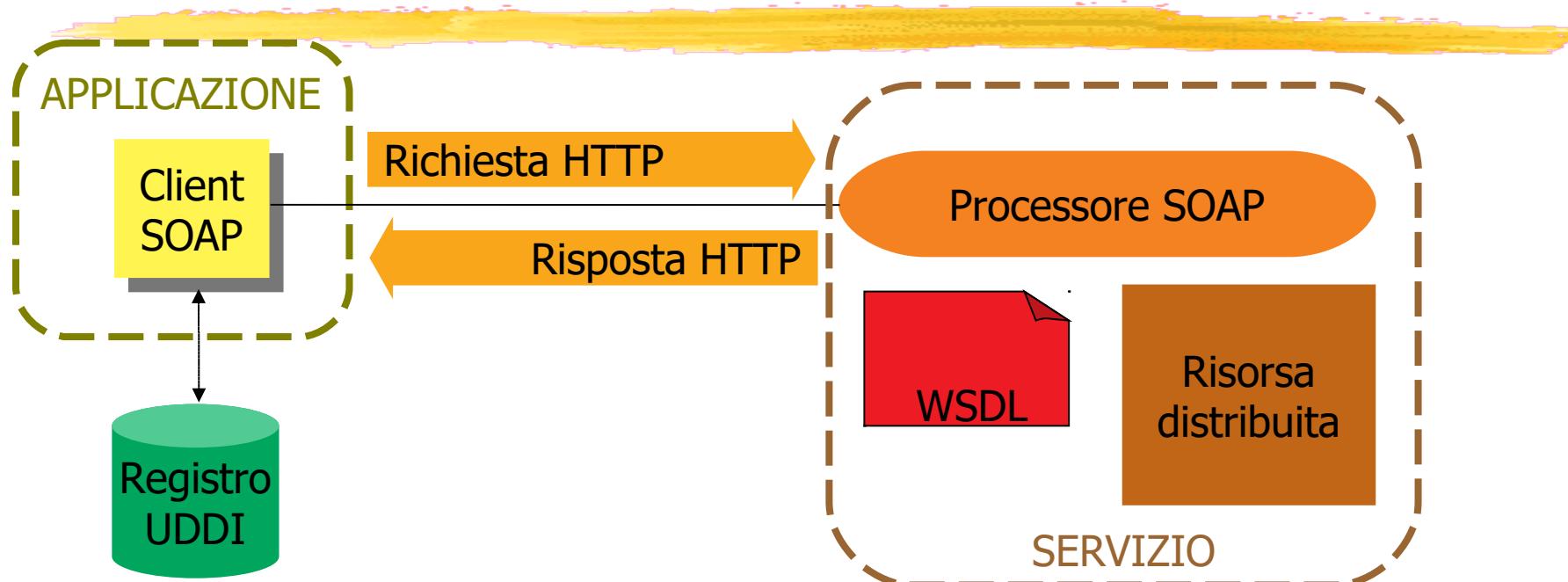
Altra definizione di Web service



La definizione base di Web Service data dall'enciclopedia TechWeb (TechWeb, The Business Technology Network, techweb.com/encyclopedia/.) è la seguente:

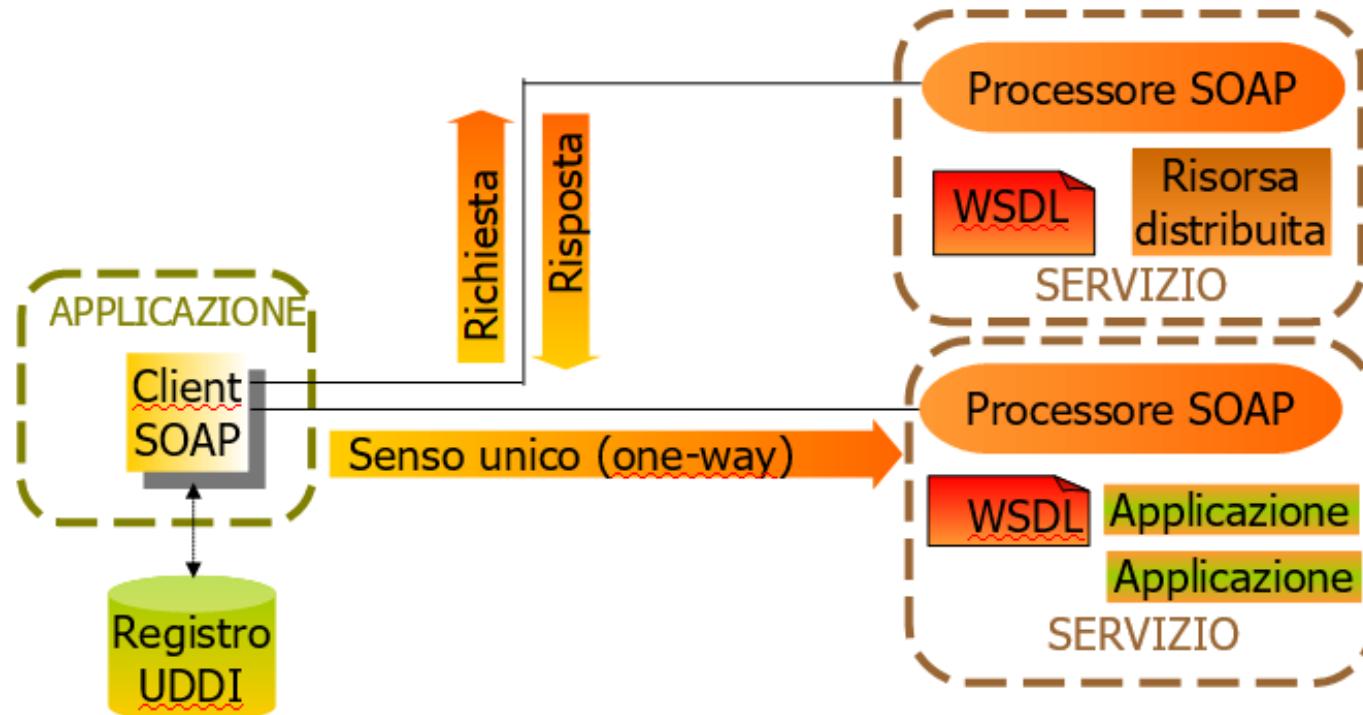
Web Service: applicazione Web in grado di interagire dinamicamente con altre applicazioni Web attraverso un protocollo di messaggistica XML, ad esempio SOAP.

Interazione semplice in un servizio Web



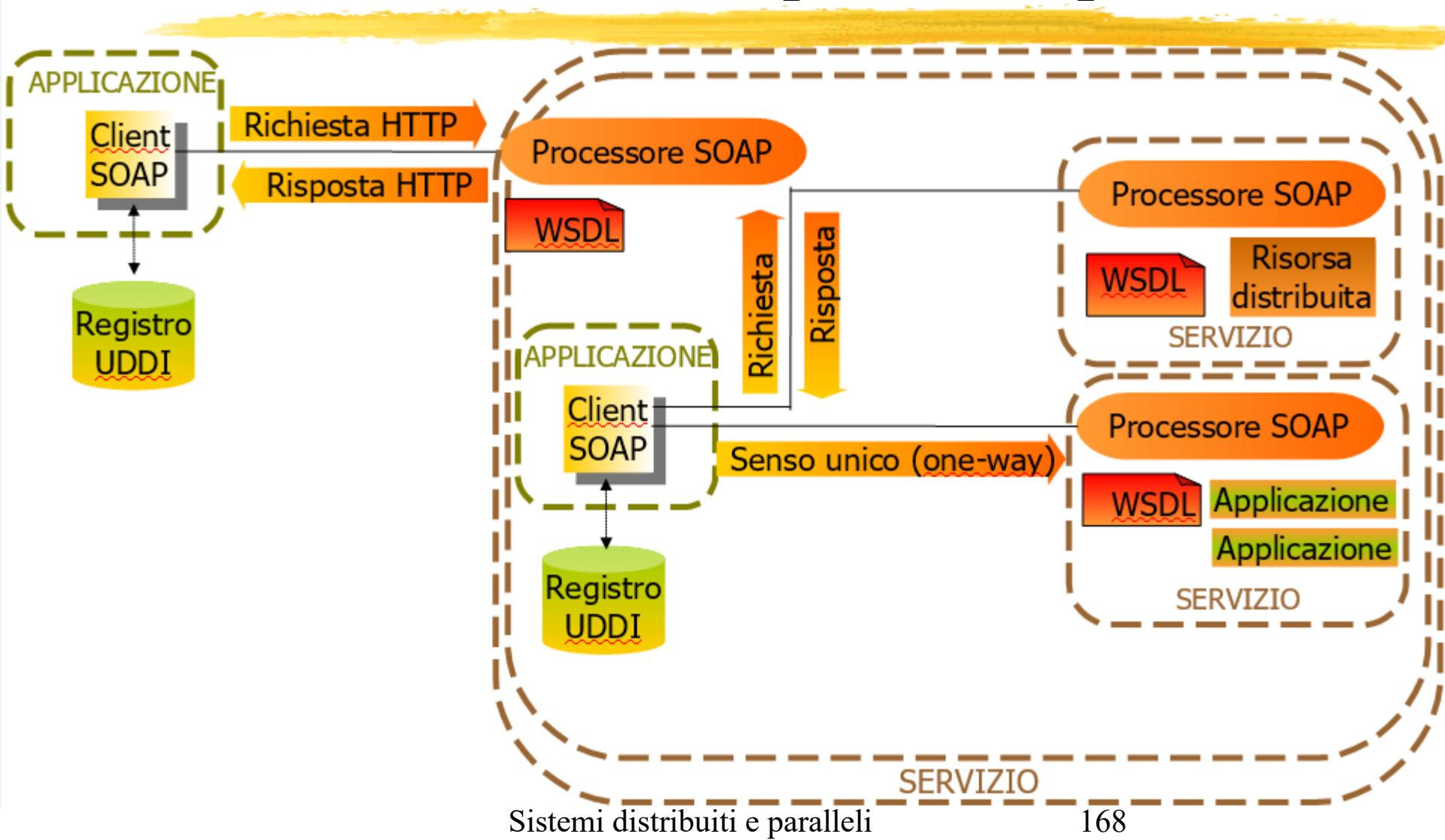
Il client di un servizio Web deve rintracciare un'applicazione o un elemento funzionale di un programma che si trova nella rete. Esso interroga un registro UDDI, effettuando una ricerca per nome, categoria, identificativo o specifiche, e ottiene informazioni sulla posizione di un documento WSDL, nel quale sono indicate le modalità per mettersi in contatto con il servizio Web e il formato dei messaggi di richiesta sotto forma di schema XML. Il client creerà un messaggio SOAP conforme allo schema XML trovato nel documento WSDL e invierà una richiesta all'host che dispone del servizio.

I componenti separati dell'architettura di un servizio Web



Il servizio rappresenta una serie integrata di applicazioni per realizzare una specifica funzione. Ogni servizio si può considerare come elemento completamente autonomo (*self-contained*) e definito in ogni particolare (*self-describing*) come parte integrante di un sistema di dimensioni più ampie, a prescindere dal fatto che sia implementato come componente "a grana fine" e incaricato di eseguire una singola operazione oppure come insieme d'applicazioni che eseguono un'intera gamma di funzionalità aziendali.

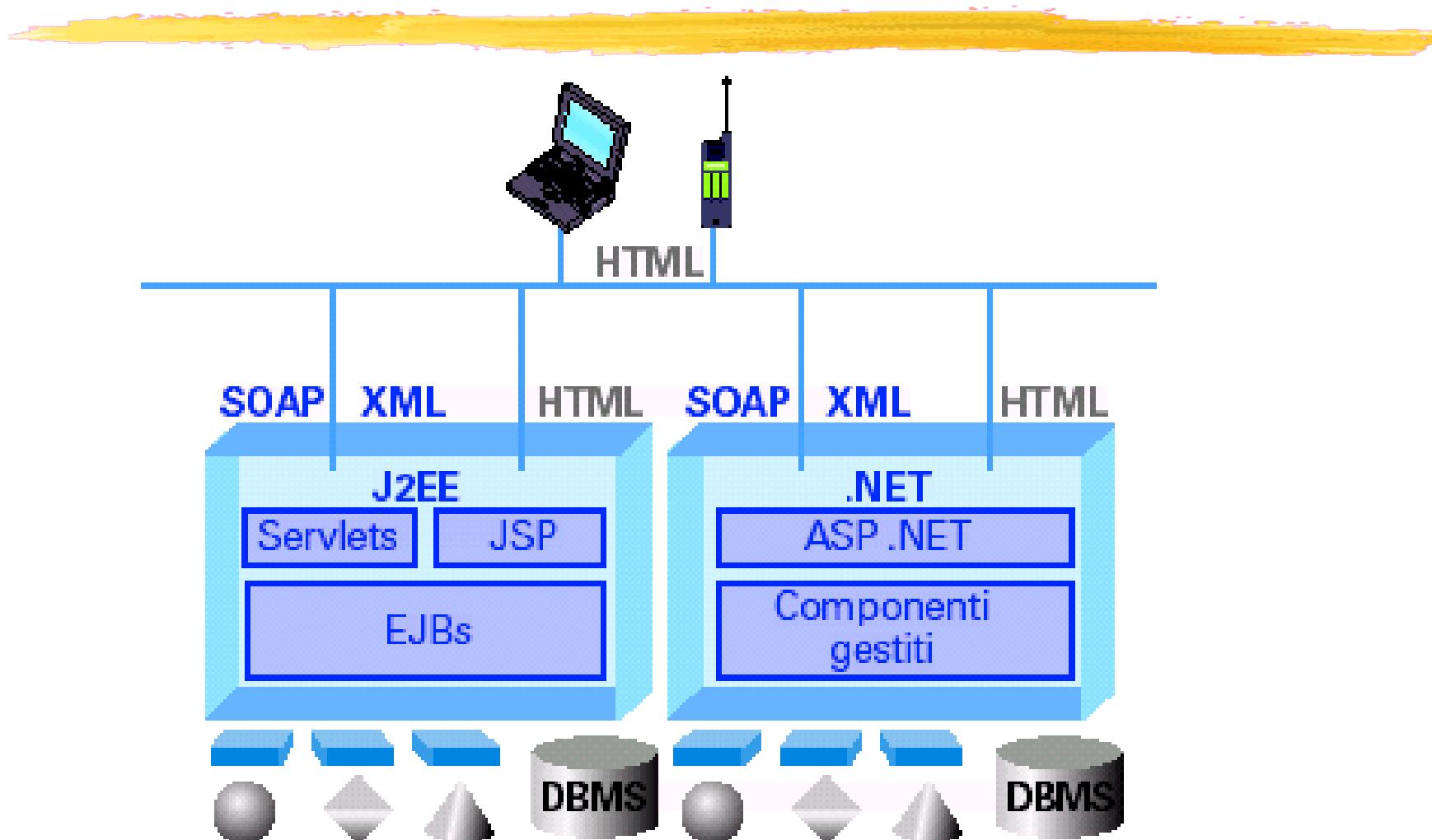
Servizi Web all'interno di un ecosistema più ampio



Piattaforme per Web service

- La tendenza del settore è di rendere disponibili le infrastrutture per Web Service come funzioni integrate delle piattaforme di application server, e tutti i vendor si sono già attrezzati in questo senso. Nel mondo odierno, ciò ha portato alla costituzione di due famiglie distinte di tecnologie:
 - JEE**— offerta da numerosi vendor fra cui BEA, IBM e Sun
(The Source for Java Technology, Sun Microsystems, Inc., java.sun.com/);
 - .NET**— offerta da Microsoft.
- JEE e .NET sono in molti sensi piattaforme concorrenti. Tuttavia sono entrambe assai diffuse, e la natura dei Web Service e degli standard che li supportano rende molto semplice la loro coesistenza in azienda. Secondo le proiezioni di settore, due terzi delle imprese implementeranno sia JEE che .NET.

Piattaforme



J2EE

- Gli application server Java sono estremamente diffusi, come dimostra l'evoluzione della specifica J2EE (Java 2 Enterprise Edition) per lo sviluppo e il deployment di componenti server distribuiti. La versione corrente dello standard contiene già opzioni relative alla distribuzione di Web Service, anche se la loro introduzione formale avverrà solo con la release 1.4. In forma semplificata, gli application server J2EE supportano le seguenti funzionalità (riportate anche più sopra in tabella):
 - applicazioni Web in HTML con JSP (Java Server Pages) e Servlet;
 - logica sul lato server basata sugli Enterprise JavaBeans (EJB);
 - integrazione dei database via JDBC;
 - integrazione della logica di business ottenuta con diversi metodi, fra cui JNI (Java Native Interface)e JCA (J2EE Connector Architecture);
 - accesso ai Web Service mediante XML e SOAP.

.NET

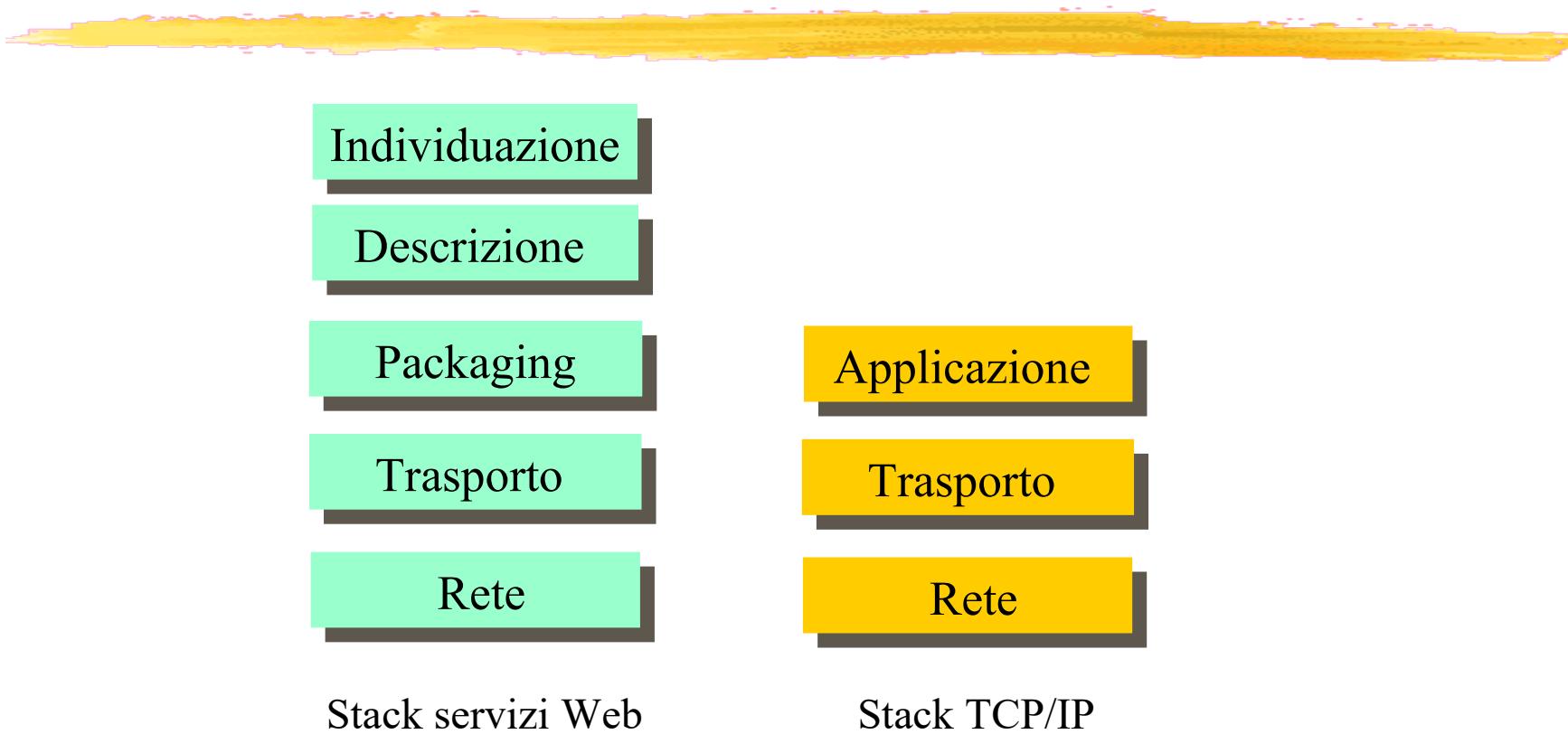


- .NET è l'architettura scelta da Microsoft per l'erogazione dei Web Service. I servizi base corrispondenti all'ambiente J2EE vengono forniti da .NET Framework e supportano, come indicato più sopra in tabella:
 - applicazioni Web in HTML con ASP.NET (Active Server Pages);
 - logica sul lato server basata su Managed Code;
 - integrazione dei database via ODBC/ADO (ActiveX Data Objects);
 - integrazione della logica di business con COM+;
 - accesso ai Web Service mediante XML e SOAP.

J2EE e .NET

- L'iniziativa Microsoft .NET è molto più di un semplice application server, e include sistemi operativi client (Windows XP) e server, una vasta gamma di server (SQL Server, Exchange Server) e una serie di Web Service veri e propri (Passport e MSN).
- Quando si valutano i Web Service, è importante che il confronto venga effettuato fra Microsoft .NET Framework e un application server J2EE.
- Tuttavia, poiché entrambe le piattaforme supportano lo standard SOAP, tutte e due le famiglie di soluzioni possono essere utilizzate per la creazione e la distribuzione di questi servizi.

