

Architetture Orientate ai Servizi


Riferimenti

- I. Sommerville, Ingegneria del Software, 10° edizione, capitolo 18

Premessa

- La realizzazione di sistemi distribuiti su piattaforme eterogenee richiede l'utilizzo di middleware che forniscono i servizi atti a consentire la comunicazione fra componenti distribuiti.
 - I middleware possono essere più o meno difficili da usare (es. CORBA vs. Java RMI), e fornire servizi più o meno sofisticati.
 - Non è possibile far comunicare componenti aderenti a diversi modelli di componente e relativi middleware (es. client DCOM e server RMI).
 - I middleware sono usabili solo in ambito intranet, data l'impossibilità per i dati scambiati (in binario basati su specifici protocolli) di superare i firewall aziendali.

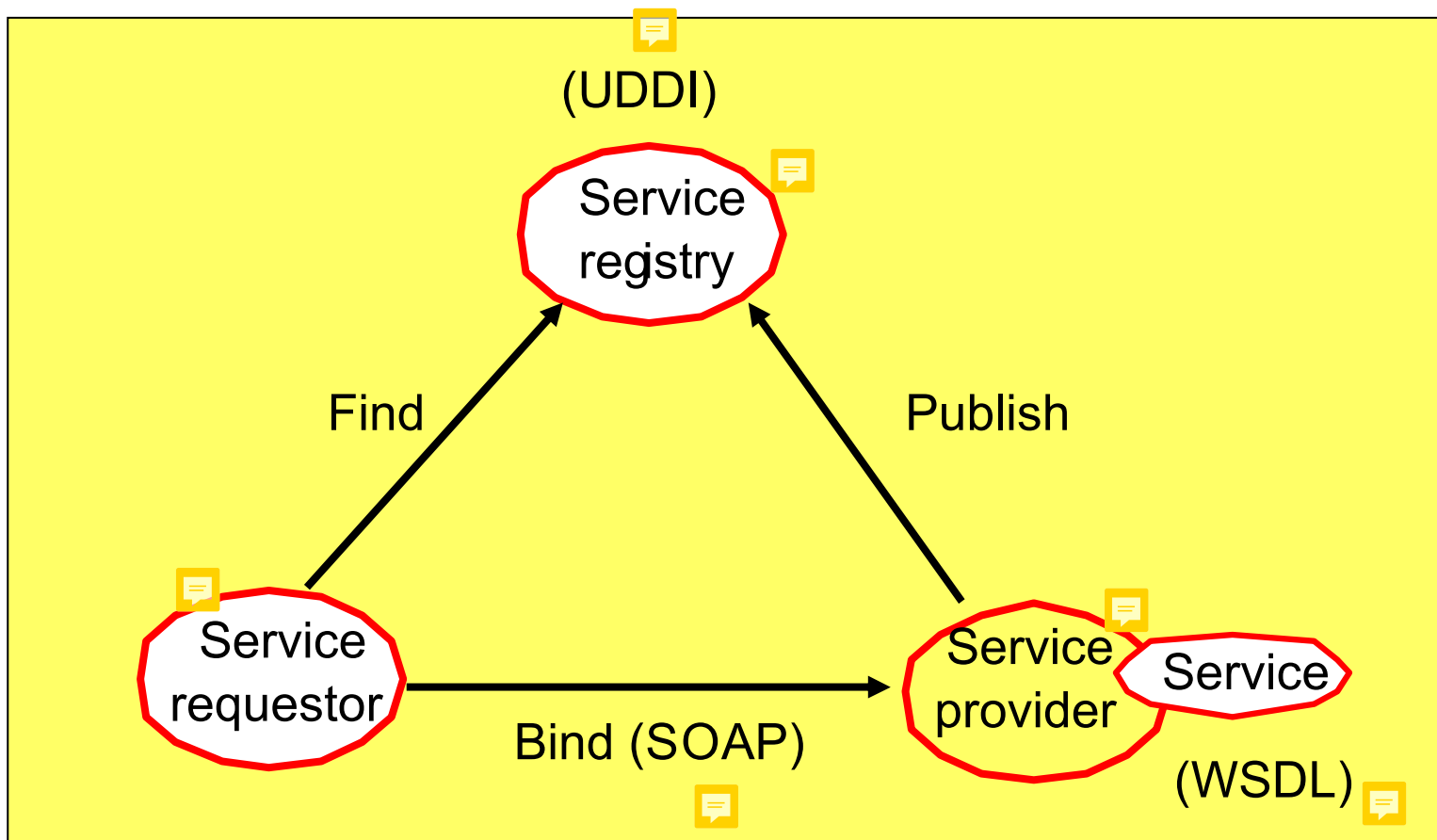
Il futuro: Architetture orientate ai servizi

- Le **Architetture Orientate ai Servizi** (SOA) sono state proposte per consentire una sistematica interazione fra applicazioni (application-to-application)
 - Svincolandosi dai limiti delle architetture a componenti basate su middleware 
 - Es. Una implementazione delle SOA è quella fornita dai Web Services

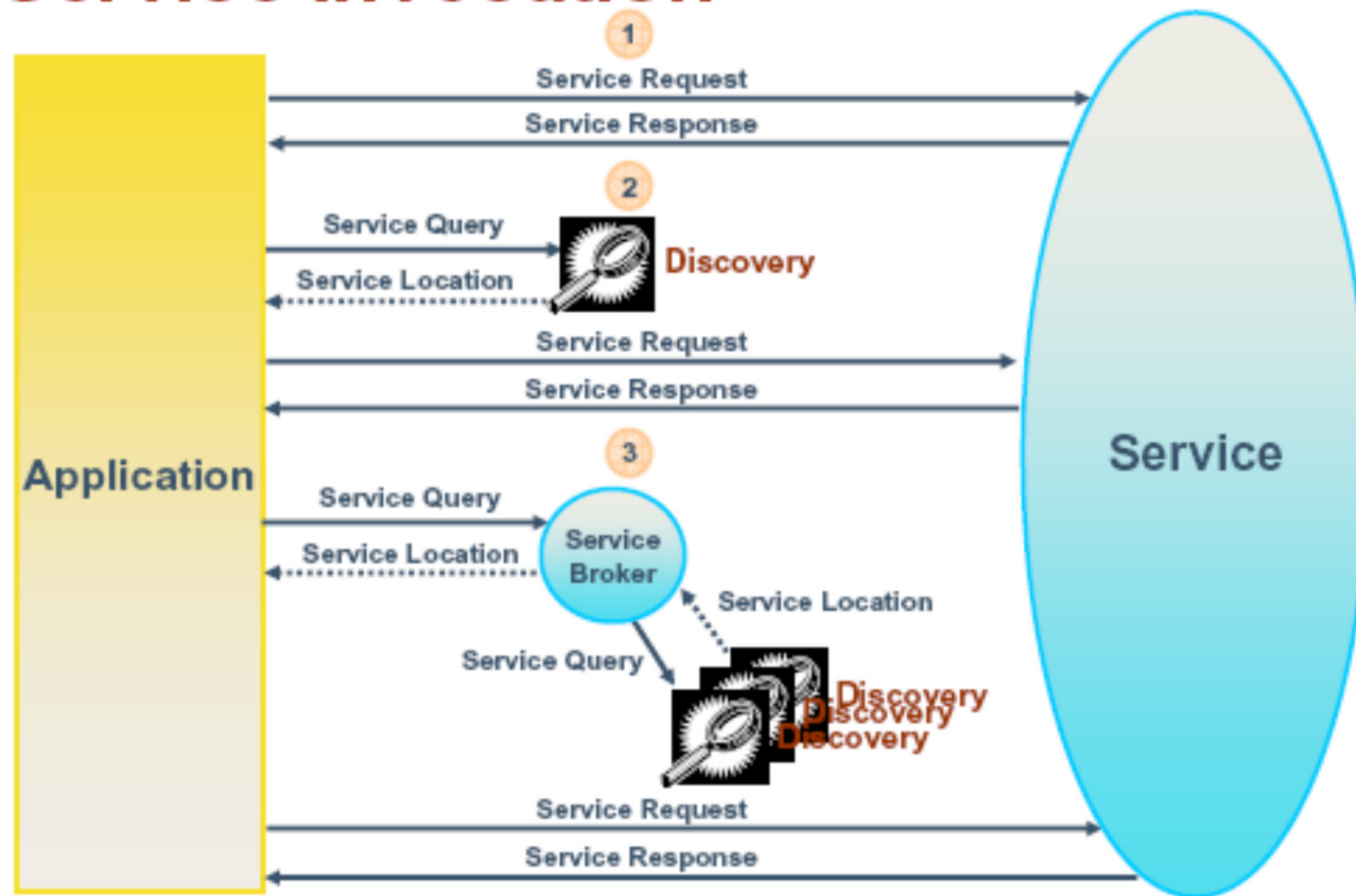
Service-oriented architectures (SOA)

- Le Architetture Orientate ai Servizi (SOA) sono un modo di progettare sistemi distribuiti i cui componenti sono costituiti da **servizi indipendenti**.
- I servizi sono componenti *stand-alone*, in grado di risolvere specifici problemi, che possono essere offerti da differenti fornitori ed essere eseguiti su computer diversi.
- Per rendere possibili tali attività, devono essere adottati **protocolli standard** per supportare:
 - la comunicazione fra servizi
 - lo scambio di informazioni fra servizi

Attori nelle SOA




Service Invocation



Sviluppo di Servizi

- Obiettivi :

- Consentire **l'interoperabilità** universale 
- Consentire una **diffusa adozione e l'ubiquità** dei servizi
- Abilitare il **binding dinamico** (su scala geografica)
- Supportare efficacemente sia ambienti aperti (Web) che più vincolati



Sviluppo di Servizi

- **Requisiti:**

- Basati su standard (che forniscono un supporto pervasivo)
- Si richiede solo la disponibilità di una minima infrastruttura (è richiesto solo l'implementazione di un set minimo di standard)
- Possibilità di concentrarsi su messaggi e documenti scambiati, piuttosto che su API

Sviluppo di servizi: Vantaggi

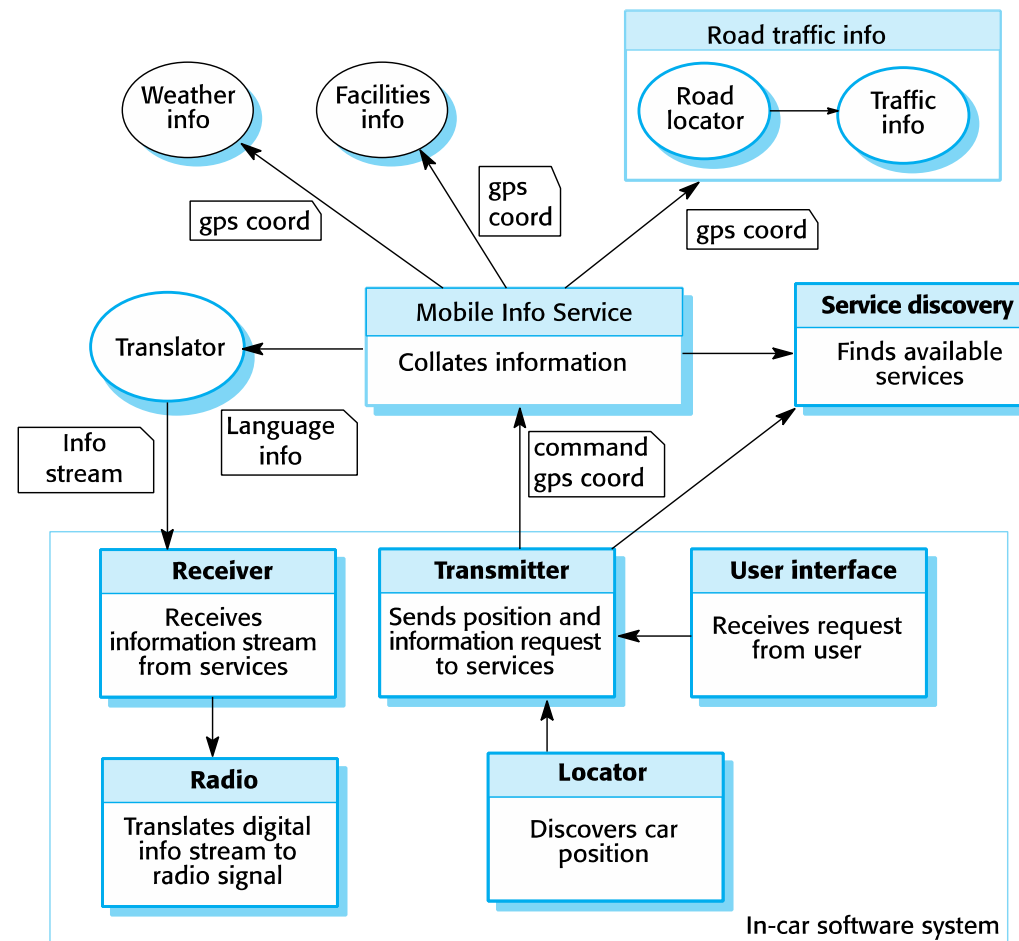
- I servizi possono essere sviluppati *localmente*, o essere prodotti da fornitori esterni.
- Per accedere ai servizi è necessario conoscere unicamente la loro **interfaccia**
 - L'accesso ai servizi è possibile attraverso un linguaggio standard, indipendentemente dal linguaggio con cui i servizi sono stati implementati
- E' possibile recuperare (tramite wrapper) funzionalità di sistemi esistenti (legacy systems).
- Aziende o altre organizzazioni possono cooperare utilizzando funzioni di business altrui e avvalendosi di un meccanismo di scambio dati semplificato: supporto al **Collaborative Computing**

Un esempio di sistema basato su Servizi

- Un sistema informativo per auto deve fornire al conducente informazioni su condizioni meteorologiche, condizioni del traffico stradale, informazioni locali ecc. È collegato all'autoradio in modo che le informazioni vengano trasmesse come segnale su un canale radio specifico.
- L'auto è dotata di ricevitore GPS per scoprire la sua posizione e, in base a tale posizione, il sistema accede a una serie di servizi di informazione. Le informazioni possono essere fornite nella lingua specificata dal conducente.