Stack della tecnologia dei servizi Web



Standardizzazione dei servizi Web

Protocolli di Packaging

- Protocollo SOAP/XML
 - Originariamente acronimo di "Simple Object Access Protocol" ora è la base per W3C XML Protocol. La versione 1.1 della specifica è disponibile sul sito <http://www.w3.org/tr/soap>. La bozza di lavoro della documentazione della versione 1.2 è disponibile presso <http://www.w3.org/tr/soap12>. Ulteriori informazioni su SOAP e W3C XML Protocol possono essere trovate visitando la home page del gruppo di lavoro al sito <http://www.w3.org/2000/xp/>
 - XML-RPC
 - La forma originale di SOAP messa a punto da Dave Winer di Userland Software. Questo semplice e diffuso protocollo, se ne sia uno standard ufficiale, gode di una forte presenza nella comunità open source. Per informazioni vedi il sito <http://www.xmlrpc.org/>
 - Jabber
 - E' sia un protocollo di trasporto sia un semplice protocollo di packaging che può essere usato nei servizi Web asincroni peer-to-peer. Anch'esso non è uno standard ufficiale, ma si diffondono sempre più. Informazioni nella home di jabber <http://www.jabber.org>
 - DIME
 - Il protocollo Direct Internet Message Encapsulation (DIME) è un "formato leggero di encapsulamento binario che può essere usato per encapsulare applicazioni multiple definite come entità o payload di tipi arbitrari, oltre che per fornire un'efficace delimitazione dei messaggi". Informazioni presso http://www.gotdotnet.com/team/xml_wsspecs/default.aspx
- Sistemi distribuiti e paralleli 175

Standardizzazione dei servizi Web

Protocolli di Descrizione

□ WSDL

- Web Service Description Language è il linguaggio standard di fatto per la descrizione dei servizi Web. E' stato sottoposto al W3C per la standardizzazione e un gruppo di lavoro apposito è in fase di organizzazione. WSDL sostituisce i precedenti tentativi di descrizione proposti da IBM e Microsoft (rispettivamente NASSL e SDL). La versione 1.1 della specifica WSDL è disponibile presso <http://www.w3.org/tr/wsdl>

□ WSFL

- Il Web Services Flow Language è un'estensione di WSDL che permette l'espressione di flussi di lavoro all'interno dell'architettura dei servizi Web

□ DAML-S

- DARPA Agent Markup Language Ontology per i servizi Web è un progetto di ricerca accademica per la descrizione semantica dei servizi Web. E' possibile trovare informazioni presso <http://daml.semanticweb.org>

□ RDF

- Si è discusso molto sul fatto che RDF possa essere usato "molto facilmente" come metodo di descrizione dei servizi Web. WSDL può essere modificato per essere conforme alla sintassi RDF. DAML-S è un altro esempio costruito interamente su RDF. E' possibile trovare informazioni presso <http://www.w3.org/rdf>

Standardizzazione dei servizi Web

Protocolli di Individuazione

- UDDI
 - L'iniziativa Universal Description Discovery and Integration promette di definire un registro di servizio standard. Informazioni disponibili presso <http://www.uddi.org>
- WS-Inspection
 - Web Service Inspection Language fornisce un indice XML per individuare i servizi Web disponibili in una data porzione di rete. E' possibile trovare informazioni presso <http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html>
- ebXML Registry
 - Parte di ebXML (<http://www.w3.org/rdf>) era definire un modello di registro standard per individuare i servizi aziendali. L'approccio è in parte differente, ma non incompatibile con UDDI e include molti più tipi di informazioni rispetto a UDDI.
- JXTA Search
 - L'infrastruttura dei servizi peer-to-peer JXTA, sponsorizzata da Sun, definisce un protocollo di ricerca distribuito per l'individuazione dei contenuti e dei servizi in un'architettura peer-to-peer. E' possibile trovare informazioni presso http://jxta.org/project/www/white_papers.html

Standardizzazione dei servizi Web

Protocolli di Protezione

- XML Digital Signature
 - Un progetto congiunto del W3C e di IETF per definire un metodo standard per la rappresentazione delle firme digitali come contenuto XML (<http://www.w3.org/Signature/>)
- XML Encryption
 - La specifica Encryption XML consente di crittografare i dati XML e consente di esprimere i dati crittografati come XML
- SAML
 - Il Security Assertions Markup Language è una grammatica XML per esprimere il verificarsi di eventi di protezione, come un evento di autenticazione. Usato insieme all'architettura dei servi Web, esso fornisce un sistema di autenticazione standard piuttosto flessibile
- XKML
 - Gli CML Key Management Services sono un insieme di servizi relativi alla protezione e al trust che aggiungono ai servizi Web capacità di Private Key Infrastructure (PKI)
- P3P
 - LA W3C Platform for Privacy Preferences è una grammatica XML per l'espressione dei criteri di privacy dei dati.

Aree di applicazione



- Una delle principali aree di applicazione dei web services saranno le comunicazioni business to business (B2B).

Architettura server to server

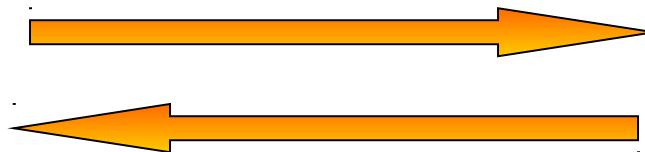
Business A -
server



Business B -
server

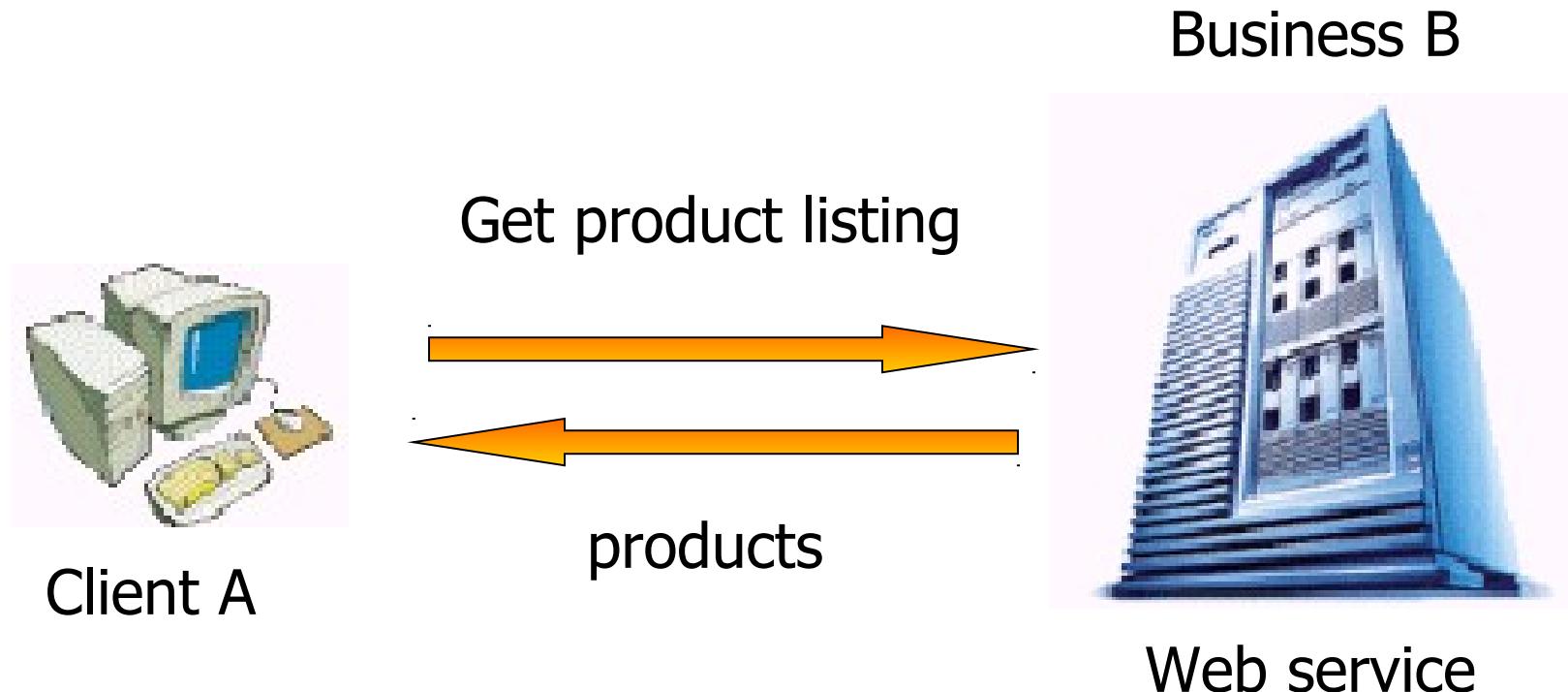


Get product listing



Return product listing

Architettura business to consumer



Architettura business-middleware-consumer

- In un'architettura business to consumer il server non è in grado di controllare come il documento appaia sul client.
- Per risolvere questo problema è possibile utilizzare un middleware tra il server e i diversi client che si occupi di "customizzare" il web service per i diversi dispositivi.
- Inoltre le aziende, che potrebbero avere applicazioni già esistenti e scritte in linguaggi come C o Cobol, potrebbero includere queste applicazioni in un web service, in modo da renderle disponibili per un elevato numero di client.

XML e i web services



- Si basano su XML per la semplice ragione che è indipendente da linguaggi, applicazioni e piattaforme specifiche.
- XML garantisce:
 - Ricchezza espressiva
 - Estendibilità
 - Portabilità
 - Facilità di comprensione
- Inoltre gli schemi XML possono essere validati da entrambe le applicazioni che comunicano.

Linguaggi

- ❑ Il linguaggio “ideale” per realizzare un web service è considerato Java, in quanto fornisce un framework eccellente per costruire applicazioni.
- ❑ Il server più utilizzato per realizzare web services è Tomcat (<http://jakarta.apache.org>).
- ❑ Supporta JSP (Java Server Pages) e Servlet.
- ❑ Esistono API Java per connettersi ai database.
 - JDBC, che fornisce connettività, esecuzione SQL, processing dei risultati.
 - Sqlj, standard ANSI per integrare SQL nelle applicazioni java.
- ❑ Java è supportato da piattaforme e device differenti (cellulari, palmari etc.)

SOAP

Introduzione



- **SOAP (Simple Object Access Protocol) è un protocollo basato su XML per la trasmissione di dati tra applicazioni su protocollo HTTP.**
- Caratteristiche:
 - SOAP è un protocollo di comunicazione
 - SOAP è stato creato per la comunicazione tra applicazioni
 - SOAP è un formato per l'invio di messaggi
 - SOAP è stato creato per la comunicazione in Internet
 - SOAP è indipendente dalle piattaforme
 - SOAP è indipendente dai linguaggi
 - SOAP è basato su XML
 - SOAP è semplice e estendibile
 - SOAP permette di “aggirare i firewall”
 - SOAP verrà sviluppato come standard W3C.

Breve storia



- Il protocollo SOAP è nato da un consorzio composto da UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, Microsoft e SAP.
- E' stato proposto al W3C per la standardizzazione nel maggio 2000.
- Il primo Working Draft del W3C è stato pubblicato nel dicembre 2001.
- L'attuale versione rilasciata è la 1.2

Situazione attuale



- Prima le applicazioni generalmente comunicavano usando Remote Procedure Calls (RPC) tra oggetti (DCOM e CORBA).
- Ci sono stati però problemi di sicurezza perché i firewall generalmente bloccano questo tipo di comunicazioni.

Caratteristiche



- SOAP è fondamentalmente un protocollo stateless e a una via.
- E' possibile però creare applicazioni che gestiscano schemi di applicazione più complessi combinando lo scambio di messaggi a una via con le caratteristiche rese disponibili dal protocollo sottostante.
- SOAP non fornisce dettagli sulla semantica dei dati che trasmette.
- SOAP non fornisce dettagli tecnici (routing dei messaggi, trasferimento affidabile dei dati, attraversamento dei firewall etc.).

SOAP Processing

- ❑ SOAP prevede uno scenario di utilizzo formato da:
 - Un messaggio SOAP originato da un SOAP sender
 - Un messaggio SOAP inviato a un destinatario
- ❑ Il messaggio nel percorso può transitare da zero o più nodi SOAP.
- ❑ Un nodo SOAP può essere:
 - Sender
 - Receiver
 - Intermediary
- ❑ Un intermediario è sia un sender che un receiver.
- ❑ Un nodo SOAP che riceve un messaggio deve processarlo.
- ❑ Un nodo SOAP deve essere identificato da un URI.

Ruoli

- ❑ Nel processare un messaggio un nodo SOAP riveste un ruolo.
- ❑ I ruoli sono identificati da un URI che rappresenta il ruolo.
- ❑ Il ruolo assunto da un nodo deve essere invariante mentre processa il messaggio.
- ❑ I ruoli previsti sono:
 - "http://www.w3.org/2002/06/soap-envelope/role/next". Ogni nodo ricevente o intermediario deve rivestire questo ruolo.
 - "http://www.w3.org/2002/06/soap-envelope/role/none". Nessun nodo SOAP deve rivestire questo ruolo.
 - "http://www.w3.org/2002/06/soap-envelope/role/ultimateReceiver". Per identificare se stesso come nodo destinazione finale un nodo SOAP deve agire in questo ruolo. I nodi intermediari non possono rivestire questo ruolo.

Ruoli



- Il ruolo rivestito da un nodo SOAP viene visualizzato nell'elemento *role* del blocco header di un messaggio.

Regole di processing

- Durante il “percorso” i nodi SOAP analizzano gli header.
- Se uno soltanto dei blocchi header non è comprensibile da un nodo che lo deve processare viene generato un errore SOAP con codice "env:MustUnderstand"
- Se viene trasmesso un messaggio di errore il messaggio non deve più essere processato.
- Errori relativi al contenuto del body non devono essere generati a questo livello.
- Nel caso si giunga al nodo destinatario finale, si deve processare l’elemento body.

Intermediari attivi



- Possono esistere dei nodi intermediari attivi che modificano il messaggio (anche il contenuto) durante il processing del messaggio.
- Ad esempio nodi che si occupano di fornire servizi di sicurezza, di annotazione, di manipolazione del contenuto.
- I nodi intermediari attivi devono descrivere i cambiamenti apportati al messaggio aggiungendo dei blocchi header.

Regole di processing (2)

- Quando un messaggio deve transitare per più nodi si creano degli header per ciascuno dei nodi intermedi.
- Quando un nodo intermediario ha effettuato il processing e lo trasmette può inserire un nuovo header per il nodo successivo.

Regole di sintassi

- Un messaggio SOAP **deve** essere codificato in XML.
- Un messaggio SOAP **deve** avere un elemento SOAP Envelope
- Un messaggio SOAP **può** avere un elemento SOAP header
- Un messaggio SOAP **deve** avere un elemento SOAP Body
- Un messaggio SOAP **deve** utilizzare un namespace SOAP Envelope
- Un messaggio SOAP **deve** utilizzare un namespace SOAP Encoding
- Un messaggio SOAP **non deve** contenere riferimenti a una DTD
- Un messaggio SOAP **non deve** contenere una XML Processing Instructions

SOAP namespace



- **SOAP envelope:**

<http://www.w3.org/2001/12/soap-envelope>

- **SOAP encoding and data types:**

<http://www.w3.org/2001/12/soap-encoding>

- **SOAP è basato su XML Schema.**

Struttura di un messaggio SOAP



- Un messaggio SOAP è un documento XML formato da tre parti costituenti:
 - SOAP envelope, che definisce il contenuto del messaggio.
 - SOAP header (opzionale), che contiene informazioni relative all'intestazione del messaggio.
 - SOAP body, che contiene informazioni sul messaggio vero e proprio.

SOAP Envelope

- L'elemento <envelope> è l'elemento radice di un documento SOAP.
- Si può considerarlo come una specie di "busta" che racchiude tutto il messaggio.
- Ha un attributo name, per identificare il messaggio.
- Contiene la dichiarazione dei namespace.
- **Può** contenere un elemento header.
- **Deve** contenere un elemento body.
- Ha un attributo encodingStyle da utilizzare per definire l'encoding degli elementi del messaggio che verranno serializzati.
- L'encoding di default è <http://www.w3.org/2002/06/soap-encoding>.

SOAP header



- SOAP header è un elemento che può contenere infiniti blocchi header figli.
- Ogni blocco header deve avere:
 - Un attributo *namespace*
 - Un attributo *encodingStyle*
 - Un attributo *role*
 - Un attributo *mustUnderstand*

MustUnderstand



- Attributo usato per indicare se il processing di un messaggio è obbligatorio o meno.
- Se l'Attributo è presente indica che, qualora vi siano problemi nella comprensione degli header (ad esempio la versione di linguaggio differente) non si può continuare ad inviare il messaggio e si deve generare un errore SOAP.
- Deve essere presente con valore = “true” (oppure 1).
- Omettere l'Attributo è equivalente a indicare “false”.

SOAP Body



- Il corpo di un messaggio SOAP è obbligatorio.
- Deve avere un attributo *name*
- Può avere un attributo *encodingStyle*
- Un figlio particolarmente importante di body è SOAP *fault*.

Richiesta SOAP corretta

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope">
  <env:Header>
    <m:reservation
      xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>John Q. Public</n:name>
    </n:passenger>
  </env:Header>
```

Richiesta SOAP corretta (2)

```
<env:Body>
<p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
    <p:departure>
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
        <p:departureDate>2001-12-14</p:departureDate>
        <p:departureTime>late afternoon</p:departureTime>
        <p:seatPreference>aisle</p:seatPreference>
    </p:departure>
    <p:return>
        <p:departing>Los Angeles</p:departing>
        <p:arriving>New York</p:arriving>
        <p:departureDate>2001-12-20</p:departureDate>
        <p:departureTime>mid-morning</p:departureTime>
        <p:seatPreference/>
    </p:return>
</p:itinerary>
```

Richiesta SOAP corretta (3)



```
<q:lodging  
  xmlns:q="http://travelcompany.example.org/reservation/hotels">  
  <q:preference>none</q:preference>  
</q:lodging>  
</env:Body>  
</env:Envelope>
```

Risposta SOAP corretta

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope">
  <env:Header>
    <m:reservation
      xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>John Q. Public</n:name>
    </n:passenger>
  </env:Header>
```

Risposta SOAP corretta (2)

```
<env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
    <p:airportChoices> JFK LGA EWR </p:airportChoices>
  </p:itinerary>
</env:Body>
</env:Envelope>
```

SOAP fault



- ❑ L'elemento SOAP fault è usato per indicare gli errori che possono intercorrere durante la trasmissione del messaggio.
- ❑ Ha come figli:
 - Code (obbligatorio)
 - Reason (obbligatorio)
 - Node (opzionale)
 - Role (opzionale)
 - Detail (opzionale)
- ❑ Un messaggio SOAP non corretto deve contenere un unico elemento SOAP come unico figlio di body.

SOAP CODE



- Fornisce un codice che spiega il tipo di errore verificatosi.
- Ha come valore un codice di errore.
- Può avere come figli:
 - Value
 - Subcode (che a sua volta può avere un figlio value)

SOAP Reason



- Elemento che fornisce una spiegazione in “linguaggio umano” dell’errore avvenuto.

SOAP Node



- ❑ Fornisce informazioni su quale nodo ha generato l'errore.

SOAP role



- Fornisce informazioni sul ruolo che il nodo stava “interpretando” al momento dell’errore.

SOAP detail



- ❑ Elemento che fornisce dettagli sull'errore.
- ❑ Deve essere presente quando l'errore è stato generato dal corpo del documento SOAP.
- ❑ Non deve essere utilizzato per trasmettere informazioni su un errore generato dagli header.
- ❑ Le informazioni su errori causati dagli header devono essere riportate nella sezione header.
- ❑ La presenza dell'elemento detail indica che almeno parte del corpo è stata processata prima del verificarsi dell'errore.

Valori dei codici di errore

□ L'elemento value può avere come valori:

- VersionMismatch: il nodo che ha generato l'errore non ha trovato un elemento radice corrispondente alle specifiche della versione SOAP.
Restituisce il messaggio aggiungendo un header Upgrade.
- MustUnderstand. Restituisce il messaggio aggiungendo un header Misunderstood
- DataEncodingUnknown
 - ❖ Sender, il messaggio non è stato generato correttamente dal mittente. Per rispedire il messaggio sarà necessario effettuare dei cambiamenti.
 - ❖ Receiver, il messaggio non è stato processato per ragioni relative al processing più che al contenuto (ad esempio potrebbe essere stato destinato a un host non raggiungibile).

Version mismatch error



```
<?xml version="1.0" ?>
<env:Envelope
    xmlns:env="http://www.w3.org/2002/06/soap-envelope">
    <env:Header>
        <upg:Upgrade xmlns:upg="http://www.w3.org/2002/06/soap-upgrade">
            <envelope qname="ns1:Envelope"
                xmlns:ns1="http://www.w3.org/2002/06/soap-envelope"/>
            <envelope qname="ns2:Envelope"
                xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/" />
        </upg:Upgrade>
    </env:Header>
```

Version mismatch error (2)



```
<env:Body>
  <env:Fault>
    <env:Code>
      <env:Value>env:VersionMismatch
      </env:Value>
    </env:Code>
    <env:Reason>Version Mismatch</env:Reason>
  </env:Fault>
</env:Body>
</env:Envelope>
```

Messaggio errato



```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2002/06/soap-envelope'>
  <env:Header>
    <abc:Extension1 xmlns:abc='http://example.org/2001/06/ext'
      env:mustUnderstand='true' />
    <def:Extension2 xmlns:def='http://example.com/stuff'
      env:mustUnderstand='true' />
  </env:Header>
  <env:Body> . . . </env:Body>
</env:Envelope>
```

Must understand error

```
<?xml version="1.0" ?>  
  <env:Envelope xmlns:env='http://www.w3.org/2002/06/soap-envelope'  
    xmlns:flt='http://www.w3.org/2002/06/soap-faults'>  
    <env:Header>  
      <flt:Misunderstood qname='abc:Extension1'  
        xmlns:abc='http://example.org/2001/06/ext' /> <flt:Misunderstood  
        qname='def:Extension2' xmlns:def='http://example.com/stuff' />  
    </env:Header>  
    <env:Body>  
      <env:Fault>  
        <env:Code>  
          <env:Value>env:MustUnderstand</env:Value>  
        </env:Code>  
        <env:Reason>One or more mandatory headers not understood</env:Reason>  
      </env:Fault>  
    </env:Body>  
</env:Envelope>
```

Sicurezza?

- Le attuali specifiche di SOAP non accennano alla sicurezza -> SOAP non è un protocollo di trasmissione sicuro.
- SOAP penetra attraverso i firewall.
- Se esiste un baco nell'applicazione che accede al server è possibile sfruttarlo per effettuare operazioni non consentite.
- SOAP è un protocollo a una via e le implementazioni per renderlo request/response rendono possibile la copia e la duplicazione dei messaggi.
- Si possono utilizzare funzioni offerte dai protocolli su cui SOAP si basa ma in questo modo ci si lega ai protocolli utilizzati.
- Si può usare SSL ma si devono abilitare i nodi intermediari alla lettura del messaggio SOAP e quindi la sicurezza viene a mancare.

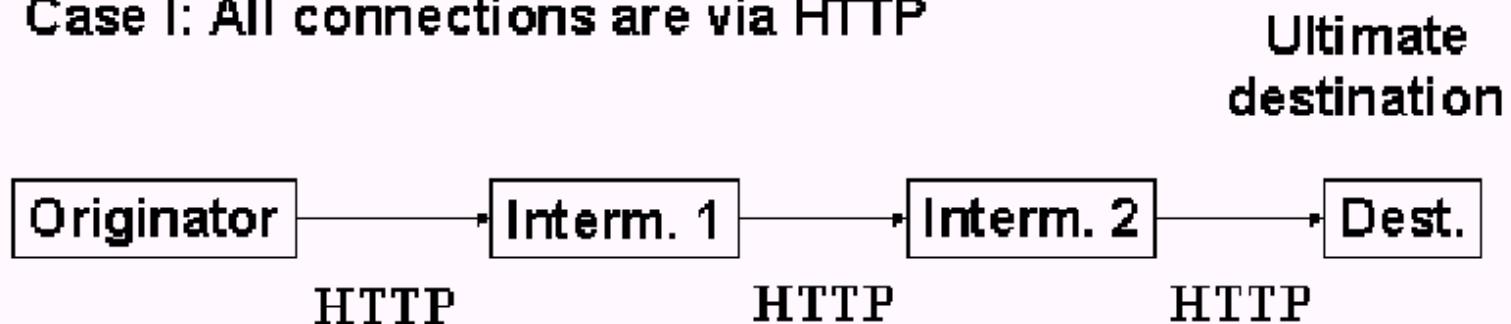
SSL



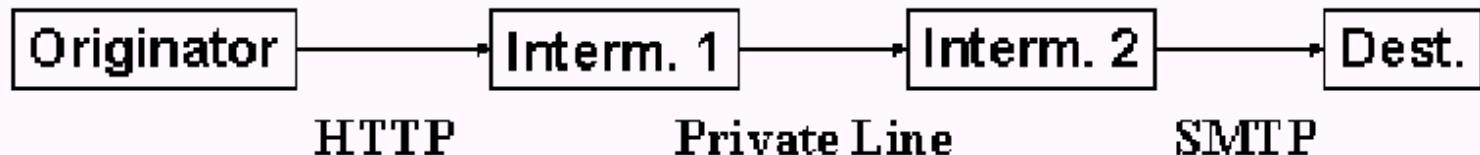
- ❑ SSL garantisce un canale di trasmissione sicuro ma:
- ❑ La comunicazione è prevista punto a punto.
- ❑ I messaggi SOAP sono processati da un numero indefinito di nodi che possono essere su reti diverse che utilizzano protocolli diversi.
- ❑ Tutti i nodi dovrebbero essere affidabili!
- ❑ I canali di comunicazione tra i nodi devono essere affidabili.
- ❑ La sicurezza end-to-end deve essere portata a livello dell'applicazione ma non dentro l'applicazione.