A service-based, in-car information system



Vantaggi delle SOA per questa applicazione

- Non è necessario decidere al momento dello sviluppo del sistema quale fornitore di servizi dovrebbe essere utilizzato, o quali servizi specifici dovrebbero essere accessibili.
- Mentre l'auto si muove, il software per auto utilizza il servizio di rilevamento del servizio per trovare il servizio di informazioni più appropriato e si lega a quello.
- A causa dell'uso di un servizio di traduzione, può spostarsi oltre i confini e quindi rendere le informazioni locali disponibili per le persone che non parlano la lingua locale.

Principi di Progettazione per i Servizi


Principi di Progettazione per i Servizi

- **Lasco Accoppiamento.** I servizi dovrebbero essere sufficientemente indipendenti da altri.
 - Devono conoscere pochi dettagli di altri servizi e non dovrebbero dipendere da altri.
- **Contratto di servizio.** Un servizio fornisce un contratto ad altri servizi, su cui una applicazione SOA può contare.
 - Il contratto è definito dall'interfaccia del servizio costituita da operazioni, parametri di input e output, o qualità del servizio offerta (come tempo di risposta o disponibilità).

Principi di Progettazione per i Servizi


- **Autonomia.** Ogni servizio è auto-contenuto e può operare indipendentemente da altri servizi.
L'eventuale interazione tra i servizi deve essere lasciata ai *servizi di coordinamento*.
- **Astrazione.** I dettagli implementativi del servizio non sono visibili all'esterno. Solo l'interfaccia è visibile.
- **Riusabilità.** I principi precedenti facilitano il riuso.
- **Componibilità.** I servizi sono progettati in modo da essere facilmente assemblati a formare servizi compositi più complessi.

Principi di Progettazione per i Servizi

- **Indipendenza dallo stato.** I servizi dovrebbero funzionare senza ricordare nulla del passato dei client che li usano. 
- **Scopribilità.** I servizi devono descriversi opportunamente, al fine di permettere di scoprirli da parte di meccanismi di scoperta.

I Pattern nell'Architettura Service Oriented

I Pattern Architetture di tipo Broker

- Le SOA si basano su meccanismi di **Service Brokers**, che permettono di scoprire un servizio e collegarlo ad un servizio client.
 - Il Broker libera il client dall'onere di mantenere informazioni sul servizio.
- Esistono diversi pattern architetture relativi al Broker. Due grandi categorie sono:
- **I Pattern Broker a Pagine Bianche**
 - Il client conosce il servizio di cui ha bisogno, ma non la posizione 
- **I Pattern Broker a Pagine Gialle**
 - Il client conosce solo il tipo del servizio, ma non il servizio specifico


I Pattern in SOA

- Pattern Broker:
 - Yellow Page Brokering
 - White Page Brokering

Il pattern Broker

- Nel pattern **Broker** (anche noto come *Object Broker* o *Object Request Broker* pattern), il **Broker** funge da intermediario fra client e servizi.
 - I servizi si registrano presso il Broker (usando il **Service Registration** pattern).
 - I Client localizzano i servizi attraverso il broker (usando o il **White Page Brokering** o **Yellow Page Brokering** patterns).
- Dopo che il broker ha fornito la connessione, la comunicazione fra client e servizio può avvenire direttamente o attraverso il Broker.

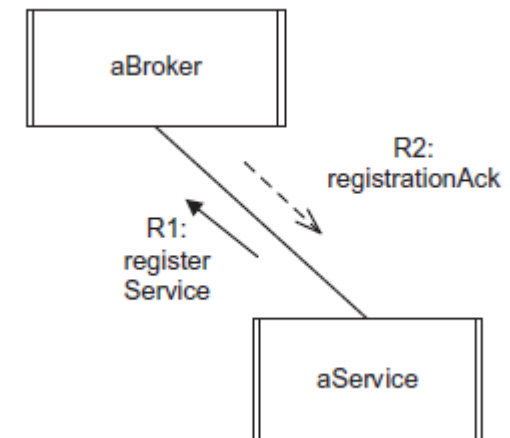
Caratteristiche del Pattern Broker

- Il Broker garantisce diverse proprietà:
- **Location transparency:** se il servizio viene spostato su un'altro nodo, I client non devono essere informati dello spostamento e solo il Broker deve esserlo. 
- **Platform transparency:** ogni servizio può essere eseguito su una diversa piattaforma hardware/software e non ha bisogno di mantenere informazioni sulle piattaforme su cui sono eseguiti gli altri servizi.

Service Registration Pattern



- Il servizio si registra presso il Broker fornendo:
 - *Nome* del servizio
 - *Descrizione* del servizio
 - *Posizione* in cui il servizio è disponibile.
- **R1:** Il Servizio manda una richiesta di Servizio di Registrazione al broker.
- **R2:** Il broker registra il servizio nel registro e spedisce un acknowledgment al servizio.



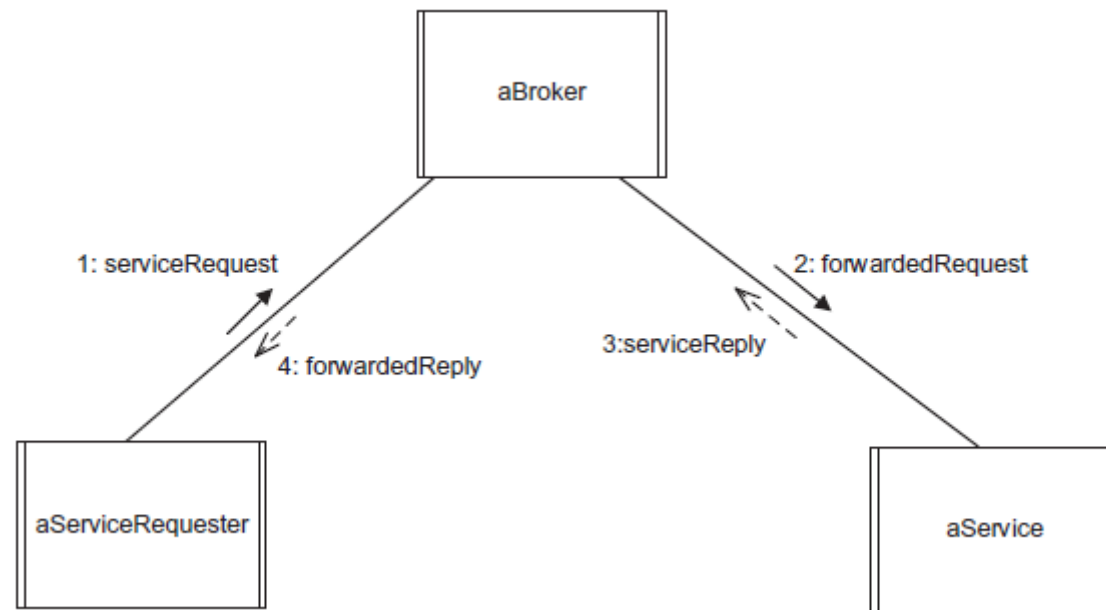
Broker Forwarding Pattern



- È uno dei possibili modi usabili dal broker per rispondere alle richieste dei client.
- Il Broker riceve dal client la richiesta del servizio e la inoltra al servizio stesso.
- Quando la risposta è disponibile, il Broker la re-inoltra al client.



Broker Forwarding Pattern



Caratteristiche del « Broker Forwarding » pattern

- Fornisce una intermediazione per ogni messaggio fra client e service.
- Potenzialmente fornisce un alto livello di sicurezza, perchè ogni messaggio può essere controllato.
- C'è però un peggioramento delle prestazioni (4 messaggi necessari invece dei 2 richiesti dal classico pattern Client/Service).

Il Pattern «Broker Handle»

- Il pattern **Broker Handle** conserva i vantaggi della trasparenza della posizione del servizio, riuscendo anche a ridurre il traffico di messaggi.
- Invece di inoltrare ogni messaggio del client al servizio, il broker restituisce un *handle* del servizio al client, che viene usato per comunicazioni dirette tra client e servizio.
- E' molto utile quando client e servizio devono dialogare scambiando molti messaggi.
- Quando il servizio si sposta, deve informare il broker della nuova posizione.

II pattern Broker Handle (White pages)

- **B1:** The client (service requester) sends a service request to the broker.
- **B2:** The broker looks up the location of the service and returns a service handle to the client.
- **B3:** The client uses the service handle to make the request to the appropriate service.
- **B4:** The service executes the request and sends the reply directly to the client.