Trasporto

Case I: All connections are via HTTP



Case II: Not all connections are via HTTP



Dati archiviati



- La sicurezza del canale preserva i dati durante la trasmissione ma non ha nessun effetto sui dati archiviati.

XML signature and processing



- Proposta W3C per integrare un layer di sicurezza nei web services implementati con SOAP.
- Utilizzo della firma digitale.
- Gli elementi da firmare vengono segnati con una chiave e resi figli di un elemento che viene dichiarato "firmato".
- Crittografia dei dati con algoritmo triple-DES.

SOAP e XML signature



- I messaggi SOAP sono divisi in header e body.
- Le informazioni di sicurezza potrebbero essere integrate nella sezione header.
- Le informazioni criptate potrebbero essere inserite nel SOAP body.
- Per l'autenticazione le proposte sono l'invio di un certificato.

SOAP Security Extensions: Digital Signature



- Specifiche per definire un SOAP header che trasporti informazioni relative alla firma digitale.
- Namespace:
<http://schemas.xmlsoap.org/soap/security/2000-12>
- L'elemento che definisce l'header è <SOAP-SEC:Signature>

XML Schema

```
<schema xmlns="http://www.w3.org/1999/XMLSchema"
  xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
  targetNamespace="http://schemas.xmlsoap.org/soap/security/2000-12"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <import namespace="http://www.w3.org/2000/09/xmldsig#" />
  <import namespace="http://schemas.xmlsoap.org/soap/envelope#" />
<element name="Signature" final="restriction">
  <complexType>
    <sequence>
      <element ref="ds:Signature" minOccurs="1" maxOccurs="1"/>
    </sequence>
    <attribute name="id" type="ID" use="optional"/>
    <attribute ref="SOAP-ENV:actor" use="optional"/>
    <attribute ref="SOAP-ENV:mustUnderstand" use="optional"/>
  </complexType>
</element>
<attribute name="id" type="ID"/>
</schema>
```

Esempio

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Header>
    <SOAP-SEC:Signature xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
      SOAP-ENV:actor="some-URI" SOAP-ENV:mustUnderstand="1">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026">
            </ds:CanonicalizationMethod>
        <ds:SignatureMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
        <ds:Reference URI="#Body">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
        </ds:Transforms>
```

Esempio (2)

```
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>j6lw3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>MC0CFFrVLtRlk=...</ds:SignatureValue>
</ds:Signature>
</SOAP-SEC:Signature>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:SOAP-
SEC="http://schemas.xmlsoap.org/soap/security/2000-12" SOAP-
SEC:id="Body">
    <m:GetLastTradePrice xmlns:m="some-URI">
        <m:symbol>IBM</m:symbol>
    </m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Regole di processing



- Quando un messaggio SOAP che integra una firma digitale viene processato un nodo deve:
 - Decidere se processarlo o meno.
 - Se lo processa, validare la firma.

WSDL



Introduzione al WSDL (Web Service Definition Language)

I Web Services



- Il concetto di **web service** è molto simile a quello di oggetto, nel significato con cui lo si usa nella Object-Oriented Programming:
 - Un oggetto è un **modulo software che offre una serie di funzioni utilizzabili dall'esterno** da parte di altro software, tramite una interfaccia di comunicazione dichiarata dall'oggetto stesso.
 - Anche un web service offre una funzionalità (servizio), ad altri client sulla rete attraverso una interfaccia ben definita.

I Web Services



- La differenza è che in questo caso il servizio è posto sul web, e **l'integrazione con il servizio avviene attraverso lo scambio di messaggi sulla rete.**
 - La definizione del servizio, ovvero della sua interfaccia, avviene tramite WSDL.
 - La “pubblicazione” del servizio, che lo rende visibile a chi, sulla rete, ne abbia bisogno, avviene tramite un altro linguaggio XML, l’UDDI (*Universal Description, Discovery and Integration*).

WSDL

- WSDL, presenta le caratteristiche architetturali di molti altri linguaggi di basati su XML nati di recente:
 - Il linguaggio fa riferimento e si integra con standard esistenti, evitando di ridefinire ciò che è già stato definito: WSDL predilige l'uso di **XMLSchema** per il type system e di **SOAP** per la definizione dei messaggi.
 - Il linguaggio è estendibile: l'uso SOAP e XMLSchema è consigliato ma non necessario.

Descrivere un Servizio Web



- ❑ WSDL si presenta molto genericamente:
un formato XML per descrivere servizi di rete come un insieme di punti terminali operanti su messaggi contenenti informazione di tipo "documentale" o "procedurale"
- ❑ A fronte di questa descrizione molto aperta, WSDL cerca di separare gli aspetti "astratti" della descrizione di un servizio da quelli "concreti" che lo legano a particolari protocolli di rete o di RPC.

Descrivere un Servizio Web

- I documenti WSDL descrivono un servizio su due livelli:
- **Livello astratto:** il servizio viene definito essenzialmente in base alle operazioni offerte (interfacce) e al tipo di messaggi scambiati per ciascuna di esse.
- **Livello concreto:** le descrizioni astratte vengono istanziate legandole a una implementazione reale.

Descrivere un Servizio Web



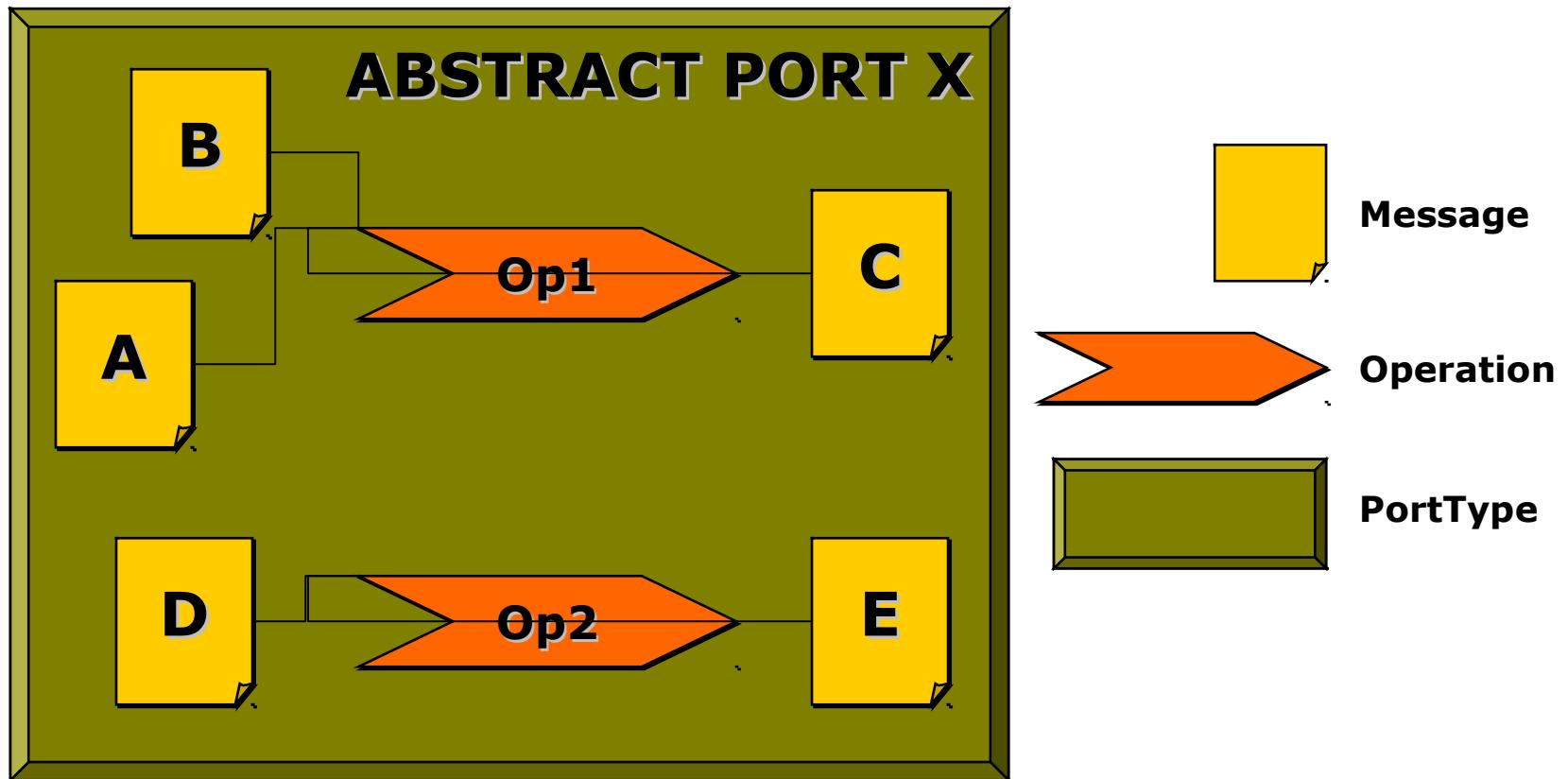
- I vantaggi della descrizione su due livelli sono ovvi:
 - Lo stesso servizio può avere **implementazioni diverse**, basandosi sulla stessa descrizione astratta.
 - Le descrizioni astratte possono essere **riutilizzate**, tutte o in parte, nella creazione di nuovi servizi.

Descrizione Astratta



- ❑ WSDL vede i servizi web come insieme di terminali di rete detti **Porte**.
- ❑ La descrizione astratta di un servizio web è composta:
 - Dalla definizione dei **messaggi** scambiati.
 - Dalla definizione delle **operazioni** inerenti il servizio, che ricevono e ritornano messaggi.
 - Dalla definizione dei **PortType**, ovvero di collezioni di operazioni (interfacce).

Descrizione Astratta

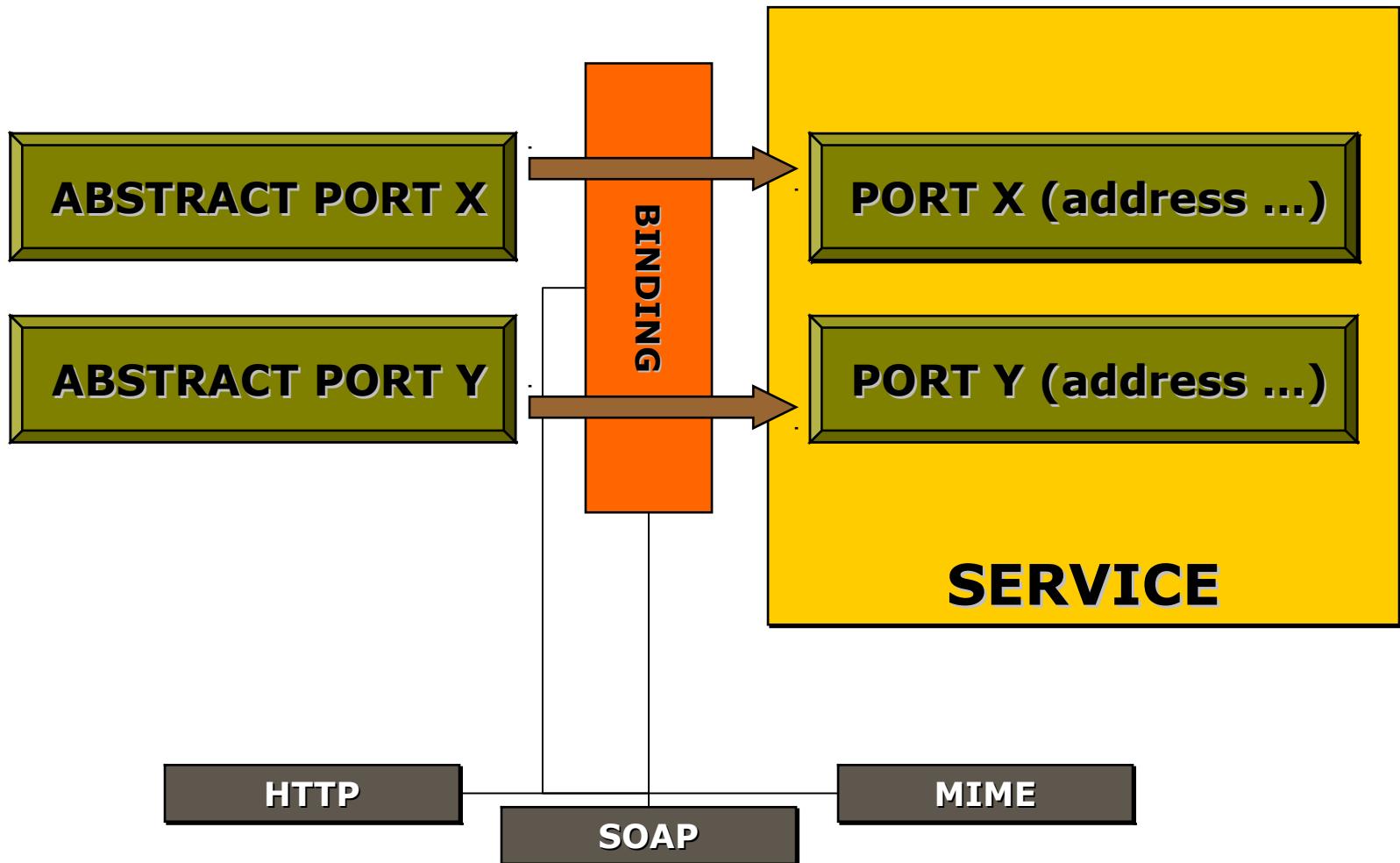


Descrizione Concreta



- La descrizione concreta di un servizio web è composta:
 - Dai **binding** che legano ciascun PortType a un protocollo e formato di dati per i messaggi scambiati,
 - Dalle **porte**, che legano i PortType istanziati dai binding a indirizzi di rete reali,
 - Dai **servizi**, composti da insiemi di porte correlate.

Descrizione Concreta



Documenti WSDL



- Ogni documento WSDL inizia con l'elemento <definitions>.
- Il namespace per gli elementi WSDL è <http://schemas.xmlsoap.org/wsdl/>.
- Altri namespaces da importare sono, di solito, quello di XML Schema e quello di SOAP.

Documenti WSDL

```
<definitions name="PlaceFinder"  
targetNamespace="http://www.geographynetwork.com/PlaceFinder"  
xmlns:tns="http://www.geographynetwork.com/PlaceFinder"  
xmlns:electric="http://www.themindelectric.com/"  
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"  
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"  
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"  
xmlns="http://schemas.xmlsoap.org/wsdl/">  
  
...  
  
</definitions>
```

Scheletro esterno di un documento WSDL.

Per gli esempi riguardanti WSDL useremo la definizione di un vero servizio web.

All'URL www.xmethods.com sono disponibili molti esempi di servizi funzionanti da studiare e provare.

Tipi nei documenti WSDL

- Ogni messaggio scambiato nell'ambito delle operazioni WSDL deve essere **tipato**.
 - Il **type system** preferito dal WSDL è quello di XML Schema, ma c'è anche la possibilità di usare dichiarazioni di tipi provenienti da altri namespaces.
- Per definire i tipi usati all'interno del documento, WSDL usa l'elemento <types>:
 - <types> è il primo figlio di <definitions>
 - all'interno di types può essere inserito un intero Schema, o qualsiasi altra definizione di tipo.

Tipi nei documenti WSDL

<types>

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="...">
    <complexType name="LocationInfo">
        <sequence>
            <element name="matchType" nillable="true" type="string"/>
            <element name="candidates" nillable="true" type="tns:ArrayOfLocation"/>
        </sequence>
    </complexType>
    <complexType name="Location">
        <sequence>
            <element name="x" type="double"/>
            <element name="y" type="double"/>
            <element name="description" nillable="true" type="string"/>
            <element name="score" type="double"/>
        </sequence>
    </complexType>
    <complexType name="ArrayOfLocation">
        <complexContent>
            <restriction base="soapenc:Array">
                <attribute ref="soapenc:arrayType" wsdl:arrayType="tns:Location[]"/>
            </restriction>
        </complexContent>
    </complexType>
</schema>
```

Definizione dei tipi.

Il contenuto dell'elemento <types> è un normale Schema XML.

Lo schema è stato semplificato rispetto all'originale disponibile su Internet.

Messaggi

- ❑ I messaggi sono il blocco di base nella costruzione di un servizio web con WSDL. Sono definiti in uno o più elementi <message> che seguono la sezione <types>.
- ❑ Ciascun messaggio ha un nome univoco ed è costituito da una o più parti, ciascuna con un nome e un tipo distinti.
 - Ad esempio, se un messaggio rappresenta la codifica di una RPC, le sue “parti” sono solitamente i vari argomenti della chiamata, o il valore di ritorno.

Messaggi

```
<definitions name="PlaceFinder">
  <types>
    ...
  </types>
  <message name="findPlace1SoapIn">
    <part name="arg0" type="xsd:string"/>
  </message>
  <message name="findPlace1SoapOut">
    <part name="Result" xmlns:ns1="..." type="ns1:LocationInfo"/>
  </message>
</definitions>
```

Definizione dei messaggi.

La definizione dei messaggi segue immediatamente quella dei tipi.

Ciascun elemento `<message>` contiene una o più `<parts>`, ognuna delle quali specifica una parte del messaggio con un determinato nome e tipo.

Nell'esempio, il messaggio **findPlace1SoapIn** ha una sola parte che si chiama **arg0** ed è di tipo `xsd:string` (stringa, tipo built-in degli schemi). Il messaggio `findPlace1SoapOut`, invece, ha una parte chiamata **Result** che contiene valori di un tipo definito nella precedente sezione `<types>`. Si tratta chiaramente della rappresentazione di una RPC. Notare che è sempre possibile incorporare tutti i parametri di una RPC in un solo tipo composto ed avere quindi messaggi composti da una sola parte.

Porte Astratte

- La descrizione astratta di una porta, intesa come **insieme di operazioni** è affidata agli elementi <PortType>.
- Ogni porta astratta ha un nome ed è composta da un insieme di elementi <operation> rappresentanti le operazioni associate.
- Gli elementi <operation> a loro volta contengono elementi <input> e <output> specificanti i messaggi scambiati durante l'operazione.

Operazioni



- WSDL definisce quattro tipi di operazioni:
 - **One-way**: il client spedisce un messaggio al servizio.
 - **Request-Response**: il client spedisce un messaggio al servizio e riceve una risposta .
 - **Notification**: il servizio invia un messaggio a un client.
 - **Solicit-Response**: il servizio invia un messaggio a un client e questo risponde.
- A queste operazioni corrispondono diverse combinazioni di messaggi in input e output.

Operazioni

```
<operation name="op1">
    <input name="op1Request" message="..."/>
    <output name="op1Response" message="..."/>
    <fault name="op1Fault" message="..."/>
</operation>

<operation name="op2">
    <output name="op2Solicit" message="..."/>
    <input name="op2Response" message="..."/>
    <fault name="op2Fault" message="..."/>
</operation>

<operation name="op3">
    <input name="op3in" message="..."/>
</operation>

<operation name="op4">
    <output name="op4out" message="..."/>
</operation>
```

Operazione Request/Response (op1).

Definiamo una `<operation>` di tipo request/response chiamata **op1**. Gli elementi `<input>` e `<output>` specificano il messaggio (definito in precedenza) associato rispettivamente all'operazione di request e response. L'elemento opzionale `<fault>` specifica il formato da usare per i messaggi di errore.

Operazione Solicit/Response (op2).

Esattamente come nelle operazioni Request/Response, ma l'ordine degli elementi `<input>` e `<output>` è invertito.

Operazioni One-Way (op3) e Notification (op4).

In questo caso è presente solo il messaggio di input (one-way) o di output (notification)

Porte Astratte

```
<portType name="PlaceFinderSoap">
    <operation name="findPlace">
        <input name="findPlace1SoapIn" message="tns:findPlace1SoapIn"/>
        <output name="findPlace1SoapOut"
message="tns:findPlace1SoapOut"/>
    </operation>
</portType>
```

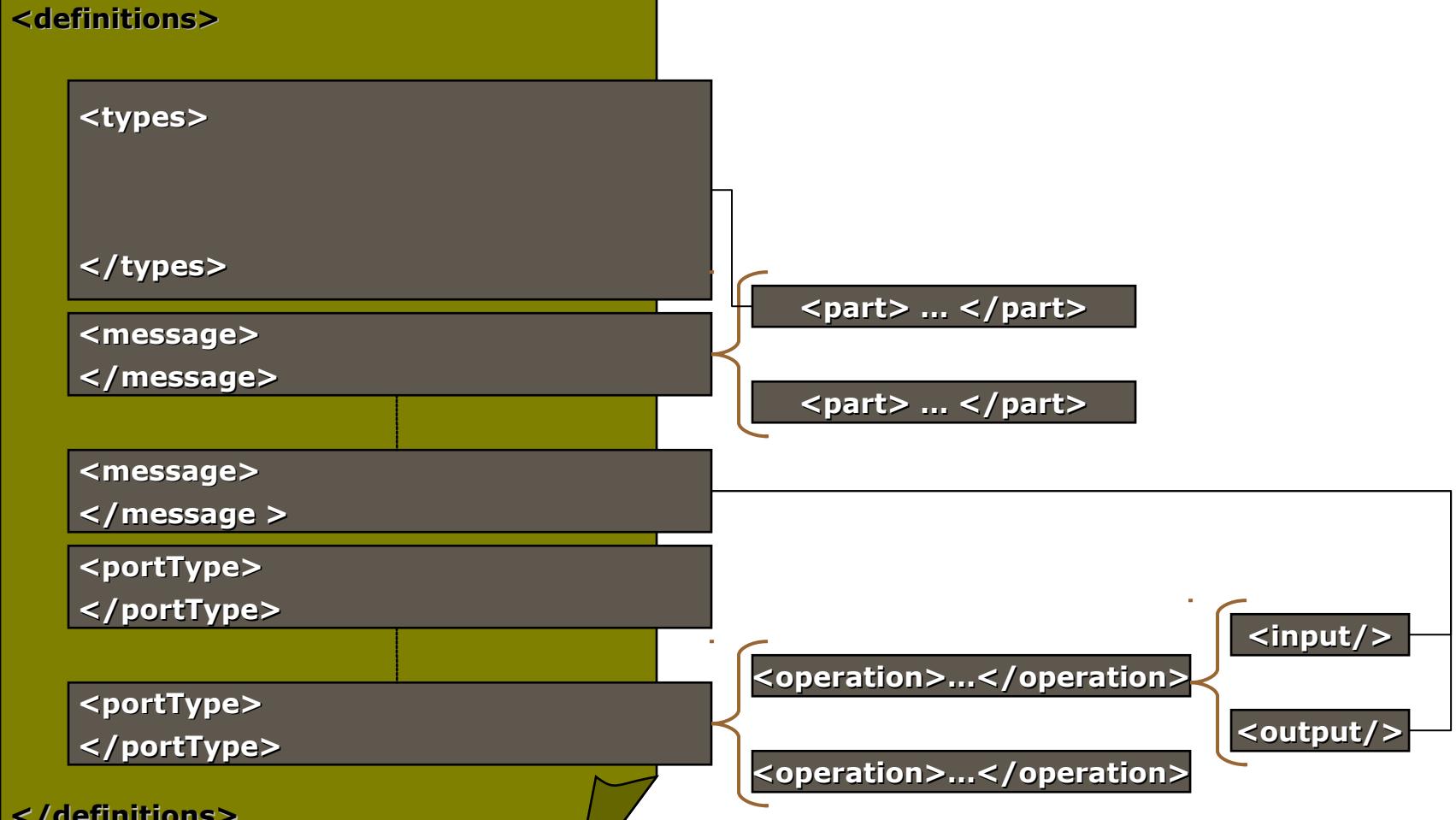
Definizione delle porte astratte.

La definizione delle porte astratte segue quella dei messaggi.

Il nostro servizio ha un solo tipo di porta, chiamata PlaceFinderSoap, che consente una sola operazione "findPlace".

Questa operazione è di tipo Request/Response: la richiesta da parte del client, specificata nell'elemento <input> e chiamata findPlace1SoapIn è effettuata trasmettendo il messaggio chiamato findPlace1SoapIn. La risposta da parte del servizio, specificata nell'elemento <output> e chiamata findPlace1SoapOut è effettuata trasmettendo il messaggio chiamato findPlace1SoapOut.

Riassunto della Definizione Astratta



Bindings

- Un binding è in pratica l'istanziazione di una porta astratta.
- L'intera definizione del <portType> associato viene ripetuta, ma al suo interno vengono aggiunti nuovi elementi che specificano come codificare le operazioni e i messaggi.
- WSDL definisce gli elementi per esprimere vari tipi di binding:
 - SOAP, HTTP, MIME
- **Noi descriveremo il binding SOAP.**

SOAP Binding



- Gli elementi SOAP contenuti nell'elemento <binding> servono a descrivere **come costruire il messaggio SOAP a partire dai messaggi astratti WSDL.**
 - <soap:binding>: specifica lo stile default di codifica da usare per tradurre le parti del messaggio WSDL in un messaggio SOAP (<Header> e <Body>), e il tipo di trasporto utilizzato da SOAP (es. HTTP).

SOAP Binding