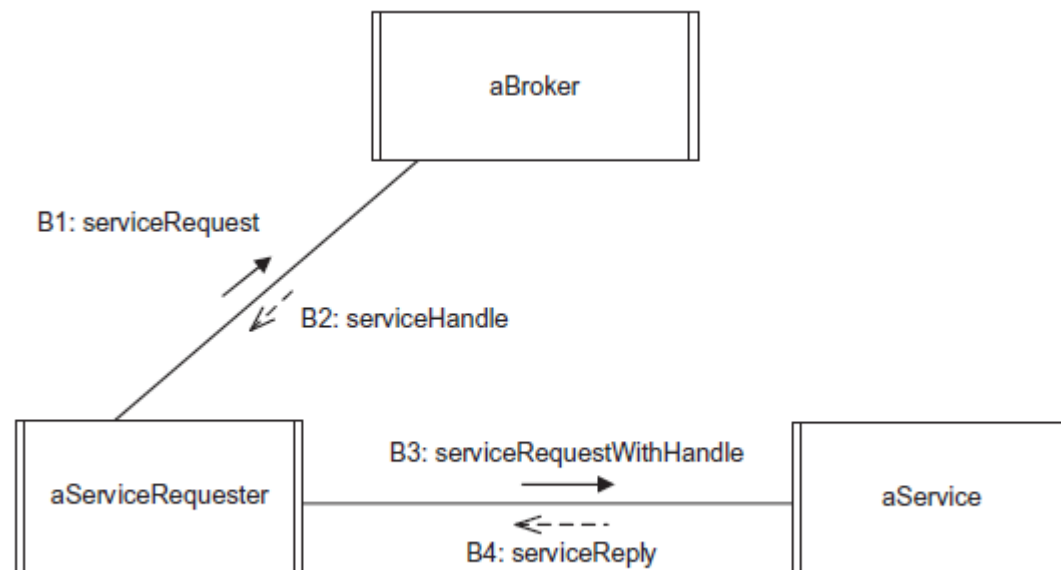
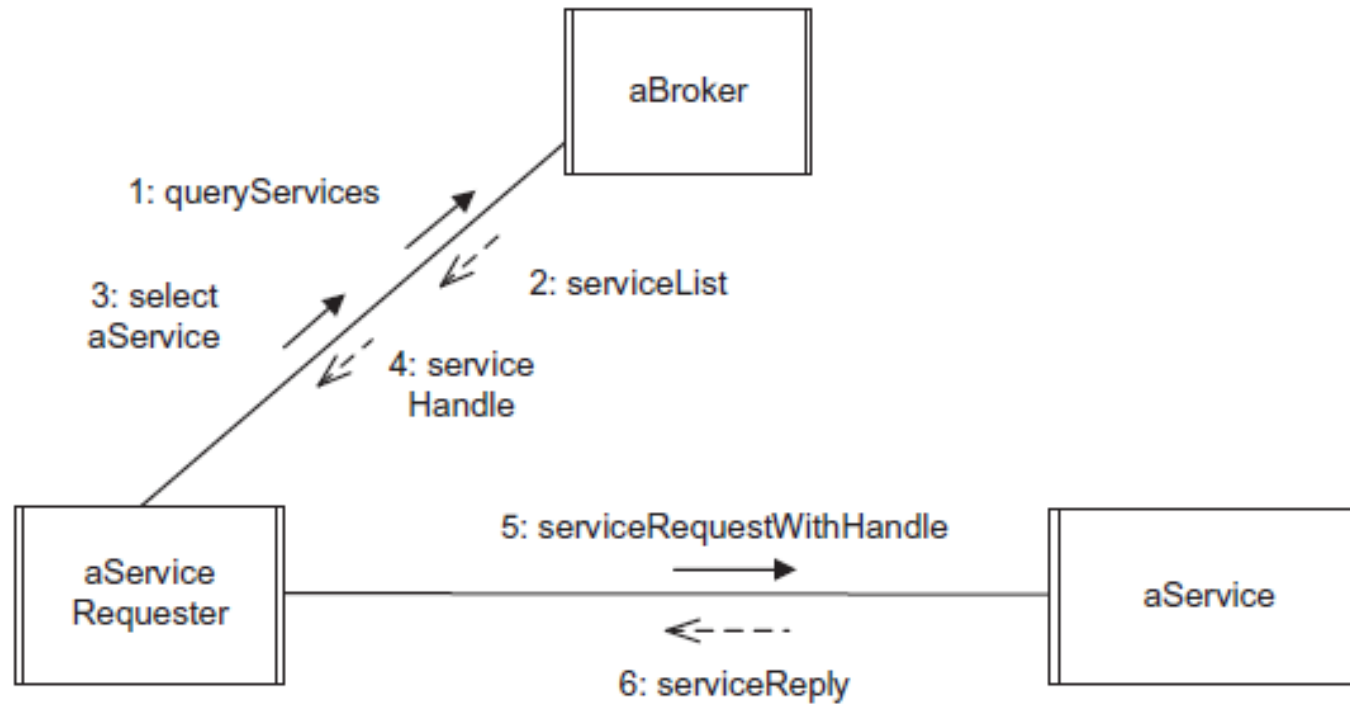


Figure 16.3. Broker Handle (white pages) pattern

Il Service Discovery Pattern (Yellow Pages)

- Quando il pattern broker è di tipo *pagine gialle*, il client spedisce una richiesta di interrogazione al broker, chiedendogli tutti i servizi di un certo tipo.
- Il broker risponde con un elenco di servizi che soddisfano la richiesta.
- Il client, dopo aver consultato l'utente, seleziona un servizio. Il broker restituisce l'handle del servizio, che il client userà per comunicare direttamente col servizio.

II Service Discovery Pattern



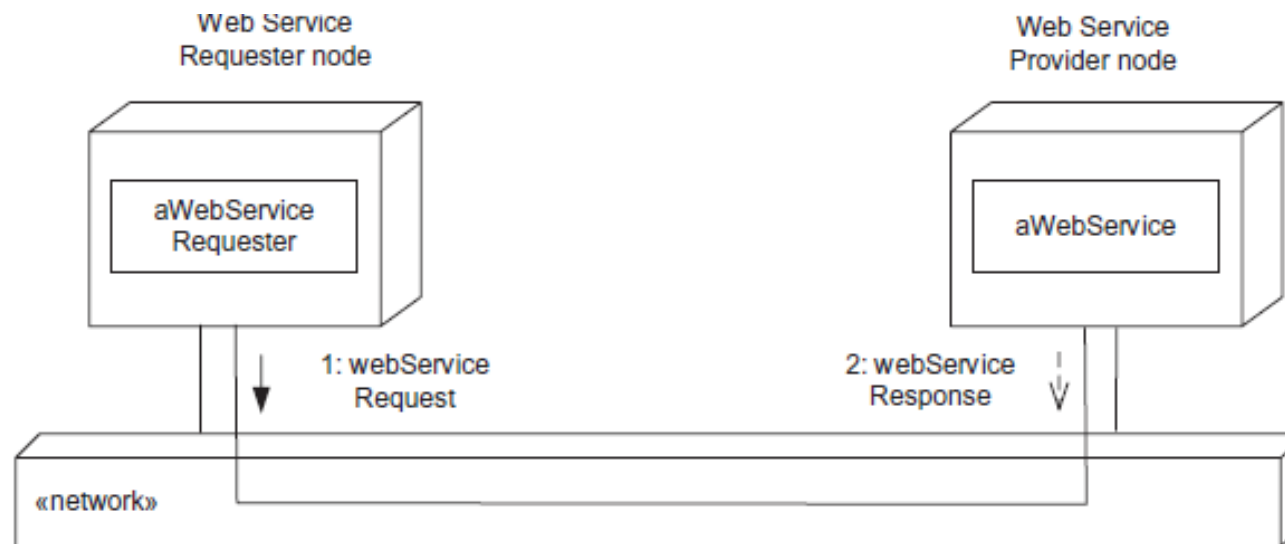
Implementazione di SOA con Web Services

Implementazione di SOA attraverso i Web Services

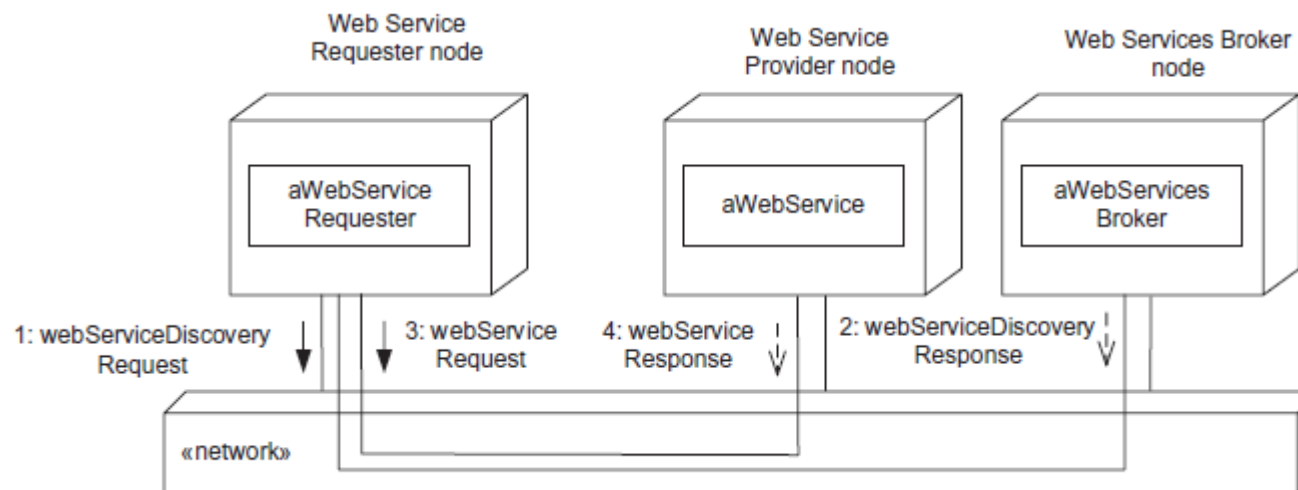
- Una possibile implementazione di architetture SOA è quella offerta dai Web Services.
 - I “**Web services**” rappresentano uno sforzo verso la costruzione di una piattaforma di calcolo distribuita basata sull'infrastruttura offerta dal Web
- Un Web service è un servizio che può essere acceduto, pubblicato, localizzato, ed invocato via Internet, usando protocolli standard Internet e XML-based.

Web Service

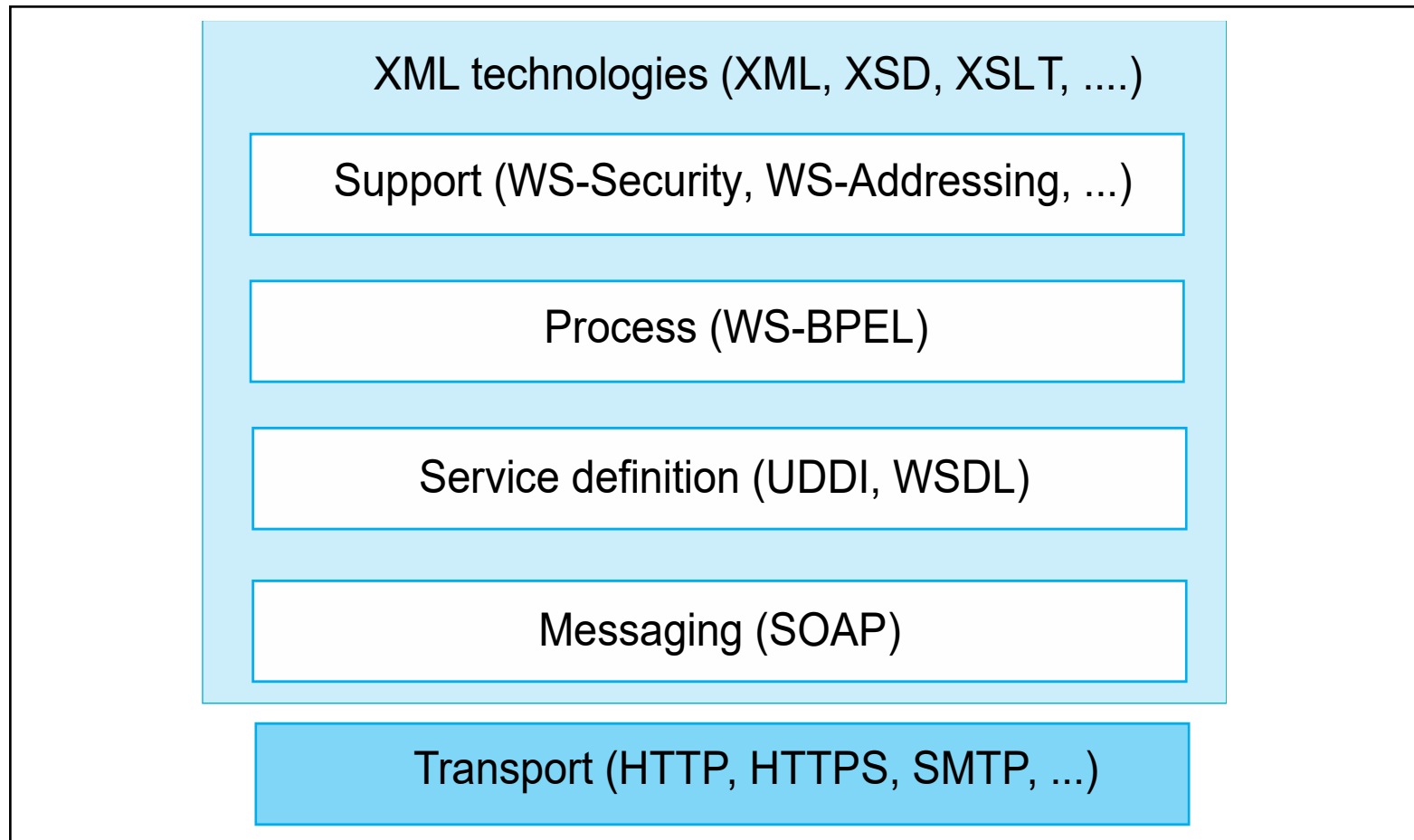
- Da una prospettiva software, i web service sono API che forniscono un modo standard per far comunicare applicazioni software sul World Wide Web.
- La comunicazione si basa su *scambio di messaggi asincroni*, XML –based, e protocolli standard.



- I Web Service hanno bisogno di:
- Servizi di Registrazione
 - Per registrarsi presso un Registro di Servizi
- Servizi di Brokering e di Discovery



Stack e protocolli per i Web Services



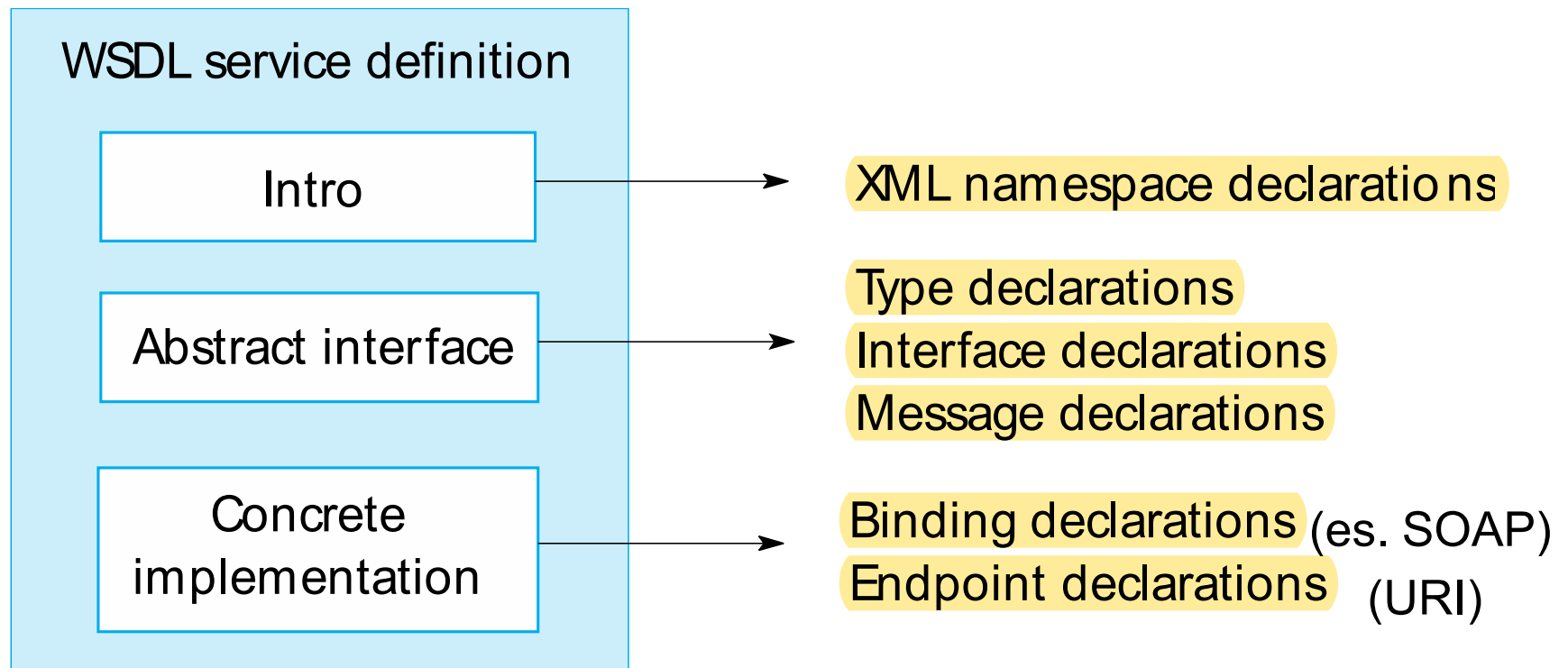
Alcuni standard

- **SOAP**
 - Protocollo per lo scambio di messaggio, tipicamente implementato su http o https
- **WSDL** (*Web Service Definition Language*)
 - Protocollo per lo scambio di informazioni descrittivi l'interfaccia del Web Service e il collegamento con il servizio vero e proprio
- **UDDI** (*Universal Description, Discovery, and Integration*)
 - Definisce la semantica del servizio, in maniera tale da consentire la scoperta automatica dell'esistenza del servizio
- **WS-BPEL**
 - Permette di definire la modalità di composizione di servizi

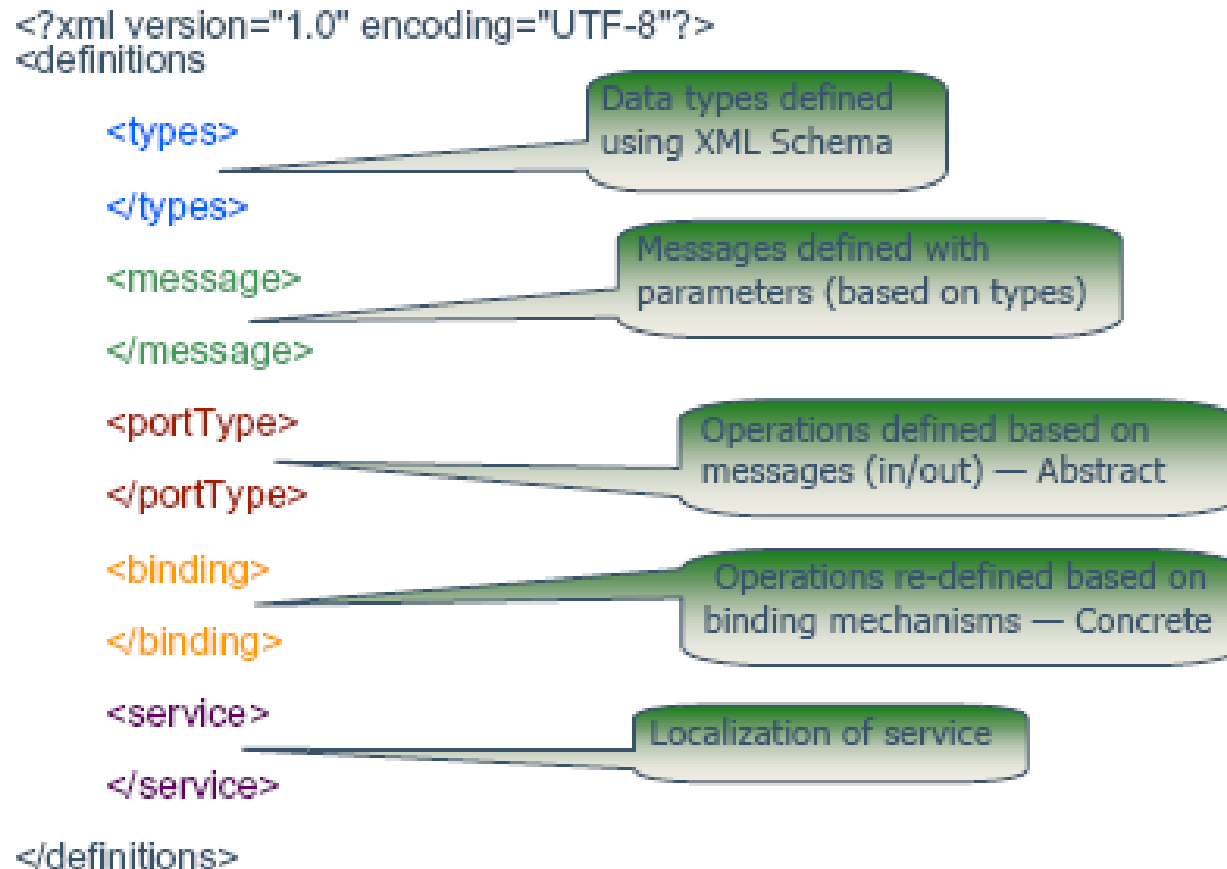
Web Service Description Language (WSDL)

- Le interfacce dei servizi sono espresse in linguaggio WSDL. Una specifica WSDL definisce
 - **Quali** operazioni sono fornite dal servizio (Interfaccia astratta)
 - Per ogni operazione è specificato il formato del messaggio di richiesta e il formato del messaggio di risposta
 - **Come** accedere al servizio
 - Binding: specifica come devono essere costruiti i messaggi di I/O ed il protocollo da seguire nello scambio di messaggi
 - **Dove** si trova il servizio, in termini di URI (Universal Resource Identifier)
 - URI è l'indirizzo di una risorsa accessibile via Internet ed è una generalizzazione del concetto di indirizzo web

Struttura di una specifica WSDL



Struttura XML di una specifica WSDL



Un frammento di specifica WSDL (1/2)

Define some of the types used. Assume that the namespace prefixes 'ws' refers to the namespace URI for XML schemas and the namespace prefix associated with this definition is weathns.

```
<types>
  <xs: schema targetNamespace = "http://.../weathns"
    xmlns: weathns = "http://.../weathns" >
    <xs:element name = "PlaceAndDate" type = "pdrec" />
    <xs:element name = "MaxMinTemp" type = "mmtrec" />
    <xs: element name = "InDataFault" type = "errmess" />

    <xs: complexType name = "pdrec"
      <xs: sequence>
        <xs:element name = "town" type = "xs:string"/>
        <xs:element name = "country" type = "xs:string"/>
        <xs:element name = "day" type = "xs:date" />
      </xs:complexType>

      Definitions of MaxMinType and InDataFault here
    </schema>
</types>
```


Un frammento di specifica WSDL (2/2)

Now define the interface and its operations. In this case, there is only a single operation to return maximum and minimum temperatures

```
<interface name = "weatherInfo" >  
  <operation name = "getMaxMinTemps" pattern = "wsdl:in-out">  
    <input messageLabel = "In" element = "weathns:PlaceAndDate" />  
    <output messageLabel = "Out" element = "weathns:MaxMinTemp" />  
    <outfault messageLabel = "Out" element = "weathns:InDataFault" />  
  </operation>  
</interface>
```

RESTful Web Services

- Gli attuali standard dei servizi Web sono stati criticati come standard "pesanti" , troppo generali e inefficienti.
- REST (REpresentational State Transfer) è uno stile architettonico basato sul trasferimento di rappresentazioni di risorse da un server a un client.
- Questo stile è alla base del web nel suo insieme ed è più semplice di SOAP / WSDL per l'implementazione dei servizi web.
- I servizi RESTFul comportano un sovraccarico inferiore rispetto ai cosiddetti "grandi servizi web" e sono utilizzati da molte organizzazioni che implementano sistemi basati su servizi che non si basano su servizi forniti esternamente nè usano il binding dinamico.

Stile RESTful

- Si basa sul trasferimento delle rappresentazioni delle risorse da un server ad un client.
- La risorsa (un elemento di dati) esiste indipendentemente dalla sua rappresentazione (es. una risorsa Capitolo Libro, può avere 3 diverse rappresentazioni (Pdf, Word, RTF).
- In REST ogni risorsa ha un identificatore univoco, la URL .
- Ad ogni risorsa sono associate 4 possibili operazioni: creare, leggere, aggiornare, modificare (POST, GET, SET, DELETE)

Esempi di servizi REST

- I servizi REST accedono direttamente ai dati (mentre i WS Soap svolgono azioni su DB remoti), sfruttando i protocolli http, https
- Es.
 - `http://weather-info-example.net/temperatures/boston`
(Richiama l'operazione GET)
 - `http://weather-info-example.net/temperatures/boston?date=20140226 & country=USA`
 - (usa una query per specificare i dati richiesti)

Differenze con SOAP

- I servizi RESTful non sono vincolati ad usare la rappresentazione XML dei dati, ma possono anche usarne altre, come ad es. JSON (Javascript Object Notation) che sono ottimizzate e danno minore overhead
- Molto utile per app mobili
- Es. JSON response:
- ```
{ "place": "Boston",
 "state": "Mass",
 "date": "26 Feb 2014",
 "unit": "Fahrenheit",
 "maxtemp": 41,
 "mintemp": 29,
}
```

## Problemi con REST

- Quando un servizio ha un'interfaccia complessa e non fornisce una risorsa semplice, può essere difficile progettare un insieme di servizi REST per implementare tale interfaccia
- Non esistono standard per la descrizione di interfacce RESTful (bisogna usare documentazione informale delle interfacce)
- Quando si usano servizi RESTful, bisogna creare una propria infrastruttura per monitorare e gestire la qualità e affidabilità dei servizi (mentre SOAP ha appositi standard per questo)

# Altri Patterns per SOA

# Patterns per i Servizi di accesso ai dati

- I servizi spesso **incapsulano dati** o forniscono accesso a dati che devono essere letti /aggiornati dai client.
- Molti servizi devono offrire operazioni di aggiornamento dati opportunamente coordinate.
- Alcuni **pattern architetturali transazionali** possono essere usati a tal fine.



# Transazioni

- Una transazione è una richiesta di un servizio che consiste di due o più operazioni che insieme svolgono una singola funzionalità, e che deve essere completata interamente o per niente.
- Le transazioni sono generate presso il client e vengono spedite al servizio per poter essere eseguite.
- Per le transazioni atomiche, (i.e., indivisibili), sono necessari servizi per iniziare la transazione, fare il *commit*, oppure *l'abort* della transazione.

- Per completare una transazione occorre eseguire correttamente tutte le operazioni incluse.
  - Se una operazione fallisce, la transazione deve essere abortita (abort o roll back) e le eventuali operazioni già svolte devono essere annullate.
- Es. Una transazione che coinvolge più conti correnti, uno da cui si preleva e l'altro su cui si deposita.