```
<binding name="PlaceFinderSoap" type="tns:PlaceFinderSoap">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    ...
</binding>
```

Binding con SOAP, prima parte.

L'elemento `<binding>` dichiara l'istanziazione di una porta astratta. L'attributo `type` dell'elemento si riferisce a un `<portType>` dichiarato in precedenza. All'interno del binding, **l'intero tipo di porta verrà ripetuto e dettagliato** usando gli elementi del binding prescelto, in questo caso SOAP.

Il primo elemento che si incontra in un binding SOAP è `<soap:binding>`.

L'attributo `transport` verrà usato per specificare il tipo di trasporto per i messaggi SOAP: in questo caso useremo HTTP.

L'attributo `style` indica il modo in cui i messaggi WSDL devono essere mappati in messaggi SOAP. In questo caso sceglieremo lo stile RPC. Notare che l'attributo `style` può essere ridefinito in ogni singola `<operation>`.

Stili di Codifica

- Lo **stile di codifica**, specificato dall'attributo style di <soap:binding> e/o <soap:operation> può essere:
 - “rpc”: il messaggio WSDL viene codificato in SOAP come una RPC.
Viene creato un elemento uguale al nome dell’operazione, che sarà inserito nel <Body> del messaggio SOAP, e tanti figli quante sono le <part> del messaggio, ognuno con lo stesso nome della parte.

Stili di Codifica

- Il secondo stile di codifica specificabile dall'attributo style di <soap:binding> e/o <soap:operation> è
 - “document”: il messaggio WSDL viene codificato il SOAP usandone il contenuto “letteralmente”. In questo caso, cioè, ogni <part> del messaggio figurerà come figlio distinto dell’elemento <Body> nel messaggio SOAP.

Stili di Codifica

```
<message name="twoPartsOpIn">
    <part name="arg0" type="xsd:string"/>
    <part name="arg1" type="xsd:double"/>
</message>

<portType name="twoPartsPort">
    <operation name="twoPartsOp" parameterOrder="arg0 arg1">
        <input name="twoPartsOpIn"
message="tns:twoPartsOpIn"/>
    </operation>
</portType>

<binding name="SoapGeneric" type="tns:twoPartsPort">
    <soap:binding style="..." 
transport="http://schemas.xmlsoap.org/soap/http"/>
    ...
</binding>
```

style="rpc"

```
<SOAP-ENV:Body>
    <twoPartsOp>
        <arg0/>
        <arg1/>
    </twoPartsOp>
</SOAP-ENV:Body>
```

style="document"

```
<SOAP-ENV:Body>
    <arg0/>
    <arg1/>
</SOAP-ENV:Body>
```

Binding con SOAP, stile RPC e Document.

Vediamo qui un frammento generico di file WSDL. Viene definita una porta astratta "twoPartsPort" con una operazione one-way "twoPartsOp" abbinata a un messaggio "twoPartsOpIn" composto da due parti. Viene quindi definito un binding SOAP per la porta.

In basso, vediamo il corpo del messaggio SOAP corrispondente all'operazione "twoPartsOp" nei due stili di codifica possibili.

Binding delle Operazioni

- All'interno dell'elemento <binding> vengono ripetuti gli elementi <operation> della corrispondente porta astratta.
- Il primo figlio di ciascuna <operation> dovrà essere un elemento <soap:operation> che specifica:
 - **Lo stile di codifica dell'operazione**, solo se diverso da quello dichiarato in <soap:binding> (attributo style)
 - La **SOAPAction** associata, solo se il trasporto sarà HTTP (attributo soapAction)

Binding delle Operazioni

```
<binding name="PlaceFinderSoap" type="tns:PlaceFinderSoap">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="findPlace">
        <soap:operation soapAction="findPlace" style="rpc"/>
        ...
    </operation>
</binding>
```

Binding con SOAP, seconda parte.

All'interno dell'elemento `<binding>` cominciamo a ripetere gli elementi `<operation>` dichiarati dal `<portType>` corrispondente (`PlaceFinderSoap`).

All'interno di ciascuna `<operation>` inseriamo una `<soap:operation>` che specifica lo stile di codifica dell'operazione (che in questo caso, essendo identico a quello specificato globalmente nel `<soap:binding>`, potrebbe essere omesso) e la `SOAPAction`, necessaria perché il trasporto del messaggio è HTTP.

Binding degli input/output

- Gli elementi <input> e/o <output> che compongono le <operation> all'interno del binding non devono ripetere il loro nome né messaggio associato.
- Devono invece contenere **elementi che specificano ulteriormente come verrà costruito il corrispondente messaggio SOAP** (<Header> e <Body>).

Binding degli input/output: Body

- Per costruire il corpo del messaggio SOAP si usa l'elemento <soap:body> specificando:
 - Le **parti del messaggio che saranno incluse** nel corpo (attributo parts).
 - La **codifica dei tipi** interessati (attributi use e encodingStyle).
 - ◆ Use="encoded" richiede la codifica di ogni parte in base al suo tipo secondo le regole di encodingStyle.
 - ◆ Use="literal" indica che le parti vanno copiate "letteralmente" nel messaggio.
 - Il namespace di provenienza degli elementi usati nel messaggio (attributo namespace).

Binding degli input/output: Body

```
<binding name="PlaceFinderSoap" type="tns:PlaceFinderSoap">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="findPlace">
        <soap:operation soapAction="findPlace" style="rpc">
        <input name="findPlace1SoapIn">
            <soap:body use="encoded" namespace="..." />
        <encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </input>
        <output name="findPlace1SoapOut">
            <soap:body use="encoded" namespace="..." />
        <encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
        </output>
    </operation>
</binding>
```

Binding con SOAP, terza parte.

All'interno di ciascun elemento `<operation>` ripetiamo gli stessi elementi `<input>` e/o `<output>` del `<portType>` associato, questa volta senza specificare il messaggio (anche la ripetizione del nome è opzionale).

L'elemento `<soap:body>`, nidificato negli elementi `<input>` e `<output>`, definisce il `<Body>` del messaggio SOAP.

Viene indicato il namespace di appartenenza degli elementi usati, e viene specificato che le parti del messaggio andranno codificate (`use="encoded"`) usando la codifica standard SOAP/Schema (`encodingStyle`).

L'attributo parts non è presente, quindi tutte le parti del messaggio saranno inserite nel corpo SOAP.

Binding degli input/output: Header

- Opzionalmente, è possibile **fornire anche un <Header> al messaggio SOAP.**
- Ciascuna header entry è ottenuta con l'elemento <soap:header> specificando:
 - Il messaggio e la sua parte che diventerà una header entry (attributi message e part).
 - ◊ Gli elementi della parte interessata possono includere gli attributi actor e mustUnderstand di SOAP, ma solo se use="literal".
 - La codifica dei tipi (use e encodingStyle) e il namespace degli elementi (attributo namespace).

Binding degli input/output: Fault

- Opzionalmente, è possibile **specificare i messaggi che saranno restituiti in caso di errore** nell'elaborazione del <Body> o dell'<Header>.
 - <soap:fault>, ha la stessa sintassi di <soap:body> e può essere incluso negli elementi <fault> associati alle <operation>.
 - <soap:headerfault> ha la stessa sintassi di <soap:header> e può essere incluso negli elementi <soap:header> stessi.

Binding degli input/output: Fault

```
<operation name="twoPartsOp">
    <soap:operation soapAction="http://dellapenna.univaq.it/wsdlExample#action1"/>
    <input>
        <soap:body use="literal" namespace="http://dellapenna.univaq.it/wsdlExample"/>
    </input>
    <fault>
        <soap:fault name="twoPartsFault" use="literal" namespace="..."/>
    </fault>
    <soap:header message="twoPartsOpHdr" part="h1" use="literal" namespace="..."/>
    <soap:header message="twoPartsOpHdr" part="h2" use="literal" namespace="..."/>
        <soap:headerfault message="twoPartsOpHdr" part="h2e" use="literal"
namespace="..."/>
    </soap:header>
</operation>
```

Binding con SOAP, <header>, <fault> e <headerfault>.

Questo frammento di binding per una operazione mostra come è possibile definire la costruzione dei messaggi di errore per il corpo e per le header entries del messaggio SOAP.

Vengono anche definite due header entries, ognuna a partire da una parte dello stesso messaggio "twoPartsOpHdr".

Porte



- **Una porta è l'istanza di una porta astratta** (`<portType>`) ottenuta con un `<binding>`.
- Le porte sono specificate con elementi `<port>`:
 - Ogni `<port>` ha un nome unico (attributo `name`).
 - Ogni `<port>` fa riferimento al nome di un `<binding>` (attributo `binding`).
 - Se il binding è con SOAP, l'elemento `<port>` deve contenere un elemento `<soap:address>` che specifichi l'indirizzo di rete della porta (attributo `location`)

Servizi



- **Le porte non sono dichiarate globalmente, ma all'interno di servizi.**

- <service> è l'elemento di livello più alto in WSDL. Esso dichiara un servizio web con un particolare nome come raccolta di porte (concrete).

Servizi

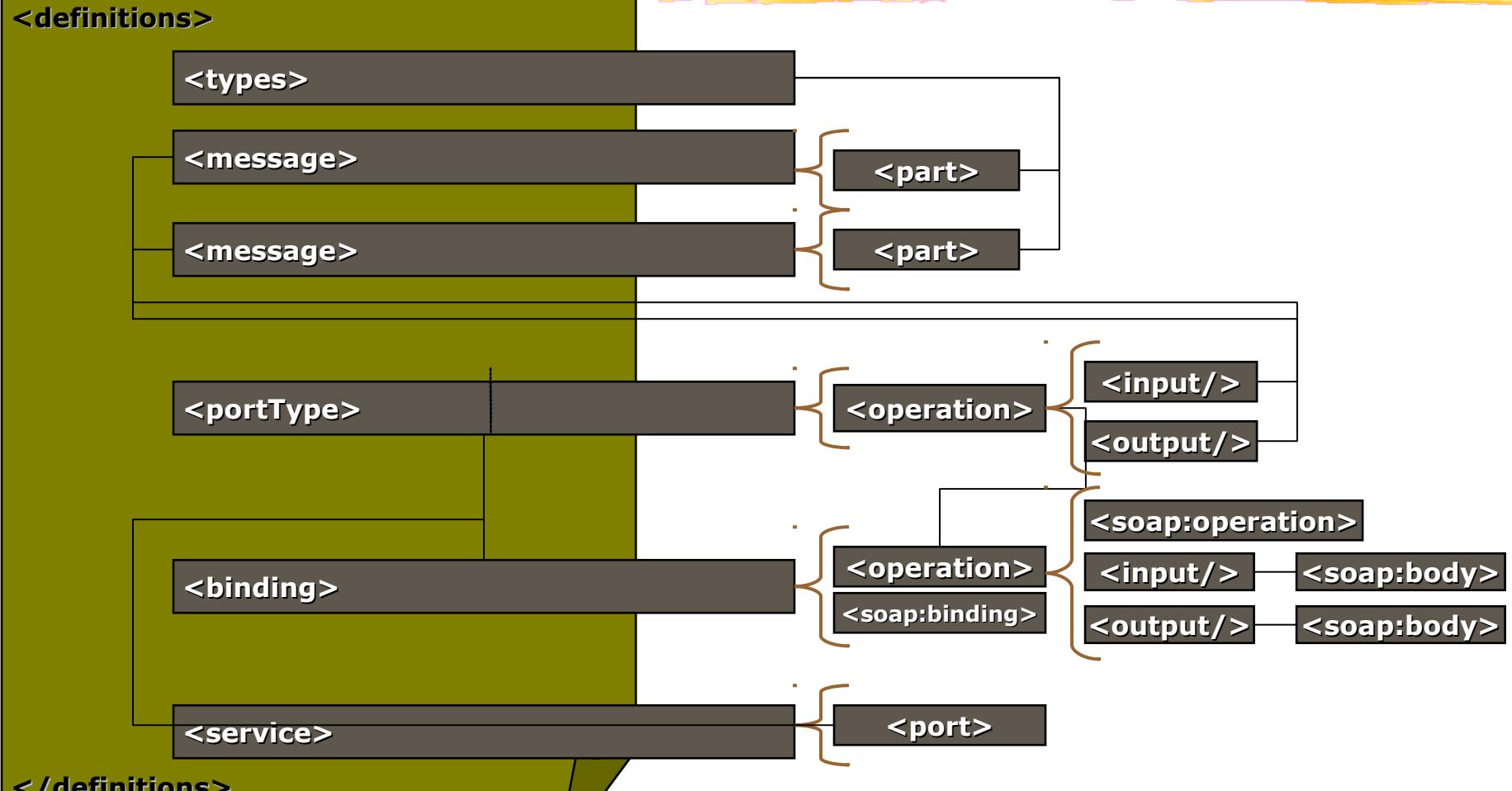
```
<service name="PlaceFinder">
    <port name="PlaceFinderSoap" binding="tns:PlaceFinderSoap">
        <soap:address
location="http://www.geographynetwork.com/geoservices/v1/PlaceFinder"/>
    </port>
</service>
```

Binding con SOAP, quarta parte.

Finalmente viene dichiarato il servizio web che si intende fornire, chiamato “PlaceFinder”. Il servizio contiene una porta “PlaceFinderSoap” ottenuta dall’omonimo binding di una porta astratta con SOAP.

L’elemento `<soap:address>` specifica, tramite l’attributo `location`, l’indirizzo internet della porta per il servizio (l’indirizzo è di tipo web perché il trasporto dichiarato per SOAP nel binding è HTTP).

Riassunto della Definizione WSDL



</definitions>

Riferimenti



- ❑ **Specifiche di WSDL dal W3C**

<http://www.W3.org/TR/wsdl>

- ❑ **Directory di Servizi WSDL da Provare**

<http://www.xmethods.net/>

UDDI

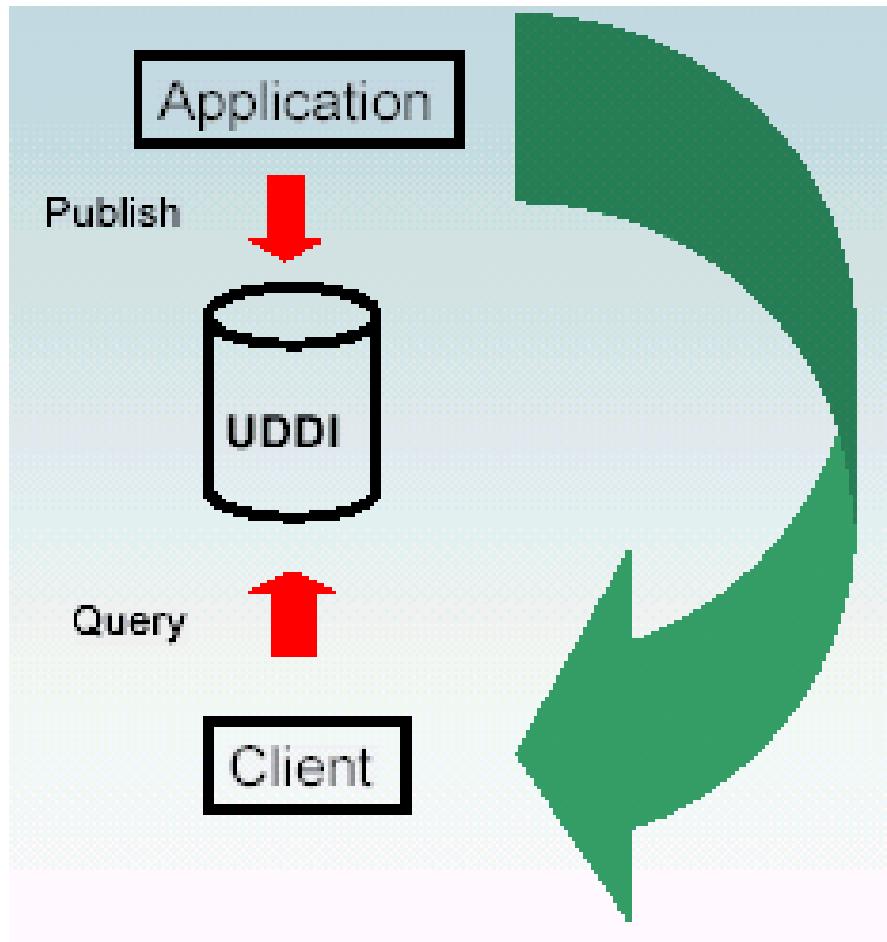


- UDDI (Universal Description, Discovery, Integration Protocol) è un servizio di directory che permette ai web services client di localizzare i web services basato su XML.
- UDDI può essere considerato come una specie di “pagine gialle” dei web services.
- Senza UDDI due applicazioni potrebbero comunicare solo se già si conoscessero, conoscessero i servizi offerti e la loro localizzazione.
- E’ necessario un “archivio” per permettere ai web services di rendere pubblica loro presenza e per renderli raggiungibili dagli utenti, siano essi utenti business o consumer.

UDDI (2)

- E' utilizzato da due classi di utenti:
 - Publisher, compagnia che offre web services
 - Client, utente o compagnia che ricerca un web service
- UDDI è molto simile a DNS (Domain Name Service) che contiene gli indirizzi IP di tutti gli host Internet.
- La differenza è che DNS lavora a un livello più basso, perché risolve indirizzi IP, mentre UDDI lavora a livello più alto poiché risolve servizi.
- UDDI è un servizio globale condiviso tra server differenti sparsi in tutto il mondo, anche se non organizzati secondo una struttura ad albero.
- I diversi server condividono i dati mediante un protocollo di replicazione
- UDDI si basa su SOAP per la trasmissione dei messaggi.

Utilizzo di UDDI



UDDI – scenario



- Un'impresa necessita un servizio
- Localizza il servizio tramite UDDI
- Un programmatore contatta il server UDDI per avere dettagli tecnici
- Viene elaborata un'applicazione per accedere al servizio.

UDDI – cancellazione



- Un publisher può decidere di eliminare il proprio servizio da UDDI inviando una richiesta di cancellazione.
- Problema! Un concorrente potrebbe cancellare un servizio di un'altra impresa.
- Soluzione: Autenticazione dei publisher.
- Ogni server mantiene traccia dei publisher e di cosa hanno pubblicato.
- Solo chi ha pubblicato un servizio è autorizzato a cancellarlo o modificarlo.

UDDI – ruoli



- Analogia con elenchi del telefono:
 - Elenco (pagine bianche), contengono informazioni sui contatti e gli indirizzi dei publisher
 - Pagine gialle, contengono informazioni sui diversi servizi disponibili organizzati per categorie di business, per tipo di servizi etc.
 - Pagine verdi, contengono informazioni sul servizio stesso, cioè la descrizione tecnica delle interfacce del servizio e del punto di accesso (URL, e-mail, ecc.) utilizzando opportuni “tModel” (possono eventualmente anche contenere il codice WSDL del servizio).
 - tModel, Definizione tecnica di un tipo di servizio tipicamente emessa da enti di standardizzazione di specifici domini (es. EAN/UCC*) per essere utilizzata dai service provider.

ID



- Nei registri UDDI ogni elemento catalogato ha un ID univoco che serve come chiave per il dato.

businessEntity



- L'elemento businessEntity mantiene dati relativi ai contatti e agli indirizzi di un publisher.
- Ha come figli:
 - description, per inserire una descrizione
 - categorybag, per assegnare l'elemento a una categoria.
 - name, per nominare l'elemento
- UDDI però non può prevedere tutti i dati che un publisher vorrà pubblicare quindi mette a disposizione degli elementi identifierBag che possono essere utilizzati per dati aggiuntivi.
- Ha come attributo obbligatorio businessKey.

BusinessService



- Elemento che descrive uno specifico servizio offerto da un publisher.
- Ha come figli:
 - description, per inserire una descrizione
 - categorybag, per assegnare l'elemento a una categoria.
 - name, per nominare l'elemento
- Ha come attributo obbligatorio serviceKey.

BindingTemplate



- Elemento che fornisce le informazioni tecniche utilizzabili per localizzare un determinato servizio.
- Ha come figli:
 - Description, per descrivere il servizio
 - Accesspoint|hostingRedirector, per identificare le modalità di accesso al servizio.
- Ha come attributo obbligatorio bindingKey.

Tassonomie



- Esistono molte tessonomie per catalogare i servizi e le imprese in modi diversi.
- Esempi:
 - NAICS (North American Industry Classification System) tessonomia ufficiale del governo americano.
 - UNSPSC (United Nation Standard Product and Service Classification Code)

UDDI API



- Le API UDDI forniscono ai programmatori un modo semplice per interagire con i dati archiviati in un registro UDDI.
- Le API UDDI consistono di tre parti principali:
 - Inquiry, per permettere agli utenti di effettuare ricerche all'interno del registro.
 - Publishing, per permettere ai publisher di inserire, modificare, cancellare informazioni rispetto ai propri servizi
 - Replication, riservate alla comunicazione server-to-server per la gestione dei registri.

Inquiry



- Esistono due tipi principali di ricerche nei registri UDDI:
 - Find, per ricercare un servizi o un publisher
 - Detail, per ottenere maggiori informazioni su un servizio o publisher.
- E' possibile ricercare per:
 - Nome
 - discoveryURLs
 - Categoria
 - Etc.

Inquiry (2)



- Per effettuare ricerche semplici si effettua una richiesta SOAP utilizzando <find_business>.
- Per effettuare una ricerca per correlazione si utilizza una richiesta di tipo <find_relatedbusiness>
- Per effettuare una ricerca basata sulle caratteristiche tecniche supportate dal servizio si utilizza l'elemento <find_binding>
- Per trovare i servizi che un determinato publisher offre si utilizza l'elemento <find_service>

Inquiry: getting details

- Quando si conosce ciò che si cerca può invece essere necessario ottenere maggiori dettagli.
- Per fare questo sono disponibili diversi elementi.
 - <get_bindingDetail> si basa su una bindingKey e restituisce tutti i dettagli tecnici del servizio
 - <get_businessDetail> si basa su una businessKey e restituisce tutti i dettagli sul publisher.
 - <get_service_detail> si basa su una serviceKey e restituisce tutti i dettagli sul servizio.

Publishing



- Solo utenti autorizzati possono pubblicare dei servizi o dei dati in un registro UDDI, mentre tutti possono consultarlo.
- Un publisher deve ottenere un token di autenticazione dal server UDDI prima di poter pubblicare.
- Esiste un elemento <get_authToken> che prende in input lo user ID e una password e restituisce un token di autenticazione

Salvataggio dei dati



- Per salvare i dati esiste uno specifico elemento <save_business> che prende in input le credenziali del publisher e i dati da salvare.
- Per salvare i dati esiste uno specifico elemento <save_services> che prende in input le credenziali del publisher e i dati del servizio da salvare.
- Per salvare i dati esiste uno specifico elemento <save_binding> che prende in input le credenziali del publisher e i dati del binding da salvare.

Cancellazione



- Per cancellare i dati esiste uno specifico elemento <delete_business> che prende in input le credenziali del publisher e i dati del publisher da eliminare.
- Per cancellare i dati esiste uno specifico elemento <delete_services> che prende in input le credenziali del publisher e i dati del servizio da eliminare.
- Per cancellare i dati esiste uno specifico elemento <delete_binding> che prende in input le credenziali del publisher e i dati del binding da eliminare.

Relazioni



- Per mettere in relazione due publisher è necessario che entrambi i publisher asseriscano la relazione all'interno del registro UDDI.
- Questo perché un'azienda potrebbe non desiderare essere messa in relazione a un concorrente.
- Per aggiungere una relazione si utilizza l'elemento `<add_publisherAssertion>` o `<set_publisherAssertion>`.
- Per eliminare la relazione si utilizza `<delete_publisherAssertion>`.

Replication



- La replicazione è un meccanismo tramite il quale i server che mantengono i registri UDDI occasionalmente si scambiano messaggi per sincronizzare i loro dati.

SVILUPPO DI APPLICAZIONI SOAP in LINUX



- REQUISITI:
- Per sviluppare e gestire servizi tramite SOAP in ambiente Linux si deve avere:
 - LATO SERVER
 - ✧ Un server web quale Apache
 - ✧ L'ambiente di sviluppo Java (J2sdk) contenente la JRE
 - ✧ Un container per le applicazioni web Java come Tomcat
 - ✧ Il pacchetto di librerie e strumenti SOAP
 - ✧ Alcune librerie Java quali:
 - *mail.jar*,
 - *activation.jar*,
 - *xerces.jar*
 - LATO CLIENT
 - ✧ L'ambiente di sviluppo J2sdk per Java con il JRE
 - ✧ Le classi SOAP per la compilazione e l'esecuzione: *soap.jar*
 - ✧ Le altre classi sfruttate alle librerie SOAP:
 - *mail.jar*,
 - *activation.jar*,
 - *xerces.jar*

SOFTWARE LATO SERVER(Ambiente Linux)

- ❑ Per prima cosa si deve installare un server web quale nel nostro caso APACHE, che è presente oggi nella maggior parte delle distribuzioni Linux anche sotto forma di pacchetto(i) .rpm.
- ❑ Installato Apache troveremo la directory `/var/www`, che conterrà tutti i file relativi ai contenuti web e alla configurazione del server-web, cosa che esula dal nostro contesto.
- ❑ Da notare che per poter funzionare il server-web richiede che sia configurata una interfaccia di rete, nei PC non predisposti di scheda ethernet basta attivare l'interfaccia di loopback (lo) e si accederà ai contenuti web attraverso l'indirizzo `http://localhost/` dove se tutto funziona sarà possibile vedere sul browser una pagina di test di Apache.
- ❑ Ovviamente bisognerà, se i servizi non partono automaticamente, provvedere a lanciare sia il demone `httpd` (il cuore di Apache) che attivare l'interfaccia di rete con i seguenti comandi (impartiti da root):
 - **\$ service network start**
 - **\$ service httpd start**

SOFTWARE LATO SERVER(Ambiente Linux)

- Visto che ci riferiamo all'utilizzo di SOAP in ambiente Java dobbiamo scaricare ed installare il JSdk, ossia il tool di sviluppo per Java che contiene sia il compilatore (javac) sia il JRE (la virtual machine che rende possibile eseguire attraverso il comando **java** i file *.class* che contengono il bytecode Java anche nel contesto delle applicazioni web se attivata da un container come Tomcat).
- Scarichiamo dal sito della Sun il file *J2sdk_1_4_1_01-linuxi586.bin* che è un file eseguibile, lanciamolo da shell e seguiamo le istruzioni date indicando un path dove installare il software, ad esempio */usr/local/*; finita la procedura di installazione avremo una nuova directory */usr/local/J2sdk_1_4_1_01...*, rinominiamola in j2sdk per semplicità
 - **\$ mv /usr/local/J2sdk_1_4_1_01... /usr/local/j2sdk**
- La versione 1.4 del jsdk non necessita di specificare alcun CLASSPATH per funzionare, ma occorre (per non doverlo immettere ogni volta) aggiungere al proprio PATH */usr/local/j2sdk/bin*, luogo dove risiedono gli eseguibili java, e */usr/local/j2sdk/jre/bin* dove si trovano i binari della javavm.

SOFTWARE LATO SERVER(Ambiente Linux)

- ❑ Risulta comodo inserire queste modifiche alle variabili d'ambiente nel file di configurazione della nostra shell ad esempio in `~/.bash_profile` o (rendendo le modifiche valide per tutti gli utenti) in `/etc/profile` aggiungiamo le seguenti righe:
 - **PATH=\$PATH:**
\$HOME/bin.:./usr/local/j2sdk/bin:/usr/local/j2sdk/jre/bin
 - **export PATH**
- ❑ Adesso dobbiamo provvedere a scaricare dalla rete Tomcat, ossia il container per le applicazioni web scritte in java che permette al server web di eseguire codice Java come ad esempio delle servlet JSP o comuni classi in java come useremo nel nostro caso.
- ❑ Tomcat è reperibile all'url www.jakarta.apache.org: dove potremo scaricare il file `tomcat-4.1.17.tar.gz` a questo punto lo scompattiamo ponendoci nella directory `/var/www`:
 - **\$ cd /var/www**
 - **\$ tar xvfz /path-to-tomcat archive/tomcat-4.1.17.tar.gz**

SOFTWARE LATO SERVER(Ambiente Linux)

- Abbiamo ora una nuova directory /var/www/tomcat/ nella quale c'è il motore CATALINA (ossia tomcat 4.x.y) e dobbiamo configurare la variabile d'ambiente \$JAVA_HOME e \$CATALINA_HOME, la prima conterrà il path del j2sdk in modo che tomcat sappia dove trovare tutto l'indispensabile per poter eseguire il bytecode Java, la seconda invece contiene il path della directory di installazione di tomcat in modo che tomcat sappia dove trovare il suo ambiente esecutivo e soprattutto la directory webbapps nella quale saranno poste le applicazioni web da gestire.
- Al solito modificheremo il file ~/.bash_profile o da root, come per le variabili java nel file /etc/profile
 - **JAVA_HOME='/usr/local/j2sdk'**
 - **CATALINA_HOME='/var/www/tomcat'**
 - **export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC JAVA_HOME CATALINA_HOME**

SOFTWARE LATO SERVER(Ambiente Linux)

- A questo punto, dopo aver avviato l'interfaccia di rete e il demone httpd (se non lo abbiamo fatto prima) come visto, avviamo tomcat con il comando:
 - **\$ \$CATALINA_HOME/bin/startup.sh**
- (a seconda della configurazione potrebbe essere necessario lanciare e fermare tomcat da robot; per fermare Tomcat basta digitare il comando:
\$ \$CATALINA_HOME/bin/shutdown.sh)
- Puntando adesso il browser all'indirizzo <http://localhost:8080> dovremmo trovarci nella pagina principale di Tomcat dalla quale possiamo accedere ad alcuni esempi di JSP, applet ecc..
- Passiamo ora all'installazione di SOAP; procuriamoci il file, scompattiamolo in una directory qualsiasi (serviranno solo alcuni file contenuti nel tar.gz),
 - **\$ tar xvfz soap-bin2.3.1.tar.gz**
- a questo punto verrà creata una directory `./soap/` (in seguito `<soap-home>`), prendiamo il file `./soap/webbapps/soap.war` (contiene le classi dell'Apache-SOAP web-administration tool, ossia lo strumento per l'amministrazione via web di SOAP) e copiamolo nella directory `$CATALINA_HOME/webbapps/(/var/www/tomcat/webapps)`.

SOFTWARE LATO SERVER(Ambiente Linux)

- "Stoppiamo" e riavviamo Tomcat quindi noteremo che il file soap.war è stato scompattato creando una directory \$CATALINA_HOME/webapps/soap; adesso copiamo il file <soap-home>/lib/soap.jar e poniamolo in \$CATALINA_HOME/common/lib (potrebbe essere necessario metterlo in /var/www/tomcat per risolvere alcuni conflitti), nella stessa directory copiamo anche i file:
 - activation.jar (scaricabile dal sito della Sun, contiene le classi del Java activation Framework)
 - mail.jar (sempre dal sito della Sun, contiene le classi per la gestione dei protocolli di scambio dei messaggi via SMTP ed altri protocolli)
 - xerces.jar (è un parser XML più evoluto di quello standard, andrebbe creato un PATH apposito per evitare conflitti con altri parser)

NOTA: questi package sono tutti contenuti in un archivio reperibile all'url
<http://www.xmethods.com/download/quickstart/quickstart.tar.gz>

SOFTWARE LATO SERVER(Ambiente Linux)

- Adesso dobbiamo impostare il CLASSPATH del nostro sistema in modo che le applicazioni java possano accedere a queste librerie, al solito possiamo modificare il file `~/.bash_profile` impostando le linee:
 - **CLASSPATH=.:./var/www/tomcat/webapps/soap/WEB-INF/classes:/var/www/tomcat/common/lib/xerces.jar:/var/www/tomcat/common/lib/mail.jar:/var/www/tomcat/common/lib/activation.jar:/var/www/tomcat/soap.jar**
 - **export CLASSPATH**
- per rendere effettive queste modifiche dobbiamo ovviamente fare un nuovo login in modo che il file di configurazione della shell venga riletto, oppure eseguiamo il comando
 - **\$ source ~/.bash_profile**

SOFTWARE LATO SERVER(Ambiente Linux)

- L'ambiente è stato così predisposto, avviamo Tomcat con il solito
 - **\$CATALINA_HOME/bin/startup.sh**
- e puntiamo il browser all'indirizzo
`http://localhost:8080/soap/servlet/rpcrouter`, dovremmo veder visualizzato un messaggio di questo tipo:
 - SOAP RPC Router
 - Sorry, I don't speak via HTTP GET- you have to use HTTP POST to talk to me.
- Se invece puntiamo direttamente all'indirizzo
`http://localhost:8080/soap/admin` accediamo al tool di amministrazione web di SOAP con il quale possiamo listare, fare il deploy e l'undeploy dei servizi pubblicati sul nostro sever SOAP.

CONFIGURAZIONE DELL'AMBIENTE DI SVILUPPO LATO CLIENT

- Per sviluppare dei servizi SOAP-Java dobbiamo fare riferimento alle classi contenute nei vari package utilizzati anche per il lato server, quali soap.jar, mail.jar, activation.jar e xerces.jar, è quindi indispensabile che il nostro CLASSPATH contenga il percorso a questi file, ciò è già stato fatto se la macchina sulla quale sviluppiamo i servizi è la stessa che contiene il server Apache-SOAP, se il sistema è un'altro (od anche un utente diverso) bisognerà come in precedenza aggiornare il file di configurazione della propria shell.

SVILUPPO DI UN SERVIZIO SOAP-RPC



- Un servizio SOAP-rpc si compone di due parti, un programma server che risiederà nell'ambiente del server SOAP e un programma client che invece potrà risiedere in ogni macchina che abbia accesso via rete (o tramile la loopback-interface lo0: 127.0.0.1) al server SOAP.

CREARE IL SERVER SOAP-RPC

- La scrittura dell'applicazione lato server in java, grazie alla flessibilità di SOAP risulta molto semplice, in quanto basta scrivere il codice di una classe che contenga il o i metodi che verranno poi esportati e rappresenteranno il `ilserviziol` messo a disposizione, senza preoccuparsi di dover gestire delle connessioni con il client e senza preoccuparsi di come gestire l'I/O, a tutto penserà SOAP!

CREARE IL SERVER SOAP-RPC

- Vediamo come esempio alcuni tratti di un progetto di esame:

```
import java.io.*;  
public class pren_server{  
    public static String prenota(String sj){  
        int[] array = new int[21];  
        int j = Integer.parseInt(sj);  
        //inizializza il codice di errore di sovrapprenotazione  
        array = readDataFromFile();  
        if (array[j] == 0){  
            //aggiorna la prenotazione e la scrive su file  
            array[j] = 1;  
            writeDataToFile(array);  
            return "OK PRENOTAZIONE EFFETTUATA";  
        } else return "ERRORE GIORNO OCCUPATO!!!";  
    }  
}
```

CREARE IL SERVER SOAP-RPC

- Al di là di quello che esegue questo metodo vogliamo puntualizzare solo alcune tracce per lo sviluppo di applicazioni SOAP-RPC based:
 - basta creare una classe pubblica che contenga i metodi che verranno poi effettivamente eseguiti, o staticamente, o come istanze
 - ogni metodo avrà o meno dei dati in ingresso provenienti dal client che sono semplicemente i parametri specificati nella dichiarazione del metodo (in questo caso String sj) ci saranno poi le linee di codice relative all'implementazione del metodo
 - i dati da rimandare al client sono proprio quelli restituiti dal metodo stesso e specificati dal return (è anche possibile gestire parametri di I/O passati al metodo)
- Una volta scritto e compilato il codice del server (per compilare usare: javac pren_server.java), dobbiamo creare un Deployment Descriptor per il servizio, ossia un file XML attraverso il quale comunicheremo a SOAP che servizio intendiamo fornire, quale è la classe che contiene il codice, quali sono i metodi che vogliamo esportare da quella classe, il tipo di applicazione ed altre informazioni.

CREARE IL SERVER SOAP-RPC

- Vediamo il DD per il nostro servizio, ossia il file prenotazioni.xml

```
<dd:service xmlns:dd="http://xml.apache.org/xml-soap/deployment"  
id="urn:prenotazioni">
```

```
    <dd:provider type="java"  
scope="Application"  
methods="getFree prenota">  
    <dd:java class="pren_server" static="false" />  
    </dd:provider>
```

```
<dd:faultListener>org.apache.soap.server.DOMFaultListener</dd:faultListener>  
    <dd:mappings />  
</dd:service>
```

- Il file si apre e si chiude con il tag <dd:sevice>, xmlns:=..... indica il namespace per il tipo di file XML del deployment descriptor SOAP ed id=Iurn:prenotazioniI, indica il nome del servizio che forniremo.

CREARE IL SERVER SOAP-RPC

- Nel tag <dd:provider> sono inclusi alcuni argomenti:
 - type=java indica il tipo di applicazione, ossia una classe java
 - scope=lu lè può assumere tre valori che indicano lo scope (visibilità) del metodo: 'Application' indica che l'oggetto rimarrà in vita (memoria) finché la servlet che ha erogato il servizio non è terminata; 'Request' indica che l'oggetto verrà cancellato appena finita la richiesta ed infine 'Session' impone che l'oggetto rimanga in vita per tutta la sessione HTTP
 - methods= ie lè la lista dei nomi dei metodi esportati, separati da uno spazio se sono più di uno
 - java class= il lè specifica il nome della classe contenente il codice del servizio
- Il tag <dd:faultListener> contiene l'indicazione del fault listener che si incaricherà di intercettare gli errori ecc..

PUBBLICARE IL SERVIZIO

- Adesso dobbiamo preoccuparci di rendere operativo il servizio e di registrarlo presso il server SOAP, in particolar modo poniamo una copia del file server-program.class (pren_server.class) in una directory che sia accessibile al class-container del nostro server web (Tomcat), ossia una directory che sia linkata nel CLASSPATH dell'ambiente server, possiamo a tal proposito mettere la nostra classe in `$CATALINA_HOME/webapps/soap/WEB-INF/classes/` oppure creare una directory specifica per i nostri servizi ed includerla nel CLASSPATH.
- Da ricordare che se la nostra classe fa parte di un package ad esempio del package 'sport-services', dobbiamo creare una directory 'sport-services' in un luogo indicato nel CLASSPATH e porvi dentro *pren_server.class*.
- Una volta sistemato il bytecode dobbiamo informare SOAP che vogliamo fornire un nuovo servizio, per fare ciò usiamo il comando:
 - **\$ java org.apache.soap.server.ServiceManagerClient \http://localhost:8080/soap/servlet/rpcrouter deploy prenotazioni.xml**
- SOAP leggendo il descrittore *prenotazioni.xml* si predisporrà per ricevere eventuali richieste di eseguire uno dei metodi esportati leggendone il bytecode dal file *pren_server.class* che abbiamo copiato prima e gestirà tutto l'I/O.

CREARE IL CLIENT SOAP-RPC

- La scrittura del codice del client richiede invece maggiori attenzioni, per prima cosa dobbiamo importare le librerie di SOAP oltre a quelle necessarie per il progetto (ecco la necessità di configurare un opportuno CLASSPATH per l'ambiente di sviluppo):

```
import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;
```

- Quando dobbiamo invocare un metodo remoto implementato attraverso il SOAP-RPC, creiamo un oggetto della classe Call ed attraverso i suoi metodi impostiamo tutti i parametri relativi alla chiamata da eseguire ed all'ambiente SOAP per la trasmissione e la codifica XML:

CREARE IL CLIENT SOAP-RPC