# Proprietà ACID delle transazioni

- **Atomicity (A).** Una transazione è una unità indivisibile di lavoro. O viene completata interamente (committed) o è annullata (rolled back).
- **Consistency (C).** Dopo l'esecuzione della transazione, il sistema deve essere in uno stato consistente, ossia tutti i vincoli di integrità noti devono essere rispettati
- **Isolation (I).** Il comportamento di una transazione non deve essere influenzato da altre transazioni.
- **Durability (D).** Le modifiche devono essere permanenti dopo l'esecuzione della transazione. Le modifiche devono sopravvivere ad eventuali failures del sistema (concetto di persistenza).

# Il Pattern del Protocollo Two-Phase Commit

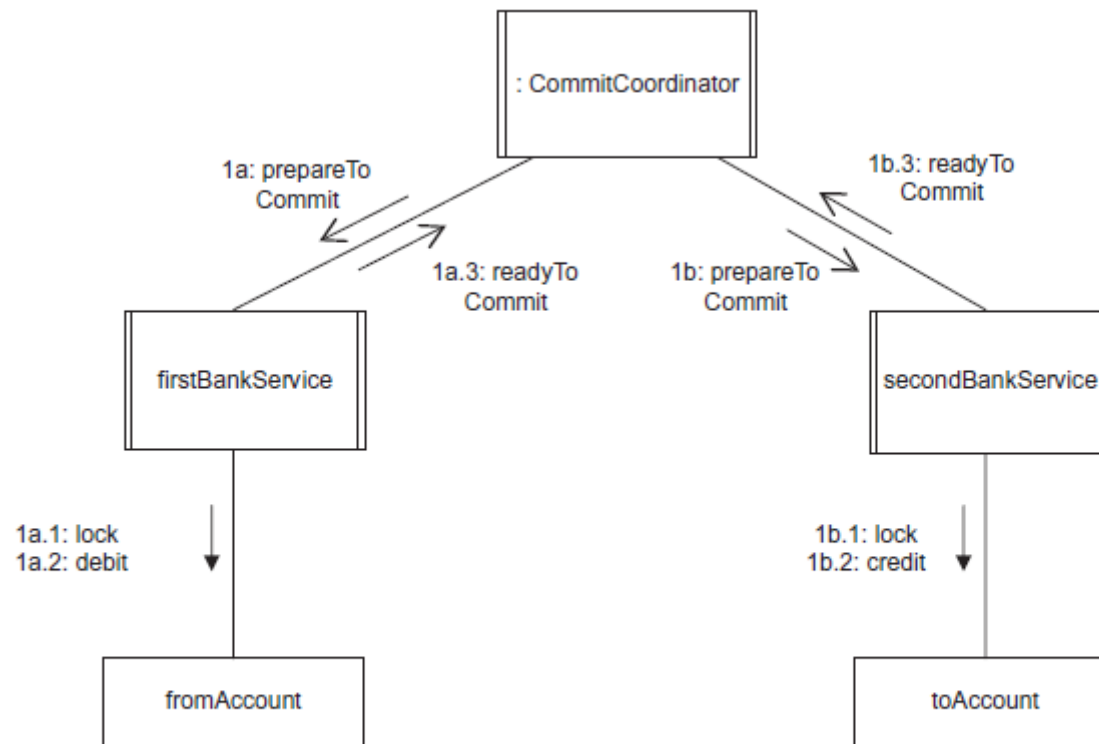
- Affronta il problema di gestire transazioni atomiche in sistemi distribuiti. Esempi in ambito bancario:
- **Transazione di Prelievo.** Una transazione di prelievo può essere svolta in una sola operazione. E' necessario un semaforo per assicurare che l'accesso sia in mutua esclusione.
  - L'esecutore della transazione blocca il record del conto per il cliente, esegue l'aggiornamento e quindi sblocca il record.
- **Transazione di trasferimento fra conti:** richiede due operazioni (1 prelievo e un accredito su conto) che devono essere entrambe svolte con successo

## Esempio: Transazione di trasferimento

- Il coordinamento della transazione è fornito da un **coordinatore** dei servizi partecipanti :
- Ci sono due servizi partecipanti:
  - 1 Servizio della prima banca che gestisce il conto su cui si preleva (from Account),
  - 1 Servizio della seconda banca che gestisce il conto destinazione (to Account).
- **Nella prima fase** del protocollo, il coordinatore invia I messaggi di *Prepare to Commit* ai due servizi partecipanti (1a, 1b). Ciascun partecipante al servizio blocca il record, esegue l'aggiornamento e spedisce un messaggio di *Ready to Commit* al coordinatore.

# Prima Fase (Prepare to Commit)

- Se un partecipante non riesce a svolgere l'aggiornamento, manda un messaggio di Rifiuto di Commit. Il Coordinatore di Commit aspetta le risposte da tutti i partecipanti.



**Figure 16.7.** Example of the first phase of the Two-Phase Commit Protocol: bank transfer

## Seconda Fase (Commit)

- Se tutti i partecipanti spediscono il messaggio di *ready To Commit*, il Coordinatore spedisce il messaggio di *commit* (2a, 2b) a ciascun servizio partecipante.
- Ciascun partecipante rende l'aggiornamento permanente (2a.1, 2b.1), sblocca i record (2a.2, 2b.2), e spedisce un messaggio di *commit Completato* (2a.3, 2b.3) al Coordinatore.
- Il Commit Coordinator attende tutti I messaggi di *commit Completato*.

## Esempio della Seconda fase

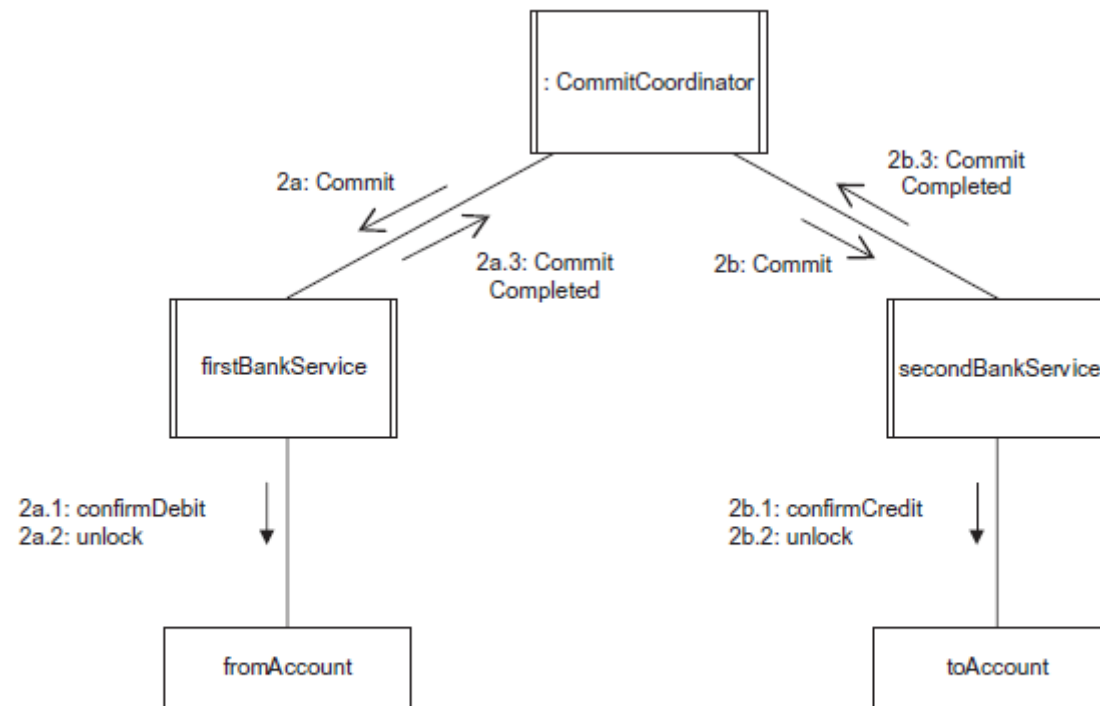


Figure 16.8. Example of the second phase of the Two-Phase Commit Protocol: bank transfer



## Eccezioni

- Se un servizio partecipante risponde al messaggio di *prepare To Commit* con un messaggio di *ready To Commit*, esso si impegna a completare la transazione. Il servizio partecipante deve completare la transazione anche se subentra qualche problema
- Se, nella prima fase, qualche servizio partecipante risponde al messaggio di *Prepare To Commit* con un messaggio di *refuse To Commit*, il Coordinatore spedisce un messaggio di *abort* a tutti i partecipanti. I partecipanti quindi annullano l'aggiornamento (abort).
- Eventuali problemi nella seconda fase (es. Timeout) si possono risolvere usando i log files per eseguire l'unroll della transazione

## Il Pattern di Negoziazione

- In alcune SOA, il coordinamento fra servizi può richiedere la *negoziazione dei servizi* fra diversi agenti software.
  - Un *agente client* esegue la negoziazione al posto dell'utente, richiedendo servizi ad un *agente server*.
  - L'agente server cerca di soddisfare le richieste del client, comunicando con altri servizi.
  - L'agente server, ottenute le opzioni dei servizi disponibili, le inoltra al client, che può: a) richiedere uno dei servizi disponibili, b) proporre altre opzioni, c) rifiutare le offerte.
- Se l'agente di servizio può soddisfare la richiesta del client, esso può accettarla, altrimenti la rifiuta.

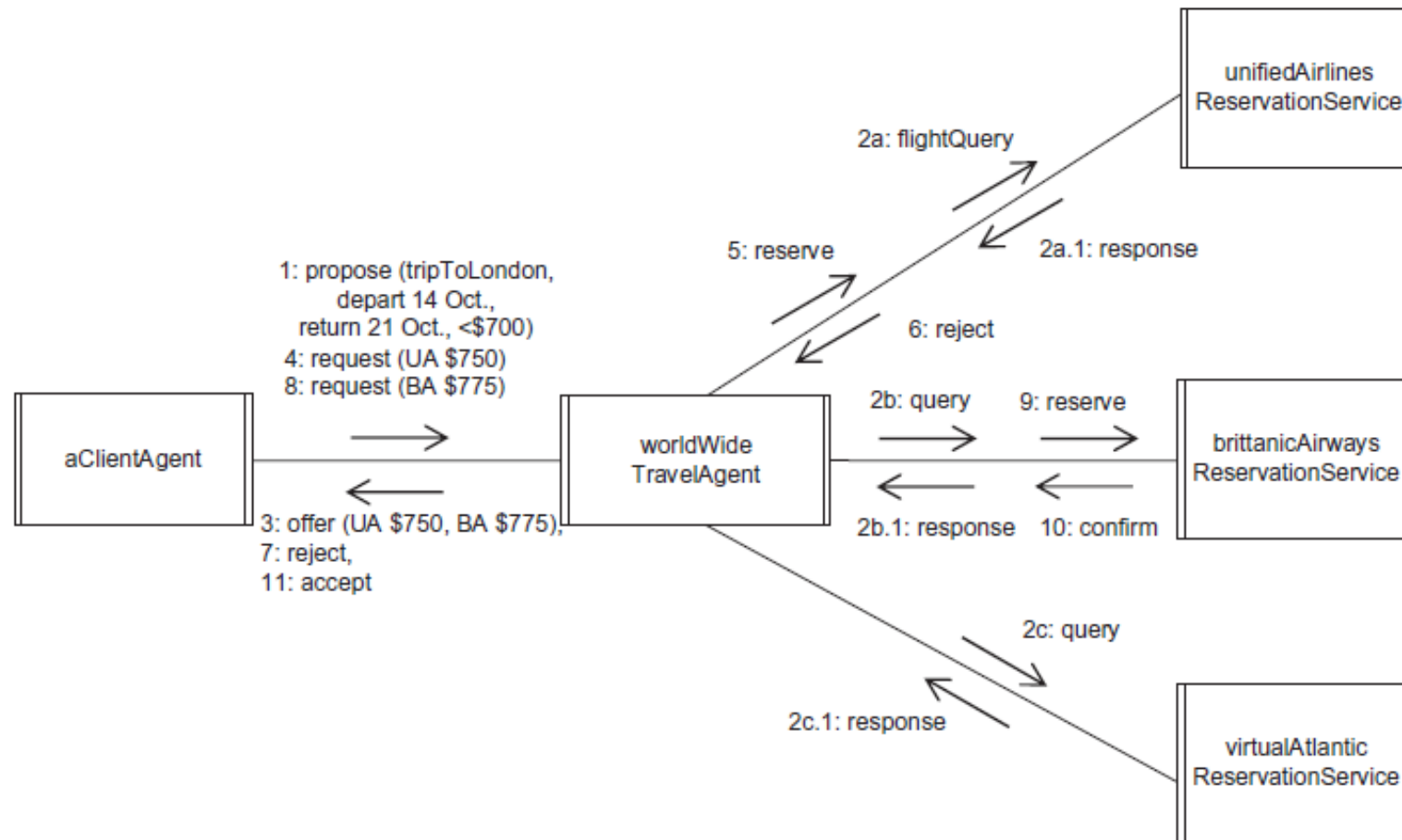
# Comportamenti degli agenti client e di servizio

- L'agente Client può:
- **Proporre un servizio** all'agente di servizio. Il servizio è negoziabile (ossia il client potrà considerare eventuali contro-offerte).
- **Richiedere un servizio.** Il client agent richiede un servizio *non-negoziabile* all'agente server, ossia senza intendere di considerare contro-offerte.
- **Rifiutare un'offerta di servizio.** L'agente client rifiuta un'offerta del server agent.

# Comportamenti degli agenti di servizio

- L'agente di servizio, agendo al posto del servizio, può:
- **Offrire un servizio.** In risposta ad una proposta del client, esso può fare una contro-proposta.
- **Rifiutare una richiesta/ proposta del client.**  
L'agente di servizio rifiuta il servizio proposto o richiesto dall'agente client.
- **Accettare una richiesta/proposta del client.**  
L'agente di servizio accetta il servizio proposto o richiesto dall'agente client.

# Esempio di Negoziazione





# Ingegnerizzazione dei Servizi

# Argomenti

- Ingegnerizzazione dei Servizi (Service Engineering)
- Sviluppo di software basato su Servizi



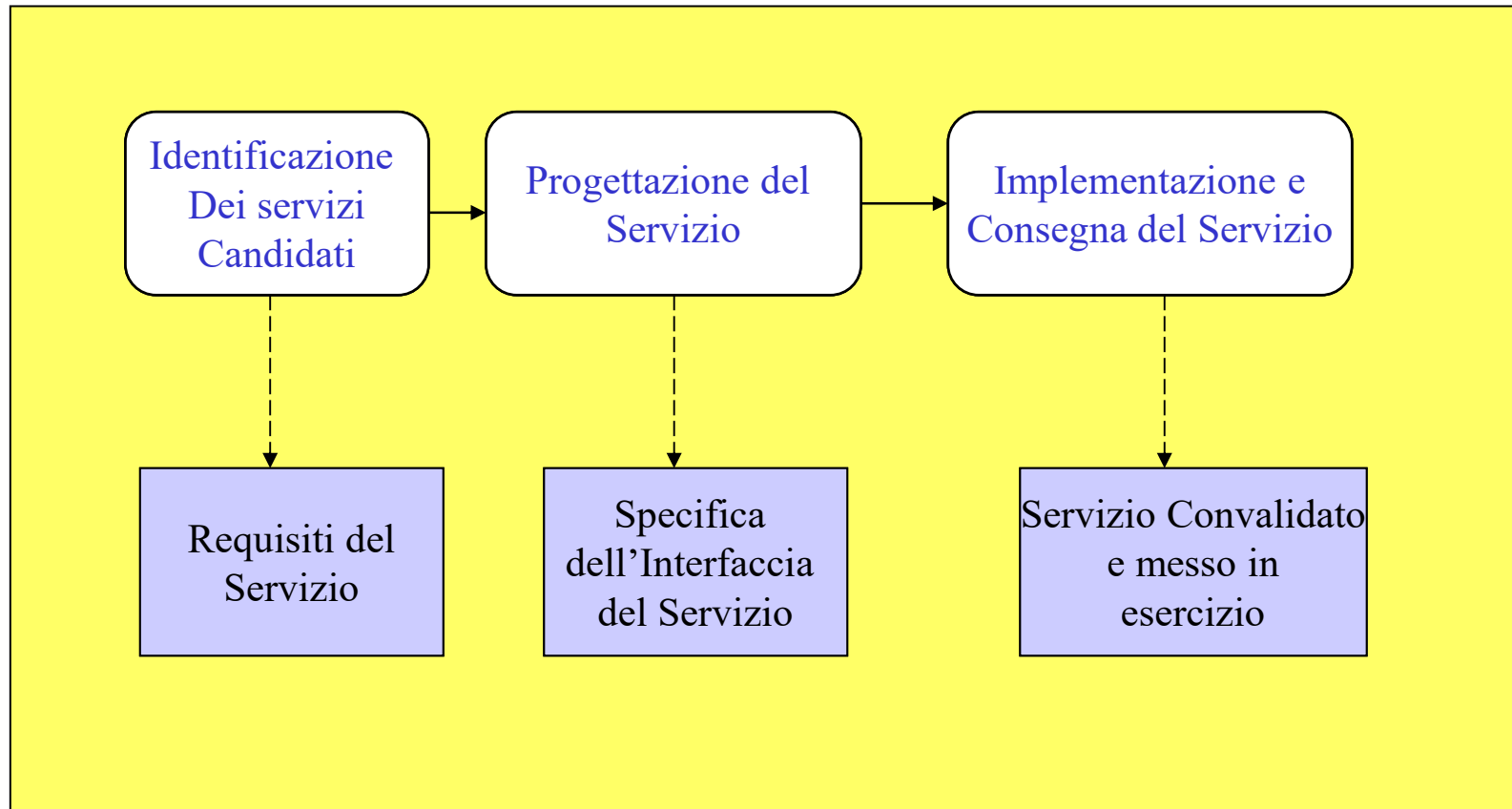
# Service-oriented software engineering

- Gli approcci di ingegneria del software esistenti devono necessariamente cambiare per tener conto dell'approccio di sviluppo software service-oriented.
- Due temi fondamentali:
  - **Service engineering.** Tratta lo sviluppo di servizi 'dependable' e riusabili
    - Sviluppo di software *per* il riuso
  - **Sviluppo Software basato su servizi.** Si occupa di sviluppare software fidato riusando servizi software
    - Sviluppo di software *con* il riuso

# Service engineering

- É il processo di sviluppo di servizi riusabili per applicazioni service-oriented.
- Il servizio deve essere progettato come una astrazione riusabile nell'ambito di diversi sistemi.
- Richiede
  - L'identificazione del servizio candidato
  - La progettazione del servizio
  - L'implementazione e consegna del servizio

# Il Processo di Service Engineering



# (1): Identificazione dei servizi candidati

- I servizi possono essere classificati in tre categorie fondamentali:
  - **Servizi di utilità** che implementano funzionalità di carattere generale, applicabili a diversi problemi
  - **Servizi di business**, utili all'interno di una particolare tipologia di dominio aziendale
  - **Servizi di coordinazione** che supportano la composizione di servizi complessi
    - Interagiscono con i servizi più generali, consentendo di ricavarne astrazioni di domini
- Un'altra classificazione distingue tra servizi orientati alle **attività (task)** o alle **entità**

# Esempi di servizi

|               | <b>Utility</b>                                                                  | <b>Business</b>                                             | <b>Coordination</b>                      |
|---------------|---------------------------------------------------------------------------------|-------------------------------------------------------------|------------------------------------------|
| <b>Task</b>   | Convertitore di valuta .<br>Localizzatore di impiegati                          | Convalida richiesta rimborso.<br>Verifica stato del credito | Gestione rimborsi<br>Pagamenti fornitori |
| <b>Entity</b> | Verificatore di validità di un documento.<br>Convertitore di un Web form in XML | Modulo Spese<br><br>Modulo di richiesta studente            |                                          |

# Domande utili per la ricerca di servizi riusabili

- Nel caso di un servizio orientato alle entità, il servizio è associato ad una singola entità logica che è usata in diversi processi di business?
- Nel caso di un servizio orientato alle attività, tale attività è eseguita da più persone di una organizzazione?
- Il servizio è indipendente, o fino a che punto ha bisogno di altri servizi?
- Il servizio ha bisogno di mantenere aggiornato uno stato interno? a tal fine, ha bisogno di un database?
- Il servizio potrebbe essere usato anche all'esterno, da più clienti?
- É probabile che clienti diversi del servizio richiedano diversi requisiti non funzionali ad esso?

# Il servizio di un Catalogo

- É un esempio di servizio orientato alle **entità** che supporta operazioni di **business**. Le operazioni devono permettere ad un produttore di articoli di mostrare il catalogo degli articoli acquistabili (es. Computer) dai suoi clienti
- Requisiti Funzionali del servizio
  - Una versione specifica del catalogo dovrebbe essere creata per ciascun cliente (con prezzi e config. concordate)
  - Deve essere possibile fare il download del catalogo
  - Si devono confrontare specifiche e prezzi di fino a 6 articoli del catalogo
  - Devono essere fornite funzionalità di ricerca e Browsing
  - Deve esserci una funzione che permette di stimare la data di consegna per gli articoli indicati
  - Gli utenti devono poter fare ordini virtuali da confermare entro 48 ore con un ordine reale.

# Il Servizio di un Catalogo

- Requisiti non funzionali del servizio
  - L'accesso al catalogo deve essere consentito solo agli impiegati di organizzazioni accreditate.
  - I prezzi e le configurazioni offerte ai vari clienti saranno confidenziali
  - Il catalogo deve essere disponibile dalle 07:00 alle 11:00
  - Il catalogo deve essere in grado di elaborare fino a 10 richieste al secondo.



# Descrizione delle Operazioni del Servizio di Catalogo

| Operazione          | Descrizione                                                                                                                                                |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CreaCatalogo        | Crea una versione del catalogo personalizzata per ciascun cliente. Prevede un parametro opzionale per creare una versione scaricabile in PDF del catalogo. |
| Confronta           | Confronta fino a 6 caratteristiche (e.g. price, dimensions, processor speed, etc.) di fino a 4 articoli del catalogo.                                      |
| Visualizza          | Mostra tutti i dati associati ad uno specifico elemento del catalogo.                                                                                      |
| Ricerca             | Permette di eseguire una ricerca di articoli che soddisfano una certa espressione di ricerca.                                                              |
| DataConsegna        | Fornisce la data stimata per la consegna di un articolo (qualora fosse ordinate oggi).                                                                     |
| CreaOrdine Virtuale | Prenota gli articoli ordinati e fornisce i dati dell'ordine al sistema per gli acquisti usato dal cliente.                                                 |

## (2) Progettazione dell'Interfaccia del servizio

- Richiede la definizione delle operazioni associate al servizio e dei loro parametri.
- Il numero di messaggi scambiati per completare una richiesta di servizio dovrebbe essere minimo.
- I servizi devono essere senza stato: pertanto potrà essere necessario passare informazioni sullo stato (in ingresso o uscita) per mezzo dei messaggi di input/output.