```
Call call = new Call();
call.setTargetObjectURI("urn:prenotazioni");
call.setMethodName("prenota");
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
Vector params = new Vector();
params.addElement(new Parameter("sj",String.class,sj,null));
call.setParams(params);
System.out.println("La tua richiesta è stata inoltrata al server.
Attendi...\n");
Response resp = call.invoke(url, "");
if (resp.generatedFault()) {
    Fault fault = resp.getFault ();
    System.out.println("\nOuch, the call failed: ");
    System.out.println(" Fault Code = " + fault.getFaultCode ());
    System.out.println(" Fault String = " + fault.getFaultString());
} else {
    Parameter res = resp.getReturnValue();
    System.out.println(res.getValue());
}
```

CREARE IL CLIENT SOAP-RPC

- Analizziamo allora questi metodi:
 - `setTargetObjectURI(iVurn:prenotazionil_)`: setta il nome URI del servizio che si intende richiedere all'RPC-ROUTER di Soap, nel nostro caso deve ovviamente coincidere con l' id specificato nel deployment descriptor, lo stesso servizio ricordiamo, può esportare più metodi.
 - `setMethodName(iUprenotal:)`: imposta il metodo specifico da eseguire tra quelli esportati dal servizio
 - `setEncodingStyleURI(Constants.NS_URI_SOAP_ENC)`: setta il tipo di codifica dei dati trasmessi
 - `setParams(params)`: Aggiunge i parametri del metodo
- Per specificare i parametri da passare al metodo si deve creare un oggetto Vector ed aggiungere come elementi degli oggetti della classe Parameter
 - **params.addElement(new Parameter("sj",String.class,sj,null))**
tra virgolette c'è il nome del parametro così come è definito nel prototipo del metodo, subito segue il tipo passato e quindi il suo valore; infine si può specificare una codifica diversa per il parametro, qualora si dovessero usare parametri che necessitano di essere serializzati.

CREARE IL CLIENT SOAP-RPC

- Una volta preparato implicitamente il messaggio non basta altro che inviarlo e ... prendere la risposta contenente i dati richiesti, tutto questo si fa istanziando un oggetto Response ed assegnandogli il risultato del metodo call.invoke:

- **Response resp = call.invoke(url, "");**

dove url rappresenta l'indirizzo del server SOAP rpc-router al quale vogliamo

inoltrare la richiesta

- **URL url = new**

- URL("http://localhost:8080/soap/servlet/rpcrouter");**

(da notare come in effetti il router per le richieste di RPC sia gestito da SOAP

tramite servlet, lo stesso vale anche per il message router che smista il traffico di

dati di qualsiasi tipo con il protocollo SOAP).

CREARE IL CLIENT SOAP-RPC



- Sull'oggetto *resp* applichiamo poi vari metodi per accedere ai contenuti rimandatici dal servizio:
 - **resp.generatedFault()**: per gestire gli eventuali errori di comunicazione o di serializzazione dei dati
 - **Parameter res = resp.getReturnValue()**: incapsula i dati ritornati in un oggetto Parameter.
 - **res.getValue()**: restituisce il valore ritornato che in effetti è di tipo String.class visto che non si sono impostati dei particolari serializzatori e delle particolari codifiche (per conoscere il tipo di un dato ritornato si può usare Parameter.getType()).

SVILUPPO DI APPLICAZIONI SOAP in WINDOWS XP



- Java è indipendente dalla piattaforma, infatti, un programma compilato Java è legato alla JVM e non al sistema operativo. E' quindi possibile eseguire lo stesso programma Java compilato una sola volta su una qualche macchina con un compilatore Java versione X, sia su una piattaforma Windows che su una piattaforma Linux, con l'unica limitazione che per fare questo però c'è bisogno che sia Windows che Linux abbiano installato una Java Virtual Machine che supporti la versione X di Java.

JAVASDK

- Il pacchetto java2SDK vers. 1.4.1. della SUN Microsystem (<http://www.sun.com>) ha il file d'installazione di 36,2 MB e permette sia di compilare il codice che eseguire i file class da riga di comando:
 - **javac nome_file.java** per compilare
 - **java nome_file** per eseguire
- Se pur sufficiente un semplice editor (tipo "Blocco Note") per implementare il codice, si consiglia di utilizzare un sw freeware chiamato JCreator, che si appoggia per la compilazione e l'esecuzione al javaSDK.
-

JAVASDK

Installazione su XP

- Eseguire il file: ***j2sdk-1_4_1_02-windows-i586.exe*** scaricato dal sito della SUN Microsystem alla pagina
<http://java.sun.com/j2se/1.4.1/>
 - Directory di destinazione: **c:\programmi\javasdk**
- Settaggio variabili d'ambiente:
 - Creazione della variabile d'ambiente JAVA_HOME:
 - ✧ *Start - pannello di controllo - prestazioni manutenzione - sistema - avanzate - variabili di ambiente - variabili di sistema*
>>nuovo (nome = JAVA_HOME; valore = c:\programmi\javasdk)
 - Settaggio variabile d'ambiente path (contiene i percorsi per i file eseguibili):
 - ✧ *Start - pannello di controllo - prestazioni manutenzione - sistema - avanzate - variabili di ambiente - variabili di sistema*
>>selezionare path>>modifica
(valore =;C:\programmi\javasdk\bin\;C:\Programmi\javasdk\jre\bin)

JAKARTA TOMCAT

Installazione su XP

- Estrazione del file: ***jakarta-tomcat-3.2.4.zip*** scaricato alla pagina <http://jakarta.apache.org/build/jakarta-tomcat/release/v3.2.4/bin/>, nella directory **C:\Programmi\Tomcat**
- Settaggio variabili all'interno del file **C:\Programmi\Tomcat\bin\startup.bat:**
 - Home directory, aggiunta riga: **SET TOMCAT_HOME = C:\Programmi\Tomcat**
 - CLASSPATH, aggiunta riga: **CLASSPATH=%TOMCAT_HOME%**
- Modifica del file **%TOMCAT_HOME%\bin\tomcat.bat** per consentire l'uso di un parser aggiornato:
 - Sostituire la riga di comando : **set CP=%CP%;%CLASSPATH%**;
con: **set CP=%CLASSPATH%;%CP%;**
- Test funzionamento Tomcat:
 - Esecuzione del file **%TOMCAT_HOME%\bin\startup.bat**
 - Esecuzione IExplorer all'indirizzo <http://localhost:8080>, provare esempi Tomcat
 - Terminazione Tomcat: esecuzione file **%TOMCAT_HOME%\bin\shutdown.bat**

APACHE SOAP

Installazione su XP

- Estrazione del file: **soap-bin-2.2.zip** scaricato dal sito
<http://xml.apache.org/dist/version-2.2> , nella directory
C:\Programmi\Tomcat\soap
 - Copia del file **C:\Programmi\Tomcat\soap\webapps\soap.war** in **C:\Programmi\Tomcat\webapps** per creare l'ambiente SOAP dentro Tomcat;
 - Copia del file **C:\Programmi\Tomcat\soap\lib\soap.jar** nella directory **C:\Programmi\Tomcat\Lib** per inserire la libreria SOAP nelle librerie di Tomcat;
- Aggiunta librerie java (scaricate dal sito della sun):
 - Estrazione file **javamail-1_2.zip** nella directory **C:\Programmi\Tomcat**
 - Estrazione file **jaf-1_0_2.zip** nella directory **C:\Programmi\Tomcat\javaaf**
 - Settaggio CLASSPATH nel file **%TOMCAT_HOME%\bin\startup.bat**, aggiunta delle righe:
 - ◆ **CLASSPATH=%CLASSPATH%;C:\Programmi\Tomcat\javaaf\activation.jar**
 - ◆ **CLASSPATH=%CLASSPATH%;C:\Programmi\Tomcat\javamail\mail.jar**
 - ◆ **CLASSPATH=%CLASSPATH%;C:\Programmi\Tomcat\lib\soap.jar**

APACHE SOAP

Installazione su XP

- Test funzionamento SOAP:
 - esecuzione del file %**TOMCAT_HOME%**\bin\startup.bat
 - esecuzione iexplorer all'indirizzo <http://localhost:8080/soap>
 - terminazione tomcat: esecuzione file %**TOMCAT_HOME**%\bin\shutdown.bat

PARSER XERCES

Installazione su XP

- Estrazione del file **Xerces-J-bin.1.2.3.zip** scaricato dalla pagina http://xml.apache.org/dist/xerces-j/old_xerces1/ nella directory **C:\Programmi\Tomcat\Xerces**
- Settaggio della variabile CLASSPATH nel file **%TOMCAT_HOME%\bin\startup.bat**, aggiunta della riga:
 - **SET CLASSPATH=%CLASSPATH%;C:\Programmi\Tomcat\Xerces\xerces.jar**

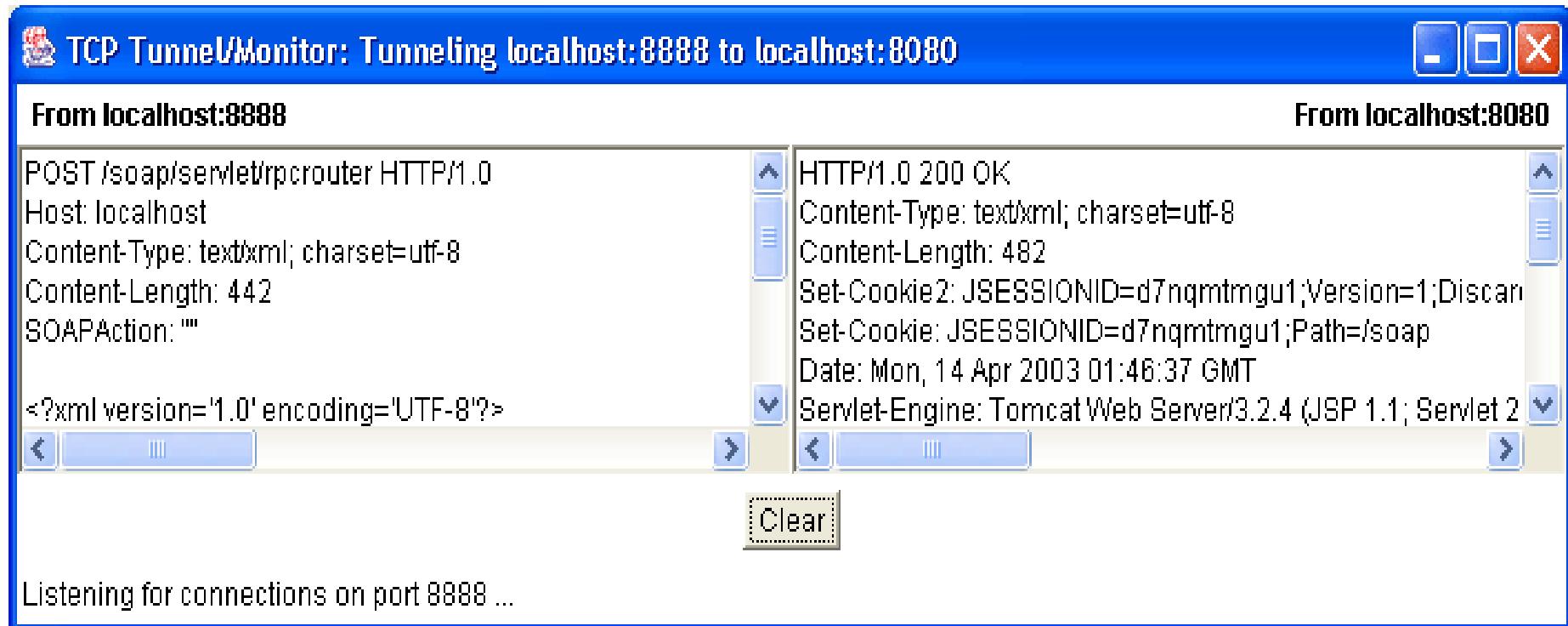
TCPTunnelGUI

Installazione su XP

- Per rendere il codice più portabile l'indirizzo è contenuto in una variabile globale "url" che viene inizializzata con il primo parametro della chiamata dell'esecuzione. Nel caso di server locale la chiamata sarà:
"java Client http://localhost:8888/soap/servlet/rpcrouter".
- In realtà la porta su cui ascolta tomcat è la 8080, ma la 8888 è quella che è stata settata per introdurre il TCPTunnelGUI che è un'applicazione messa a disposizione nelle librerie SOAP che si interpone tra Client e Tomcat e mostra i messaggi di richieste che viaggiano da Client a Server e quelli di repliche del Server verso il Client.
- Il TCPTunnelGUI viene richiamato con il comando:
 - **" java org.apache.soap.util.TcpTunnelGui 8888 localhost 8080"**
- Questo comando si può inserire in un file .bat MS-DOS, **TCPTunnelGUI.bat**, per poter avviare l'applicazione rapidamente.

TCP Tunnel GUI

Installazione su XP



SOA



- Una Service Oriented Architecture (SOA) è essenzialmente una collezione di servizi che comunicano tra di loro
- La comunicazione avviene per:
 - Semplice passaggio di dati tra servizi
 - Coordinamento tra due o più servizi

C'era una volta...

..., in un'organizzazione felice, un'applicazione con tanti utenti soddisfatti, ma ...



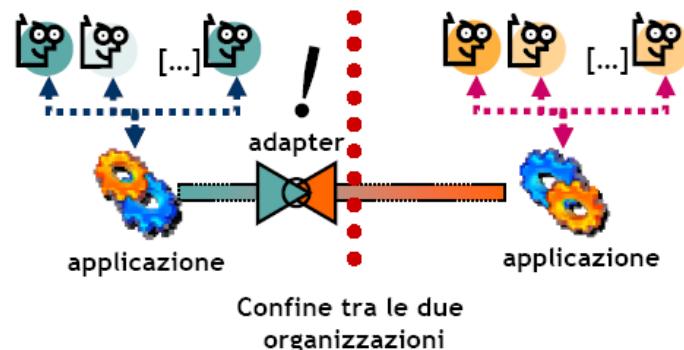
... e, per esigenze di business, le due organizzazioni sentirono l'esigenza di integrare le due applicazioni.



... ma l'organizzazione non era certo l'unica. Altre organizzazioni avevano applicazione con utenti soddisfatti e ...

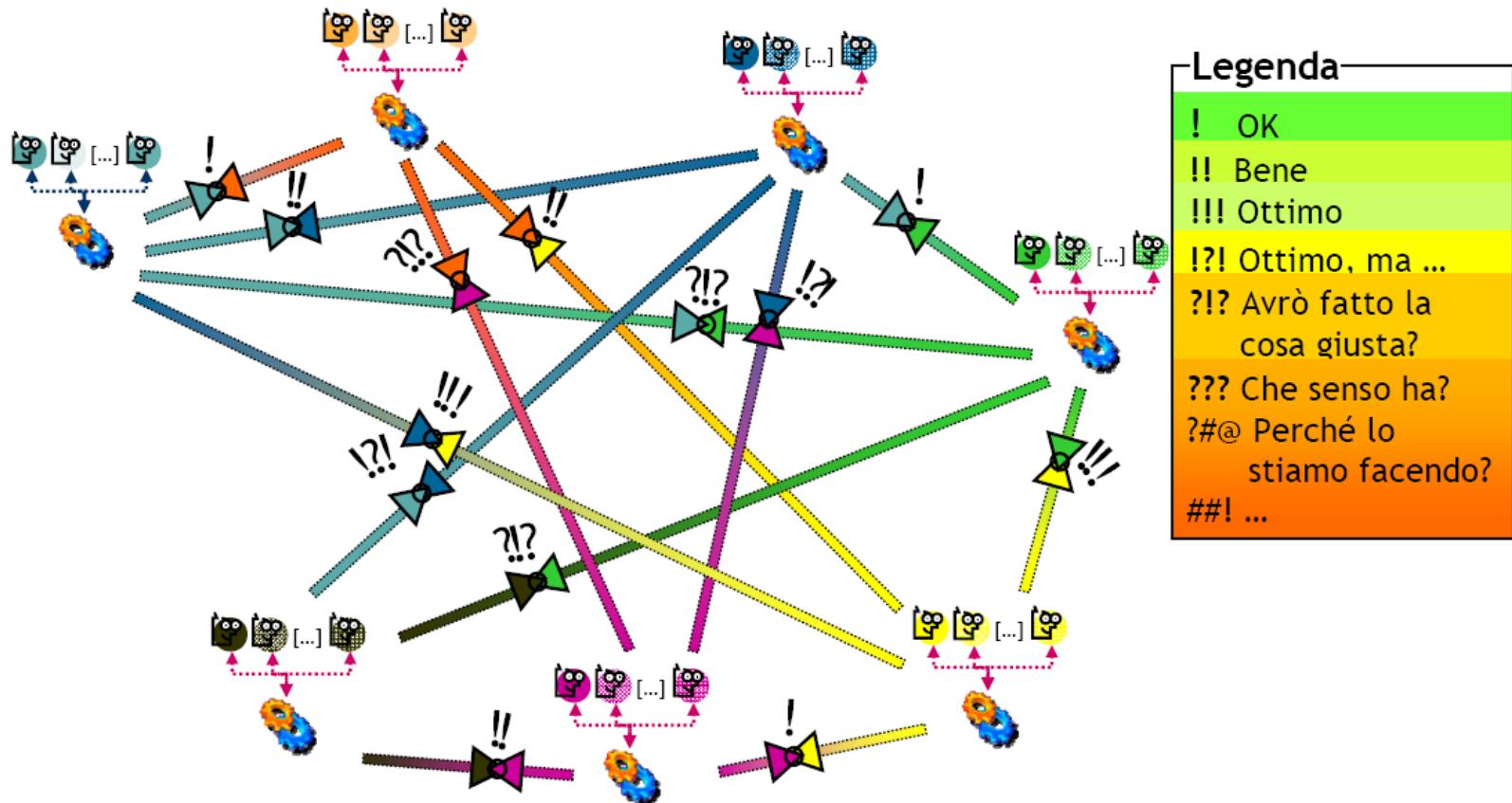


Pensando alla contingenza l'organizzazione felice decise di addossarsi lo sforzo di integrazione, ...



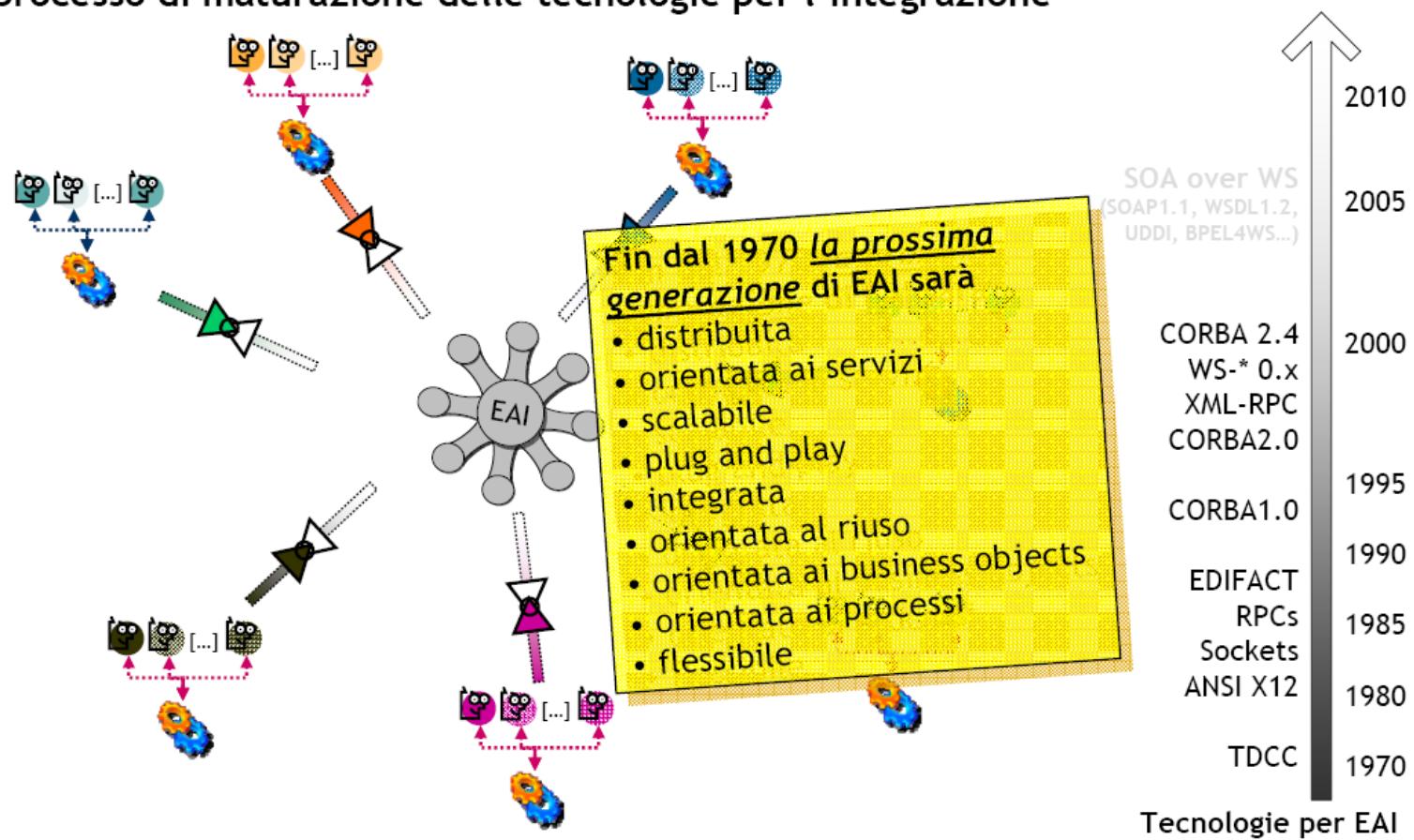
ma gestire la contingenza...

... quando il problema di integrazione coinvolge molte organizzazioni autonome ma interdipendenti non paga sul lungo periodo



fu così che...

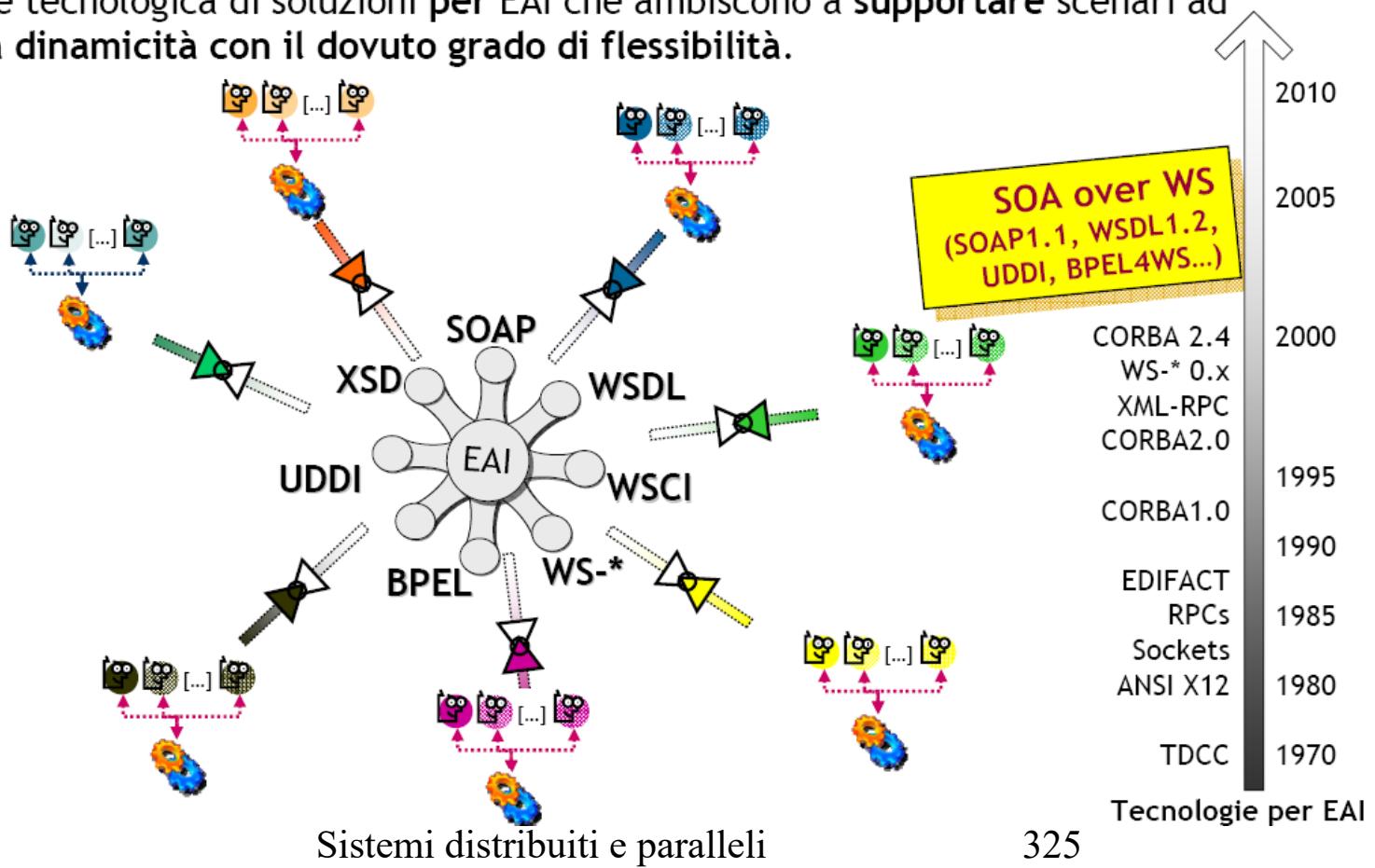
... nacquero le soluzioni di Enterprise Application Integration e iniziò il processo di maturazione delle tecnologie per l'integrazione



SOA

Service Oriented Architecture

Il continuo processo di maturazione delle tecnologie ha portato negli ultimi anni all'affermazione delle SOA implementate tramite Web Services come potenziale base tecnologica di soluzioni per EAI che ambiscono a supportare scenari ad alta dinamicità con il dovuto grado di flessibilità.



SOA e modelli di integrazione

Maturità del modello di integrazione	1 silos	2 object	3 component	4 service	5 SOA
IT vista dal business	Function oriented	Function oriented	Function oriented	Service oriented	Service oriented
metodo	strutturata	Object oriented	Component based	Service modeling	IT process modeling
applicazioni	moduli	moduli	componenti	servizi	Processi IT integrati via servizi
architettura	monolitica	a layer	a componenti	Ispirata alla SOA	SOA
infrastruttura	Piattaforme proprietarie	Piattaforme proprietarie	Piattaforme aperte	Web Services	Web Services

Sistemi distribuiti e paralleli

SOA e standard

Ogni generazione di EAI sembra identica alla precedente, ma qualcosa è cambiato.

	Java	CORBA	XML-RPC	Web Services
Meccanismo di invocazione	Java RMI	CORBA RMI	RPC	qualsiasi linguaggio
Formato Dati	Java Serialization	CDR	XML	XML
Formato wire	stream	GIOP	PDU	SOAP
Protocollo di trasferimento	JRMP	IIOP	RPC CO	protocolli internet
Descrizione dell'interfaccia	Java Interface	CORBA IDL	DCE IDL	WSDL
Meccanismo di discovery	Java Registry	COS naming	CDS	UDDI



standard proprietario



standard specifico

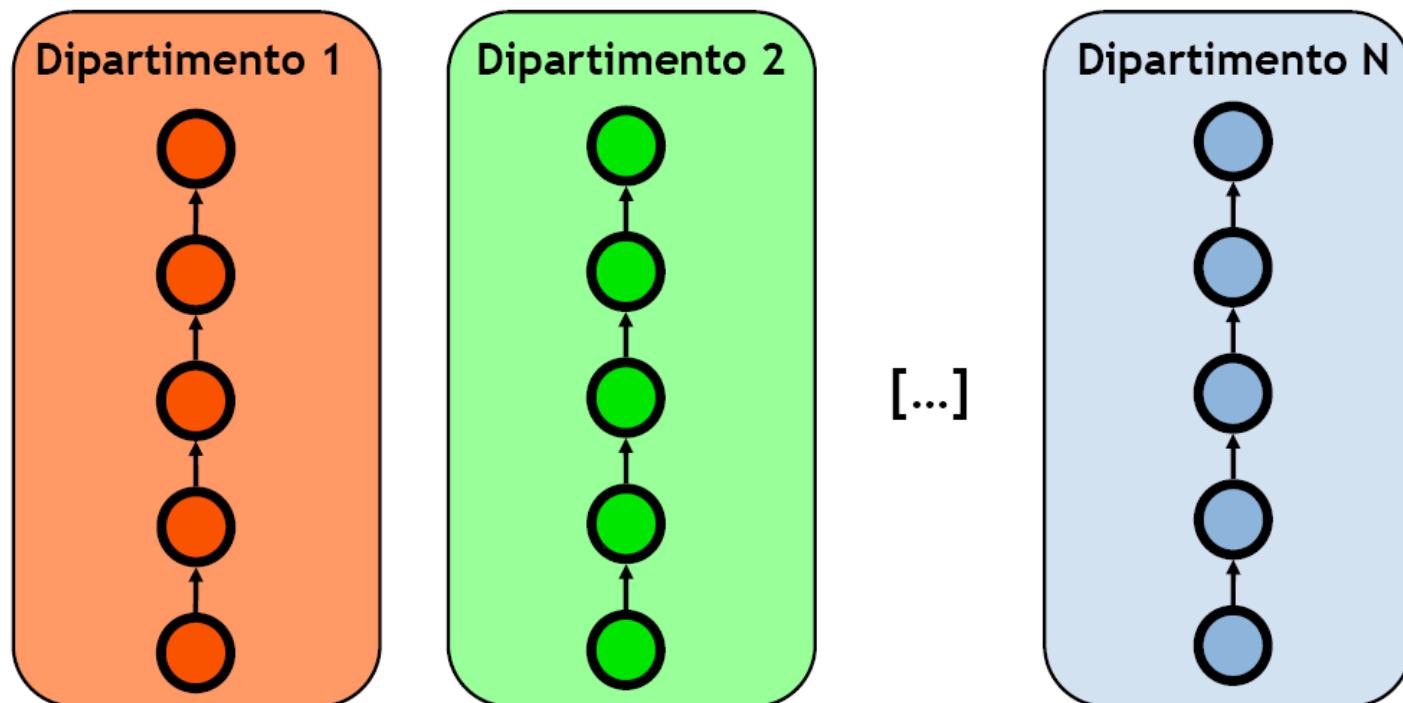


standard generico

Passi verso una SOA:

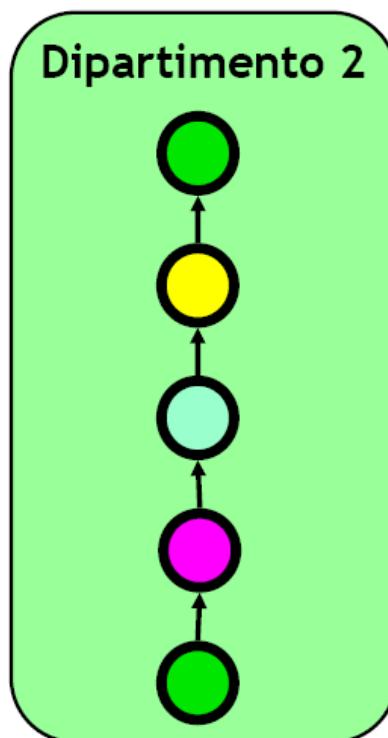
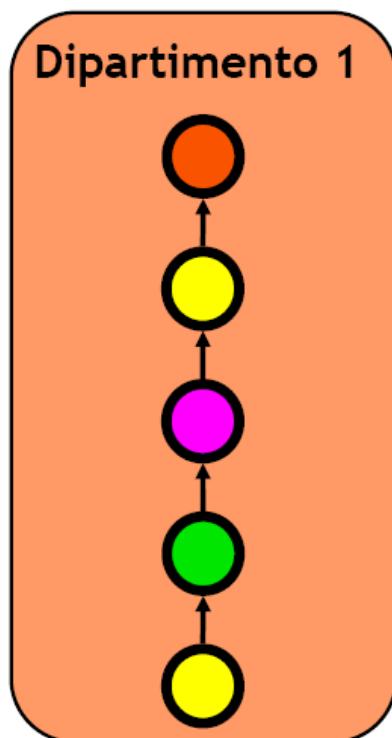
1. senza integrazione

In assenza di integrazione **ogni dipartimento adotta le proprie soluzioni IT integrate** (non semplici applicazioni) e le usa in modo esclusivo.

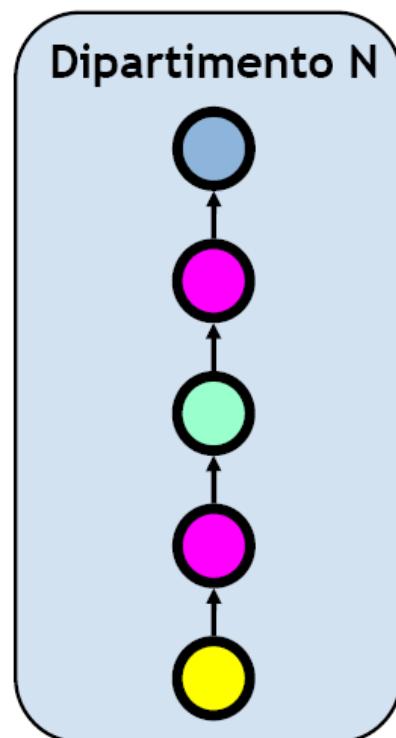


2. estrazione dei servizi

Il primo passo verso una SOA consiste nel razionalizzare le soluzioni IT esistenti individuando le parti potenzialmente comuni a più dipartimenti ed esponendole come servizi.



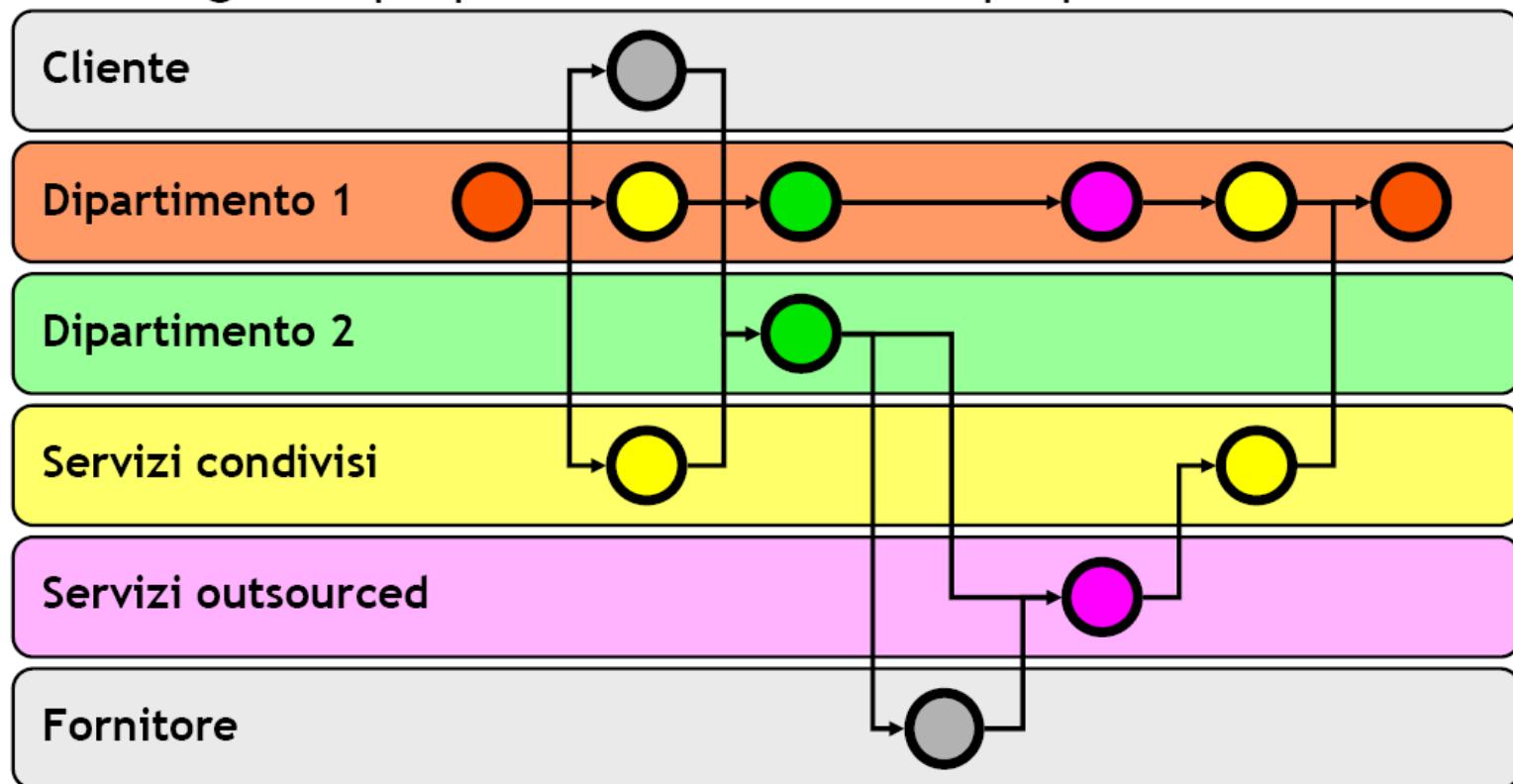
[...]



Sistemi distribuiti e paralleli

3. composizione di processi

I servizi possono poi essere composti in modo da supportare non solo i processi interni, ma anche processi che coinvolgono i propri fornitori e verso i propri clienti.

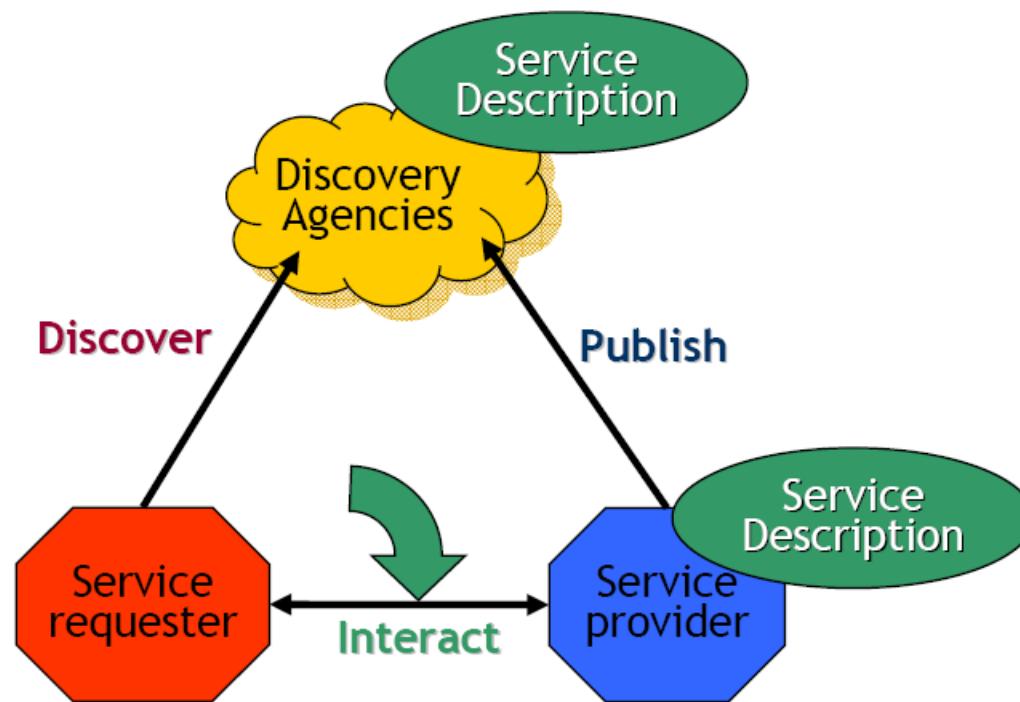


SOA vs. Web Service

	SOA	Web Services
Cos'è ...	<p>SOA è una filosofia architetturale:</p> <ul style="list-style-type: none">→ Flessibile→ Aperta al cambiamento e all'evoluzione→ Astraе dalla logica di funzionamento interna <p>Approccio allo sviluppo di una architettura basata su servizi</p>	<p>Sono un insieme di standard aperti (W3C e OASIS) ampiamente sostenuti dai principali attori ICT</p> <p>Sono una tecnologia impiegata perché</p> <ul style="list-style-type: none">→ garantisce un buon grado di flessibilità e evolvibilità→ astraendo dallo specifico linguaggio usato per implementare il service requester e il service provider
Cosa non è ...	<p>SOA</p> <ul style="list-style-type: none">→ non è una particolare tecnologia→ non può essere un prodotto vendibile. <p>Al più una specifica tecnologia può servire ad implementare una SOA e un prodotto di può essere implementato con in mente il paradigma delle SOA</p>	<p>I Web Service</p> <ul style="list-style-type: none">→ non sono una soluzione proprietaria→ non sono un prodotto vendibile <p>Al più</p> <ul style="list-style-type: none">→ uno specifico servizio può essere implementato usando i Web Service→ un'applicazione può essere esposto come Web Service

SOA su Web Service

Il mito delle Service Oriented Architecture (SOA) dice che abilitano flessibilità e dinamismo tramite una *descrizione ricca del servizio* e un *meccanismo di pubblicazione/scoperta* di tali descrizioni che permette a richiedenti occasionali di interagire con il servizio senza (o con limitate) conoscenze a priori.



Valutazione di SOA su Web Service

Processi che una SOA dovrebbe supportare

→ **Pubblicazione:**

- rendere disponibile una descrizione di un servizio

→ **Discovery:**

- trovare servizi adatti ad un certo scopo

→ **Selezione:**

- trovare il migliore servizio tra quelli disponibili

→ **Compensazione:**

- trovare un servizio per effettuare un “undo” o mitigare un effetto non desiderato

→ **Rimpiazzo:**

- sostituire un servizio con un altro equivalente

→ **Esecuzione:**

- invocare da programma servizi

→ **Composizione:**

- aggregare servizi

→ **Monitoring:**

- Controllare l'esecuzione

→ **Auditing:**

- Verificare che le esecuzione di un servizio abbia seguito un processo prestabilito

Valutazione di SOA su Web Service

La realtà è una selva di standard ufficiali, ufficiosi, de facto e presunti nota come WS-AH - Web Services Acronym Hell



Valutazione di SOA su Web Service

Nella selva di acronimi **serve esperienza per**
→ **saper cosa usare** per quale scopo e
→ **non farsi ingannare** dalla moda del momento

I tre pilastri che fanno dei Web Service una base tecnologica per le SOA

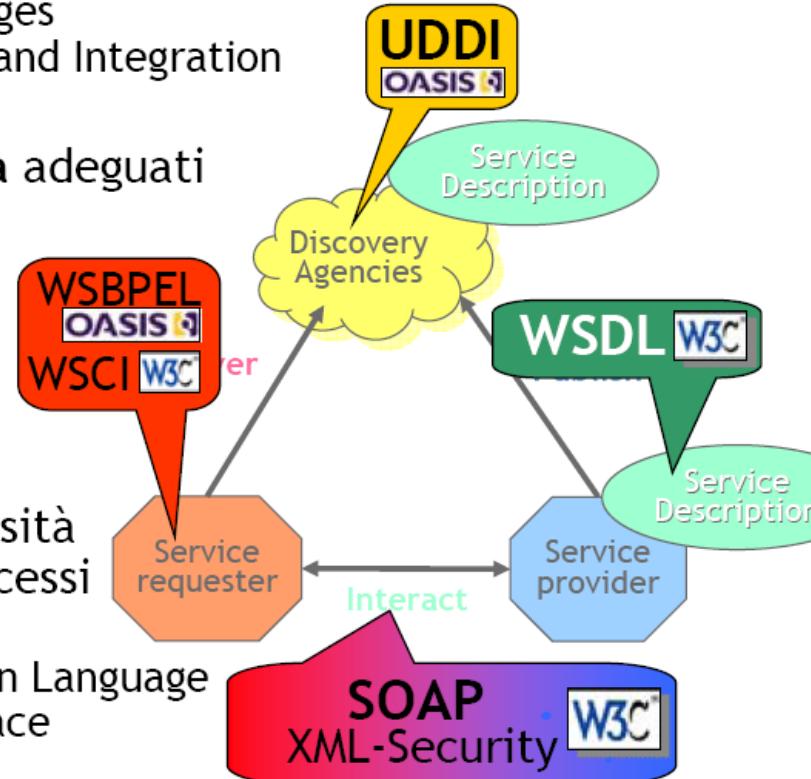
- WSDL: Web Services Description Languages
- UDDI: Universal Description, Discovery and Integration
- SOAP: Simple Object Access Protocol

servono, poi, **meccanismi di sicurezza** adeguati
→ **XML-security**

- XML-Encryption: garantire la confidenzialità
- XML-Signature: garantire autenticità e non-ripudio
- XKMS: gestione delle chiavi
- **SOAP-Security:** integrazione in SOAP

e, sempre più spesso, si sente la necessità di **comporre servizi** elementari in processi complessi

- WSBPEL: WS Business Process Execution Language
- WSCI: Web Service Choreography Interface



Documenti WSDL

Un documento WSDL descrive

→ **cosa** un Web Service offre,

→ Describe il **tipo** di servizio

→ Lo stesso tipo di servizio dovrebbe essere implementato da tutti i providers.

→ Definisce un'**interfaccia logica**, ovvero il set di operazioni del servizio, in termini di

→ I *messaggi* in *ingresso* e *uscita*

→ Il loro *formato*

→ I *tipi di dato* di ciascun elemento del messaggio

→ **come** comunica,

→ Describe il “*binding*” dell’*interfaccia astratta* a i *protocolli internet* sottostanti

→ Specifica

→ quale XML Schema deve essere utilizzata per serializzare i dati;

→ come costruire la “busta” (envelope) SOAP;

→ quali header addizionali debbano essere inclusi; e

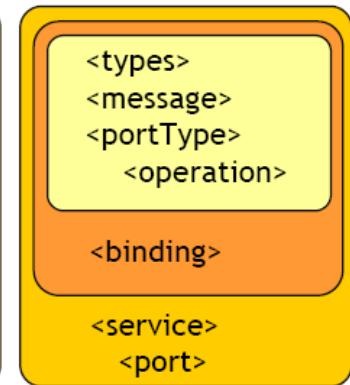
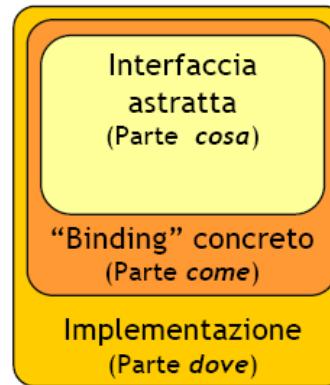
→ quale transfer protocol usare.

→ include o importa la parte *cosa* del WSDL

→ e **dove** trovarlo

→ Describe come il servizio è implementato in termini di **porte** che *implementano* un *binding concreto* dell’*interfaccia astratta*

→ include o importa la parte *come* del WSDL



Registro UDDI

Un registro UDDI (v 3.0) contiene:

→ **Pagine Bianche**

- Nome del service provider, *identificativo, indirizzo e altre informazioni per contattare la società*

→ **Pagine Gialle**

- *Sistemi di classificazione di service provider e servizi su base geografica, tipo di industria, etc.*

→ **Pagine Verdi**

- *Descrizione tecnica delle interfacce del servizio (es. Acquisto libri) e del punto di accesso (URL, e-mail, ecc.) utilizzando opportuni "tModel"*

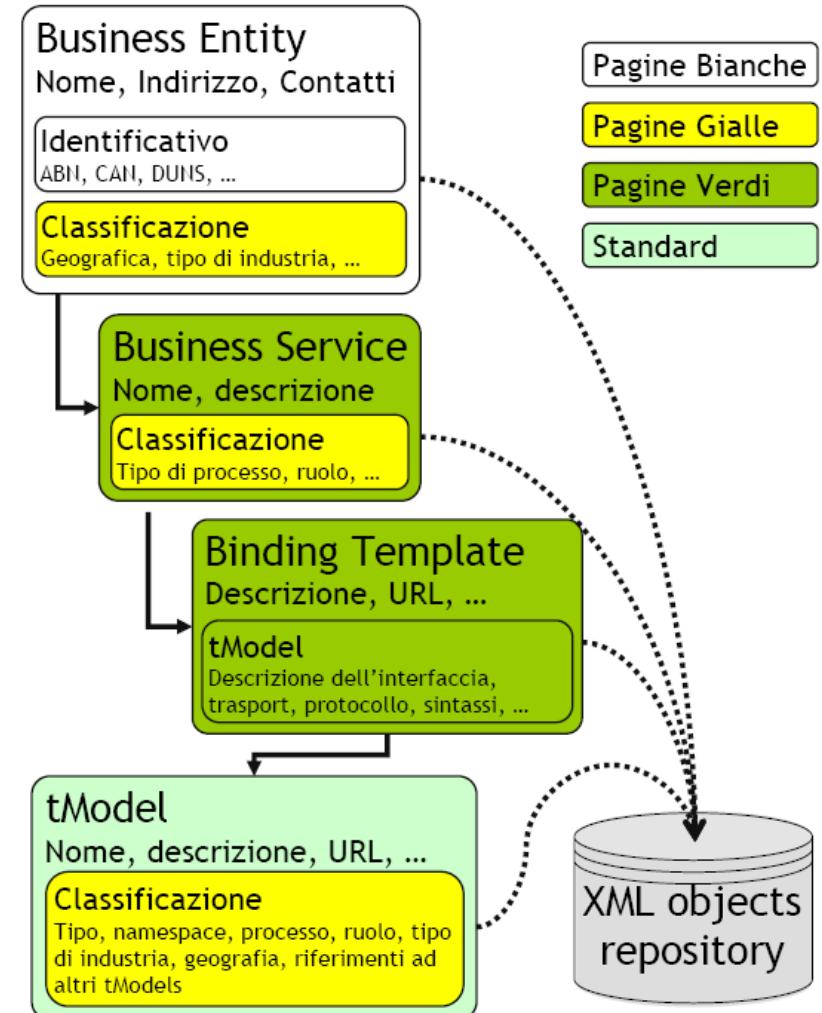
→ **tModel**

- *Definizione tecnica di un tipo di servizio tipicamente emessa da enti di standardizzazione di specifici domini (es. EAN/UCC*) per essere utilizzata dai service provider.*

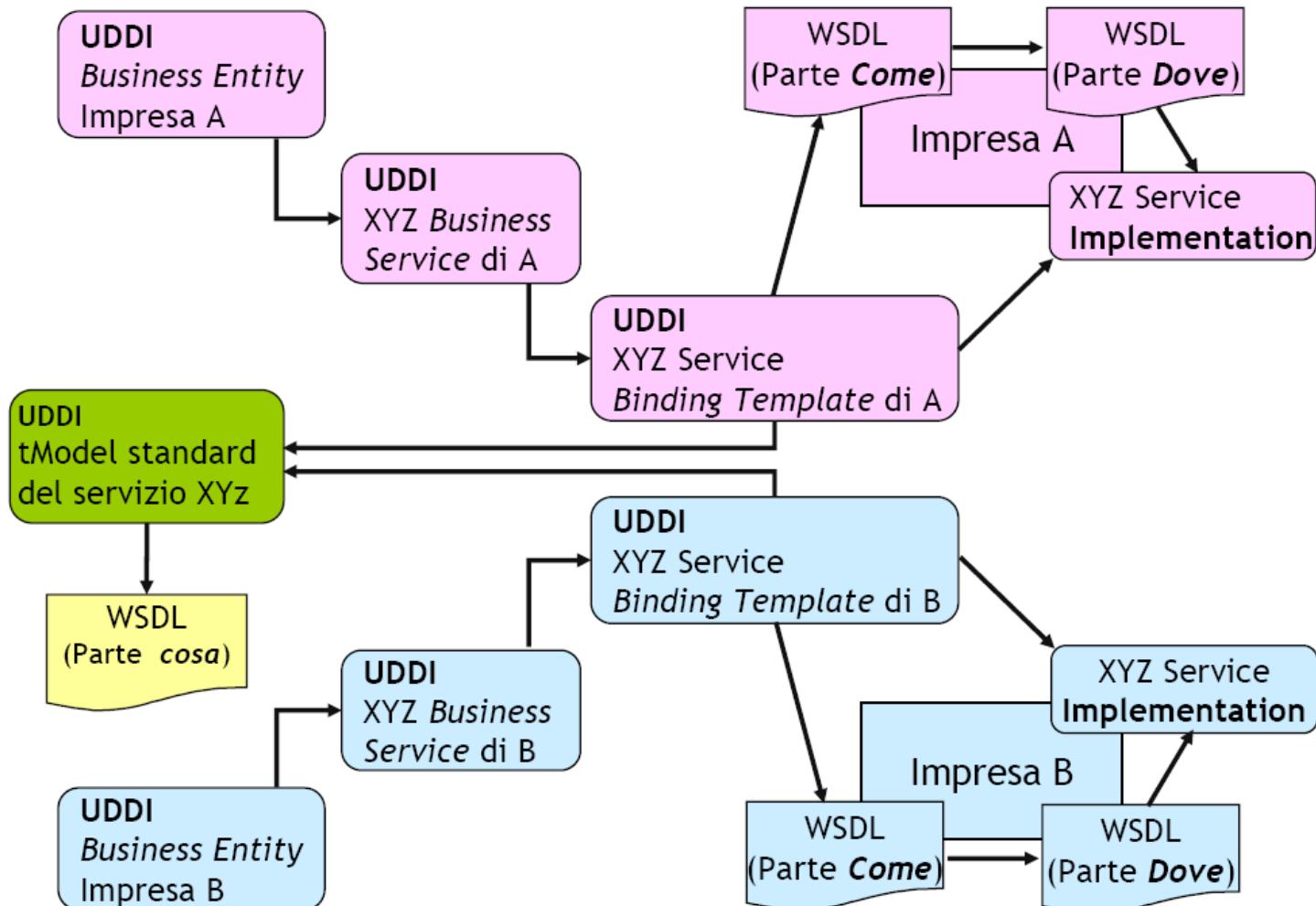
→ **Oggetti nel repository**

- Tutti gli oggetti descritti sopra sono immagazzinati come metadati nel registro UDDI prevede API per **scrivere e cercare nel repository**

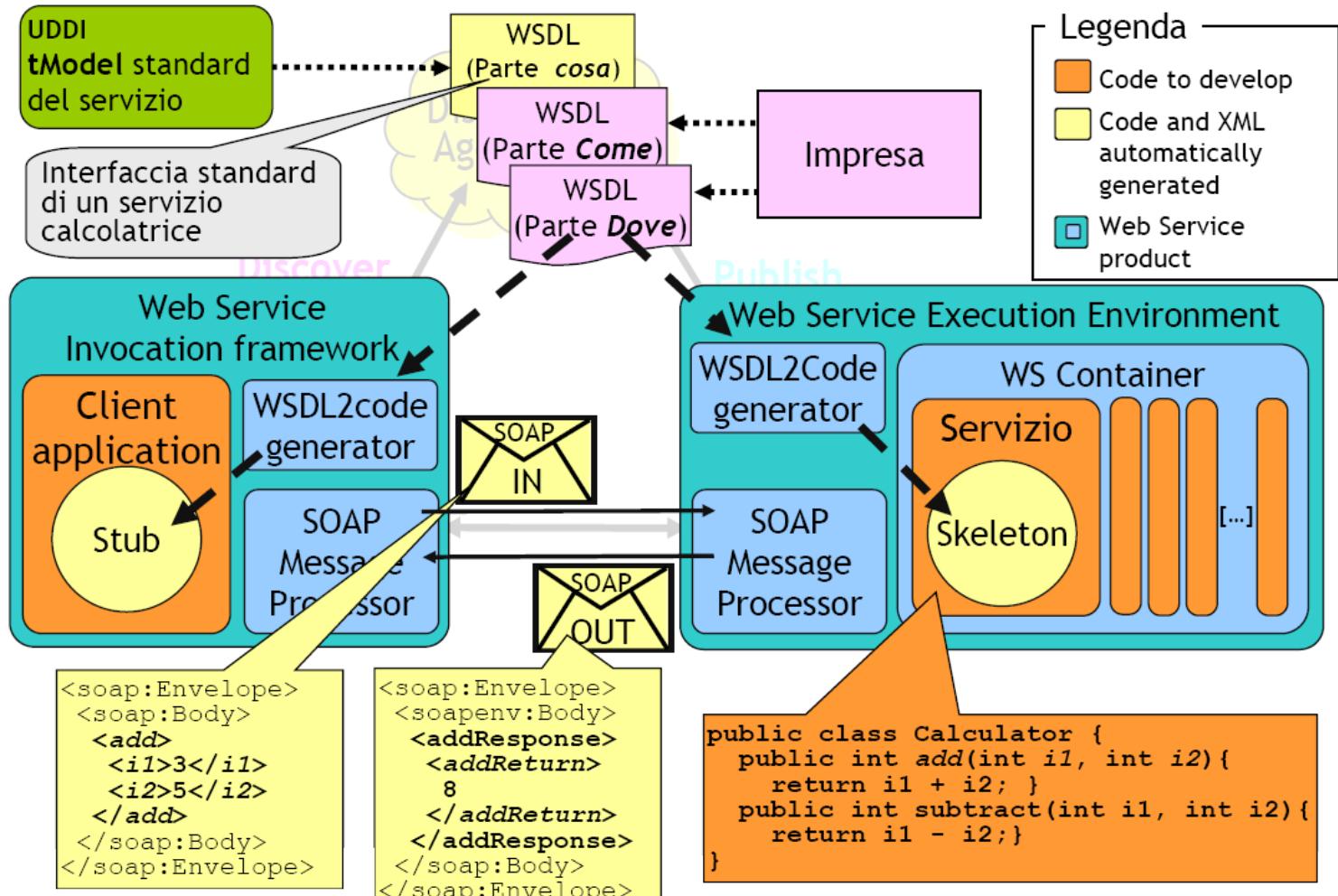
* EAN.UCC System standardizza bar codes, insiemi di transazioni EDI transactions sets, schemi XML, e altre soluzioni per rendere più efficiente l'e-business



Relazione tra WSDL e UDDI



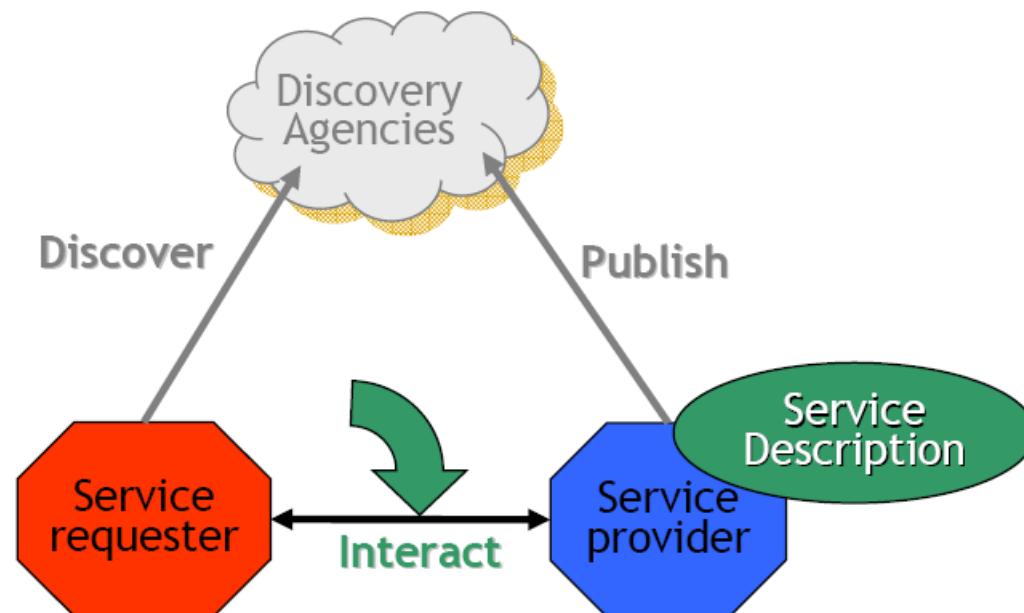
Web Service all'opera



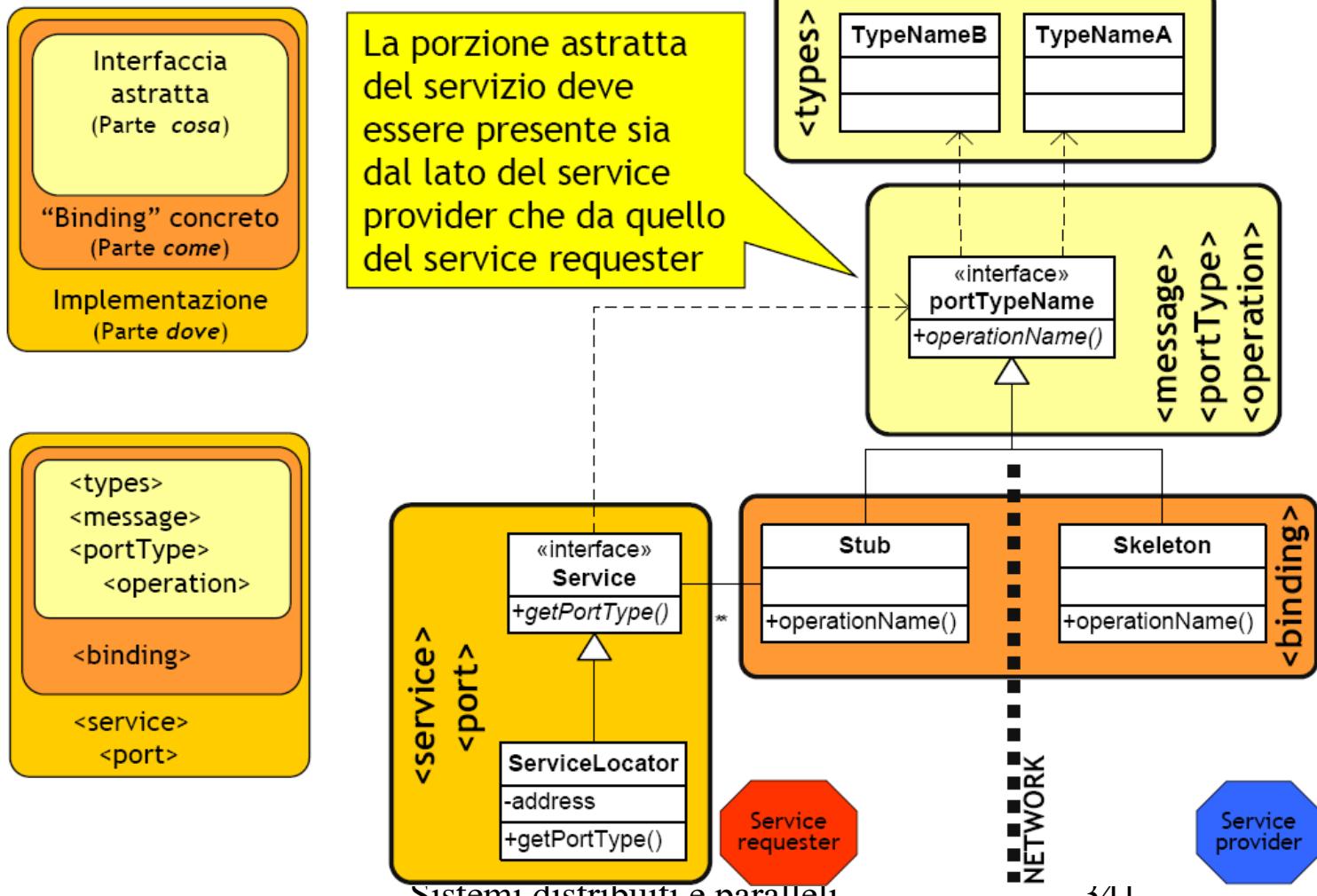
WSDL e codice

Nella realtà i Web Services **realizzano** almeno in parte la visione delle Service Oriented Architecture (**SOA**)

La descrizione in **WSDL** di un servizio permette di **automatizzare** buona parte del **lavoro richiesto per** scrivere il codice necessario a instaurare un'**interazione** via **SOAP** tra service requester e service provider.



WSDL e codice



WSDL e codice: AXIS

AXIS (il Web Service Execution Environment di Apache) offre un tool come WSDL2Java che dal WSDL genera

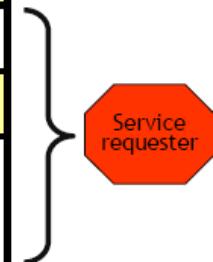
- strutture dati e interfacce comuni
- stubs per i service requester
- skeletons per i service provider



```
[lanciando c:\>java org.apache.axis.WSDL2Java subtracter.wsdl]
```

WSDL2Java genera

WSDL clause	Java class(es) generated
per ogni type	una classe java
per ogni portType	un'interfaccia java
per ogni binding	una classe stub
per ogni service	una service interface
	una implementazione del service (detta service locator)



WSDL e codice:

WSDL:types -> strutture dati Java

Dal wsdl:types è possibile generare le corrispondenti strutture dati Java

wsdl:types	java
<pre><xsd:complexType name="phone"> <xsd:all> <xsd:element name="areaCode" type="xsd:int"/> <xsd:element name="exchange" type="xsd:string"/> <xsd:element name="number" type="xsd:string"/> </xsd:all> </xsd:complexType></pre>	<pre>public class Phone implements java.io.Serializable { public Phone() {...} public int getAreaCode() {...} public void setAreaCode(int areaCode) {...} public java.lang.String getExchange() {...} public void setExchange(String exchange) {...} public java.lang.String getNumber() {...} public void setNumber(String number) {...} public boolean equals(Object obj) {...} public int hashCode() {...} }</pre>

WSDL e codice:

WSDL:portType -> interfaccia Java

Dal wsdl:portType è possibile generare un'interfaccia java per accedere alle operazioni del servizio

wsdl:portType