—

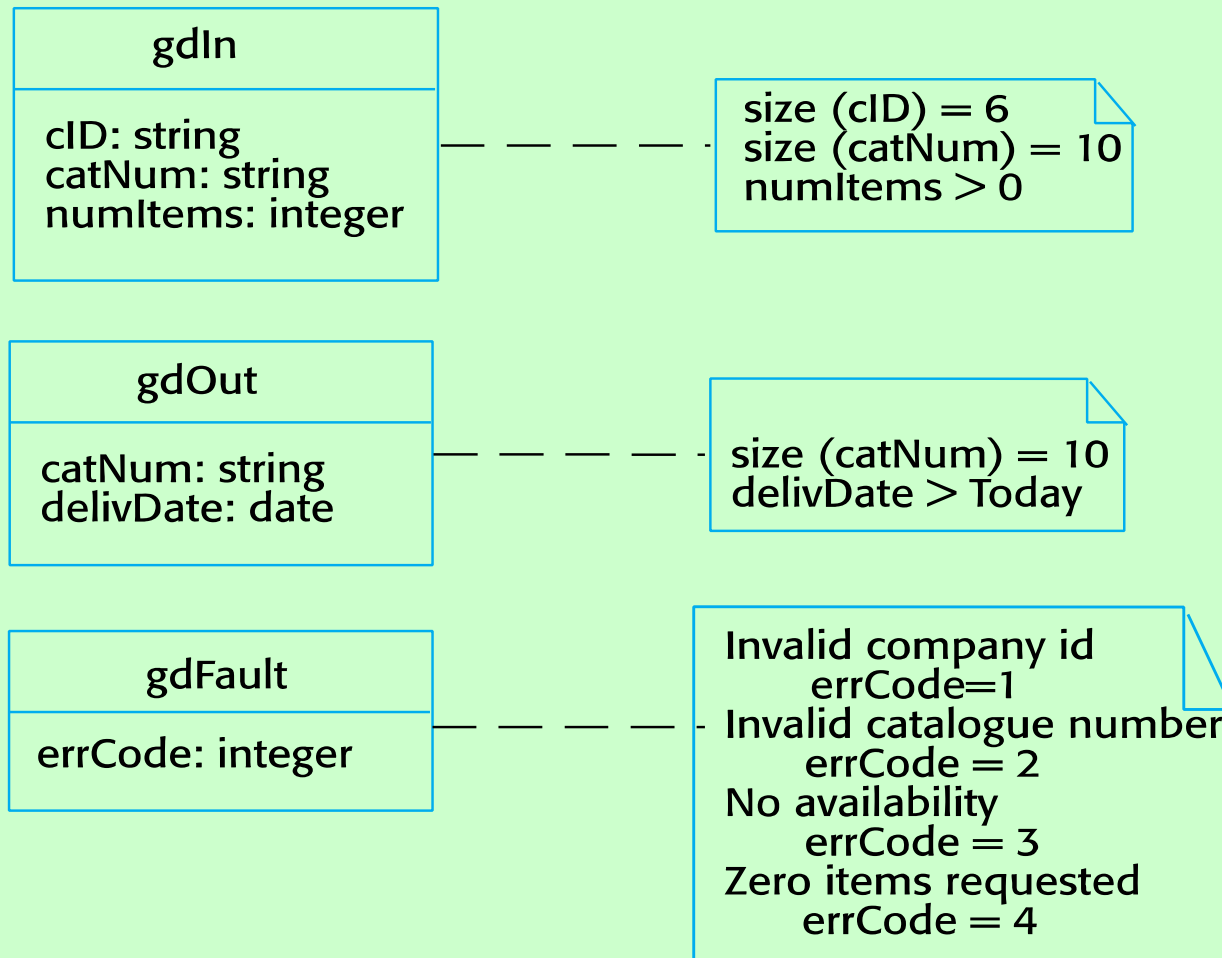
# Le fasi di progettazione dell'Interfaccia

- Logical interface design
  - Si identificano I nomi delle operazioni associate al servizio, i parametri di input e output, e le eventuali eccezioni.
- Message design
  - Progettazione dei messaggi (struttura dei messaggi in input ed output, ed il tipo dei parametri). Preferibile l'uso di linguaggi come l'UML (da tradurre poi in XML) piuttosto che direttamente l'XML.
- WSDL description
  - La specifica logica dell'interfaccia del servizio è tradotta in WSDL.

# Progettazione dell'Interfaccia del catalogo

| Operation           | Inputs                                                                                 | Outputs                                                                                                                                | Exceptions                                                                                                  |
|---------------------|----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| CreaCatalogo        | <i>mcIn</i><br>Company id<br>PDF-flag                                                  | <i>mcOut</i><br>URL of the catalogue for that company                                                                                  | <i>mcFault</i><br>Invalid company id                                                                        |
| Confronta           | <i>compln</i><br>Company id<br>Entry attribute (up to 6)<br>Catalogue number (up to 4) | <i>compOut</i><br>URL of page showing comparison table                                                                                 | <i>compFault</i><br>Invalid company id<br>Invalid catalogue number<br>Unknown attribute                     |
| Visualizza          | <i>lookIn</i><br>Company id<br>Catalogue number                                        | <i>lookOut</i><br>URL of page with the item information                                                                                | <i>lookFault</i><br>Invalid company id<br>Invalid catalogue number                                          |
| Ricerca             | <i>searchIn</i><br>Company id<br>Search string                                         | <i>searchOut</i><br>URL of web page with search results                                                                                | <i>searchFault</i><br>Invalid company id<br>Badly-formed search string                                      |
| DataConsegna        | <i>gdIn</i><br>Company id<br>Catalogue number<br>Number of items required              | <i>gdOut</i><br>Catalogue number<br>Expected delivery date                                                                             | <i>gdFault</i><br>Invalid company id<br>Invalid catalogue number<br>No availability<br>Zero items requested |
| CreaOrdine Virtuale | <i>poIn</i><br>Company id<br>Number of items required<br>Catalogue number              | <i>poOut</i><br>Catalogue number<br>Number of items required<br>Predicted delivery date<br>Unit price estimate<br>Total price estimate | <i>poFault</i><br>Invalid company id<br>Invalid catalogue number<br>Zero items requested                    |

# Struttura dei messaggi di Input e output per l'operazione CheckDelivery



### (3) Implementazione e consegna del Servizio

- Il servizio può essere **implementato** usando linguaggi standard di programmazione (come C# o Java) che forniscono librerie per lo sviluppo di servizi.
- Alternativamente, I servizi possono essere implementati usando componenti preesistenti, sistemi ereditati, o definendo una composizione di servizi pre-esistenti con linguaggi di workflow.
- I servizi devono essere poi **testati** partizionando gli input, creando vari messaggi di input, e verificando che I messaggi di output prodotti siano corretti rispetto alla specifica.
- La **consegna** richiede che il servizio sia pubblicizzato usando UDDI e che venga installato su un web server. Molti server forniscono supporto per una semplice installazione del servizio.

## Ulteriori caratteristiche

- I Web Service mostrano solamente l'interfaccia dei servizi forniti, mentre non dovrebbero aver bisogno di altre risorse
  - Se il Web Service dovesse aver bisogno di dati persistenti, nel suo codice dovrà essere inglobato il codice per l'accesso alla risorsa (ad esempio un database)
  - In questo modo il WS può operare da proxy nell'accesso al database
- I Web Service possono mantenere lo stato unicamente sfruttando variabili di sessione e le altre possibilità messe a disposizione dai protocolli della famiglia http

# Application server

- I Web Service devono essere installati (deployed) all'interno di un application server
  - L'application server, analogamente ad un web server, ascolta su alcuni porti le richieste http in arrivo
  - Le richieste codificate con linguaggi come SOAP vengono processate:
    - Viene eseguito il Web Service richiesto con i parametri che è possibile estrarre dal messaggio SOAP di richiesta
    - Viene generato e inviato al sender un messaggio SOAP di risposta contenente tra l'altro i parametri risultato dell'elaborazione
  - L'application server rende accessibile anche il catalogo dei WSDL dei servizi messi a disposizione
  - Esempio di application server: Apache Tomcat
    - Tomcat è in realtà un'estensione al web server Apache



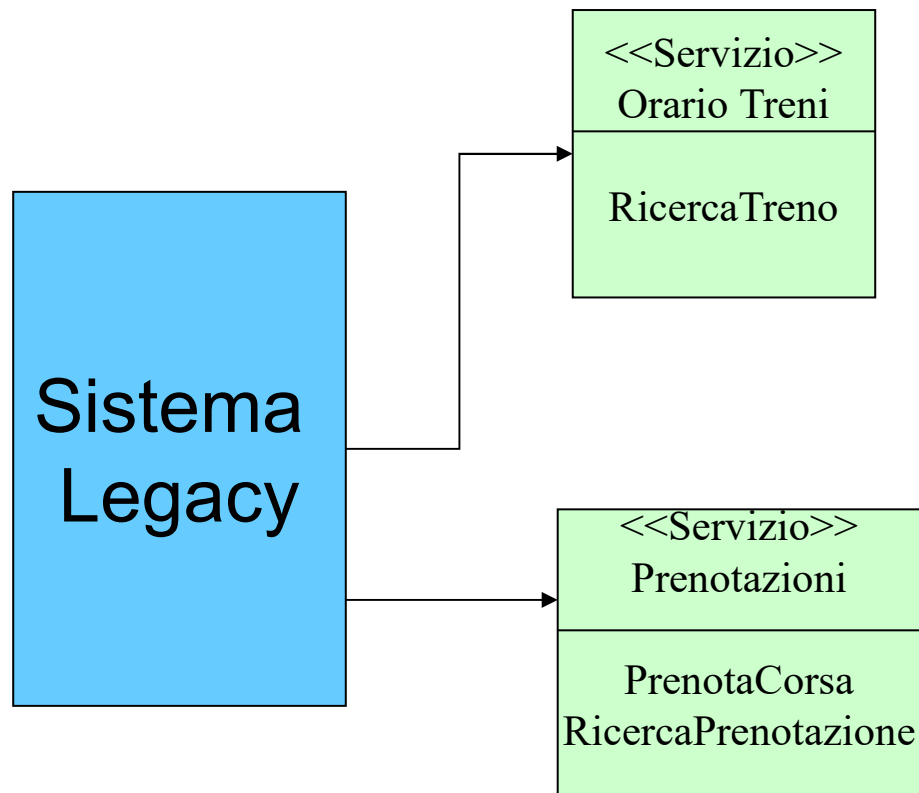
# UDDI

- Elemento fondamentale delle architetture SOA è la descrizione della **semantica** del servizio fornito che permetta, idealmente, la ricerca automatica di un servizio data la sua descrizione funzionale e, eventualmente, i requisiti di qualità richiesti o graditi
- Il linguaggio più spesso utilizzato per fornire tale descrizione semantica é UDDI (dialetto XML). Un documento UDDI contiene:
  - Dettagli sul fornitore del servizio
  - Descrizione (informale) delle funzionalità fornite
  - Posizione della descrizione WSDL dell'interfaccia
- C'è un ampio dibattito sulla definizione di standard:
  - che permettano una più precisa descrizione semantica dei servizi (utilizzando eventualmente anche ontologie)
  - Che permettano di descrivere più precisamente i requisiti di qualità (Quality of Service – QoS)

# Legacy system services

- Un importante vantaggio nell'uso dei servizi è che essi danno la possibilità di accedere alle funzionalità di legacy systems
- I Legacy systems offrono funzionalità mature e stabili, il cui riuso può ridurre i costi di sviluppo dei servizi
- Le applicazioni esterne potranno accedere a queste funzionalità attraverso l'interfaccia offerta dai servizi

# Accesso al sistema Legacy



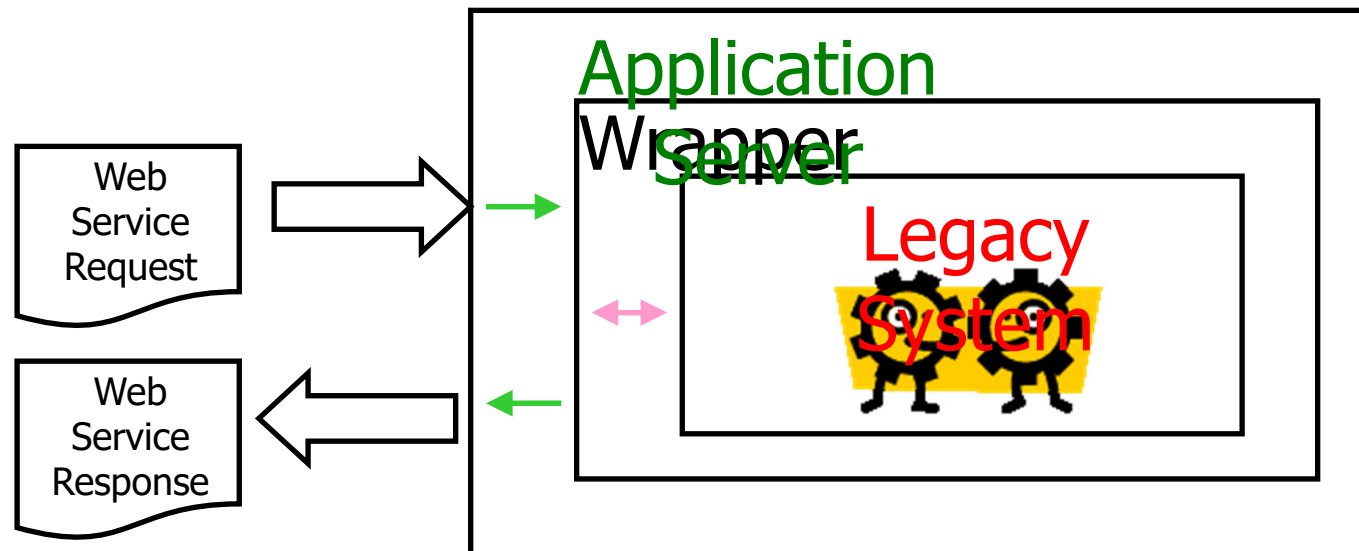
Le funzionalità del sistema legacy sono rese disponibili sotto forma di Servizi