```
<wsdl:message name="SubtractRequest">
    <wsdl:part element="impl:Subtract" name="parameters"/>
</wsdl:message>
<wsdl:message name="SubtractResponse">
    <wsdl:part element="impl:SubtractResponse" name="parameters"/>
</wsdl:message>
<wsdl:portType name="Subtracter">
    <wsdl:operation name="Subtract">
        <wsdl:input message="impl:SubtractRequest" name="SubtractRequest"/>
        <wsdl:output message="impl:SubtractResponse" name="SubtractResponse"/>
    </wsdl:operation>
</wsdl:portType>
```

java

```
package samples.subtracter;
public interface Subtracter extends java.rmi.Remote {
    public float subtract(float minuend, float subtrahend) throws java.rmi.RemoteException;
}
```

WSDL e codice:

WSDL:binding -> requester stub

Dal wsdl:binding è possibile generare uno **stub** che implementa l'interfaccia per accedere alle operazioni del servizio

wsdl:binding

```
<wsdl:binding name="SubtracterSoapBinding" type="impl:Subtracter">
    <wsdlsoap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http">
        <wsdl:operation name="Subtract">
            <wsdlsoap:operation soapAction="" />
            <wsdl:input name="SubtractRequest">
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="SubtractResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
```

Java

```
package samples.subtracter;
public class SubtracterSoapBindingStub extends Stub implements Subtracter {
    public SubtracterSoapBindingStub () {...}
    public SubtracterSoapBindingStubep, Service s) {...}
    protected Call createCall() {...}
    public float subtract(float minuend, float subtrahend) {...}
}
```

Service
requester

WSDL e codice:

WSDL:service -> service interface

Normalmente, il service requester non crea direttamente un'istanza dello stub, ma crea una classe chiamata **service locator** che con un metodo appropriato ritorna un'istanza dello stub. Questo service locator è generato a partire dal wsdl:service.

wsdl:service

```
<wsdl:service name="SubtracterService">
    <wsdl:port binding="impl:SubtracterSoapBinding" name="Subtracter">
        <wsdlsoap:address location="http://localhost:8080/axis/services/Subtracter"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Java

```
package samples.subtracter;
public interface SubtracterService extends javax.xml.rpc.Service {
    public String getSubtracterAddress();
    public samples.subtracter.Subtracter getSubtracter ();
    public samples.subtracter.Subtracter getSubtracter (URL portAddress);
}
package samples.math;
public class SubtracterServiceLocator Service implements SubtracterService {
    private String Math_address = "http://localhost:8080/axis/services/Math";
    public samples.subtracter.Subtracter getSubtracter() {...}
}
```

Service requester

WSDL e codice:

Come il requester usa lo stub?

Il service locator può essere utilizzato per ottenere un'istanza dello stub all'interno di un programma lato requester.

Java

Service
requester

```
package samples.subtracter;

public class TestClient {
    public static void main(String[] args) {
        SubtracterServiceLocator sl = new SubtracterServiceLocator();
        try {
            samples.subtracter.Subtracter s = sl.getSubtracter();
            float r = s.subtract(1.27f,3.14f);
            System.out.println("1.27-3.14f="+r);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

WSDL e codice: WSDL2Java: lato provider

Lo **Stub** è la rappresentazione Java del Web Service lato **client**

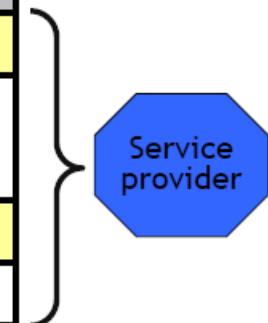
Allo stesso modo è possibile generare lo **Skeleton** (la rappresentazione Java del Web Service **lato server**).

Lanciando

```
c:/> java org.apache.axis.WSDL2Java  
--server-side --skeletonDeploy true subtracter.wsdl
```

WSDL2Java genera

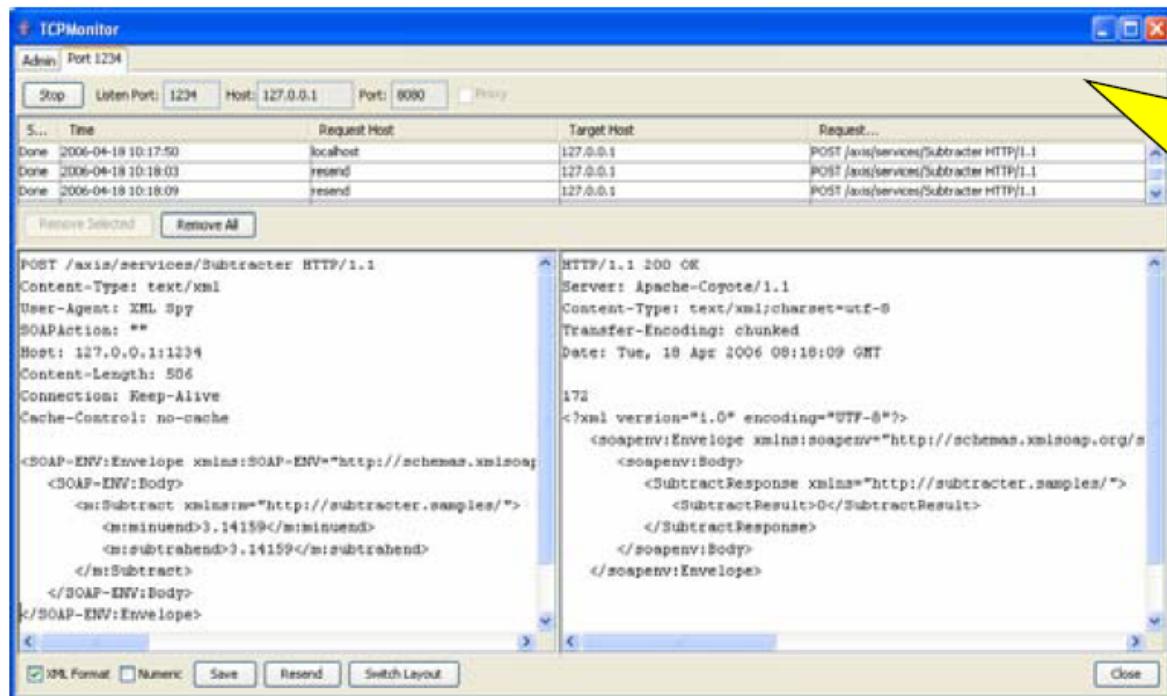
WSDL clause	Java class(es) generated
per ogni binding	una classe skeleton
	un template di implementazione del servizio lato server
per ogni service	un file deploy.wsdd
	un file undeploy.wsdd



WSDL2Java e SOAP

WSDL2Java permette di implementare sia il service requester che il service provider in Java, ma **requester e provider non devono essere implementati nello stesso linguaggio**.

Perchè si possano parlare è sufficiente che **implementino un protocollo internet** (es. HTTP, JMS, etc.) e che **utilizzino SOAP** per scambiarsi informazione.



Tool come *TCPMonitor* (incluso nella distribuzione di Axis) permettono di vedere i messaggi SOAP scambiati

Valutazione di SOA su Web Service

Processi che una SOA dovrebbe supportare

→ **Pubblicazione:**

- rendere disponibile una descrizione di un servizio

→ **Discovery:**

- trovare servizi adatti ad un certo scopo

→ **Selezione:**

- trovare il migliore servizio tra quelli disponibili

→ **Compensazione:**

- trovare un servizio per effettuare un “undo” o mitigare un effetto non desiderato

→ **Rimpiazzo:**

- sostituire un servizio con un altro equivalente

→ **Esecuzione:**

- invocare da programma servizi

→ **Composizione:**

- aggregare servizi

→ **Monitoring:**

- Controllare l'esecuzione

→ **Auditing:**

- Verificare che le esecuzione di un servizio abbia seguito un processo prestabilito

Pagella ai WS come implementazione di SOA

<i>Pubblicazione :</i> rendere disponibile una descrizione di un servizio	8
<i>Discovery :</i> trovare servizi adatti ad un certo scopo	6
<i>Selezione :</i> trovare il migliore servizio tra quelli disponibili	5
<i>Compensazione :</i> trovare un servizio per effettuare un “undo” o mitigare un effetto non desiderato	4
<i>Rimpiazzo :</i> sostituire un servizio con un altro equivalente	4
<i>Esecuzione :</i> invocare da programma servizi	9
<i>Composizione :</i> aggregare servizi	6
<i>Monitoring :</i> Controllare l'esecuzione	6
<i>Auditing :</i> Verificare che le esecuzione di un servizio abbia seguito un processo prestabilito	6