# Migrazione di sistemi legacy verso SOA

- Le tecnologie SOA (e i Web Service in particolare) sono particolarmente utili per la migrazione di funzionalità critiche di sistemi legacy
  - Spesso, infatti, le funzionalità presenti nei sistemi legacy risolvono problemi particolarmente critici e sono da considerare molto affidabili, in quanto sono in opera da molti anni
- Migrando tali funzionalità sotto forma di Web Services, si aprono scenari per:
  - L'adattamento a interfacce moderne (ad esempio interfacce web)
  - La composizione all'interno di architetture moderne
- Una tecnica di migrazione basata su Wrapping per rendere accessibili le funzionalità di un sistema legacy sotto forma di servizi web è presentata in [1]

# Il ruolo del Wrapper nella migrazione di sistemi legacy con interfaccia utente basata su form

- Il wrapper ha il ruolo di interagire col sistema legacy (fornendo ad esso dati e comandi) al posto dell'utente.
- Grazie al wrapper il sistema legacy esporterà una interfaccia da Web Service

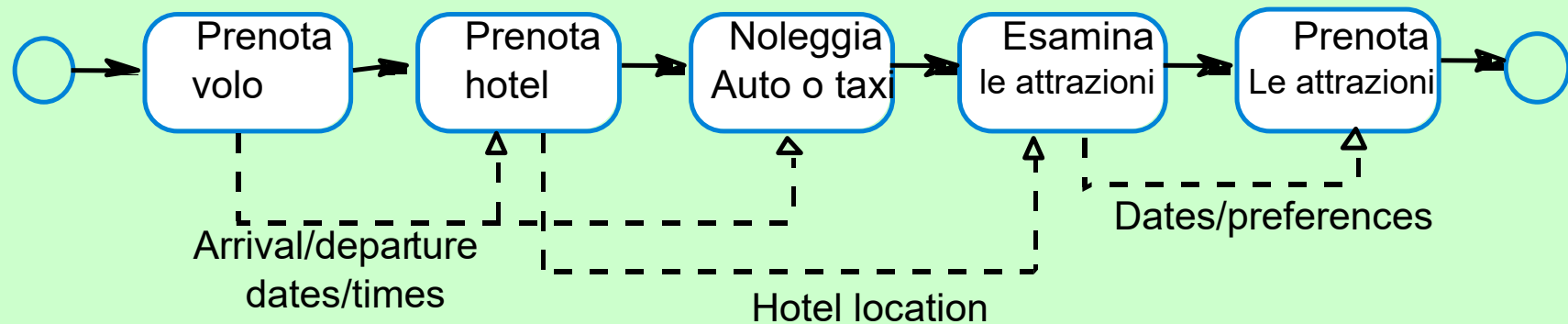


# Sviluppo basato su Servizi

# Sviluppo Software basato su servizi

- L'idea è di comporre e configurare servizi esistenti per creare nuovi servizi composti ed applicazioni.
- La base per la composizione del servizio è spesso un workflow
  - I Workflow sono sequenze logiche di attività che, insieme, modellano processi aziendali
  - Ad esempio, una composizione di servizi per prenotare un viaggio che coordina vari servizi di prenotazione volo, auto ed albergo.

# Esempio: workflow per la prenotazione di una vacanza

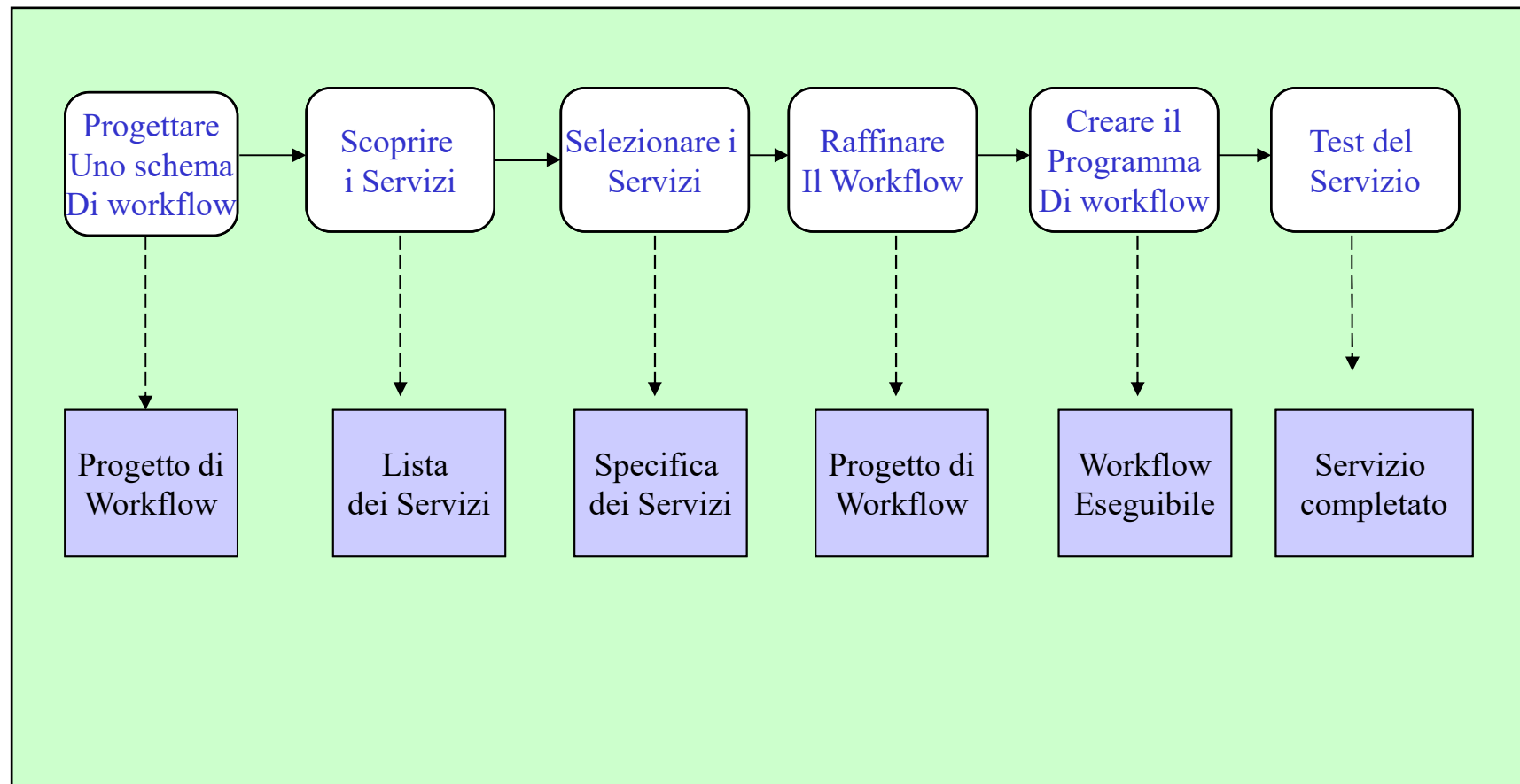




# Composizione di servizi

- Lo scenario *ideale* prospettato dalle SOA è uno scenario nel quale:
  - Si descrive un problema in un linguaggio formale (ma il più possibile intuitivo e vicino al linguaggio naturale)
  - Un motore di interpretazione scompone il problema in un **workflow** di sottoproblemi
  - Vengono automaticamente ricercati e reperiti Web Service in grado di risolvere ogni sottoproblema
  - Viene automaticamente composto il workflow in grado di risolvere il problema proposto

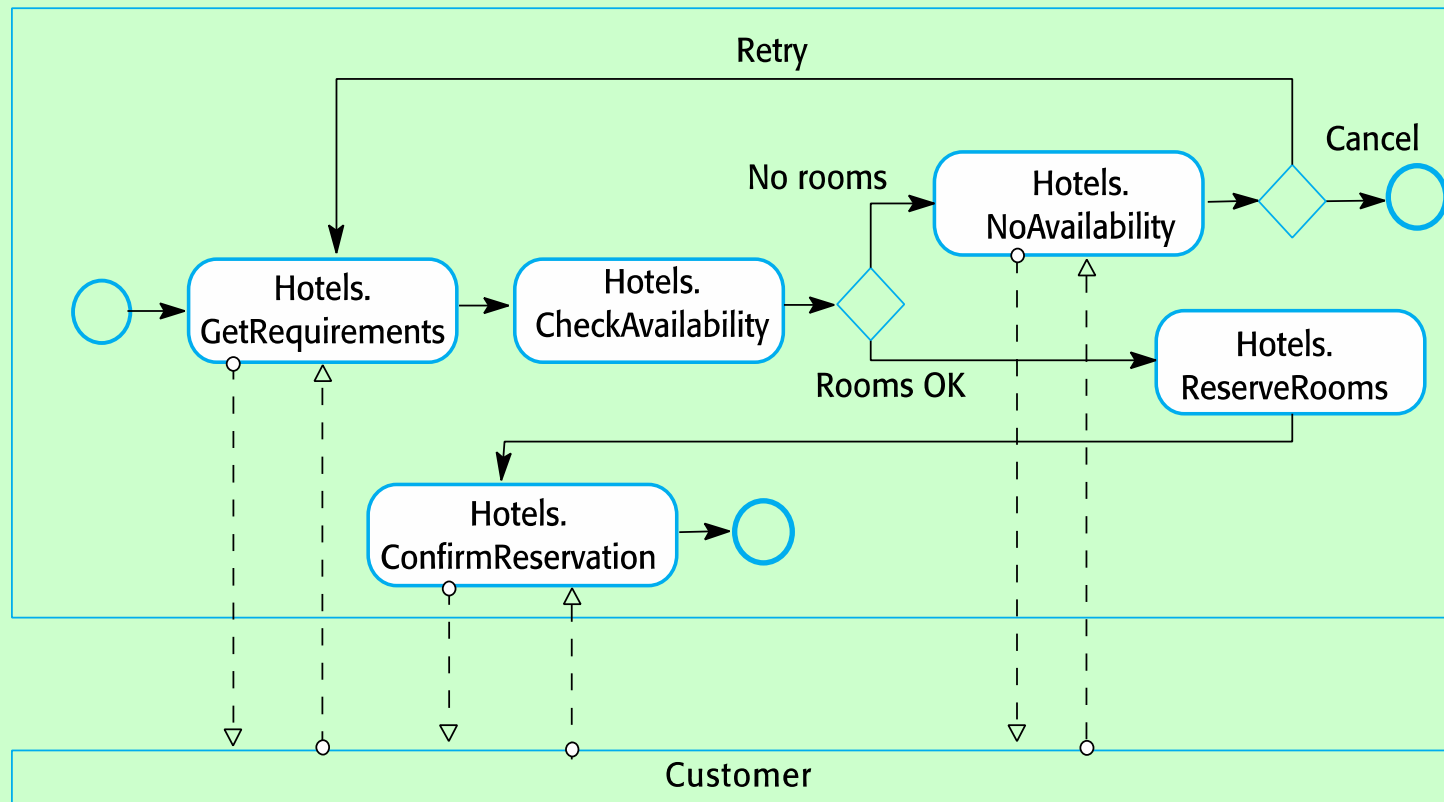
# Costruzione di un servizio mediante composizione



# Processo di composizione

- Le difficoltà legate alla realizzazione di un workflow di servizi provenienti da fonti eterogenee sono comunque notevoli
- In particolare bisogna tener conto di tutte le possibili **eccezioni** di ogni servizio e delle conseguenti azioni da intraprendere per recuperare dall'eccezione
- Ad esempio, se la prenotazione del viaggio va a buon fine ma fallisce quella dell'albergo, la semantica del servizio potrebbe prevedere la necessità di annullare la prenotazione del viaggio
  - Per fare ciò è necessario un ulteriore Web Service di annullamento prenotazione

# Il workflow per la prenotazione di una camera di hotel



# Progetto e implementazione dei workflow

- Sono stati proposti diversi linguaggi per la descrizione di workflow di servizi web
  - WS-BPEL è al momento il più comune. E' anch'esso un dialetto XML
  - Esistono componenti che vanno a estendere le funzionalità dell'application service, che agiscono da interpreti di documenti BPEL ed esecutori di Workflow
    - Esempio: BPEL4WS
  - Esistono inoltre strumenti come BPMN per la definizione visuale di workflow di web service

# Il Testing dei Servizi

- Il Testing ha lo scopo di trovare difetti e dimostrare che un sistema soddisfa i suoi requisiti funzionali e non-funzionali.
- Il testing dei servizi da parte del cliente del servizio è difficile in quanto i servizi (esterni) sono 'black-boxes'. Le tecniche di Testing basate sul codice sorgente non possono essere usate (se non nell'ambiente del produttore), mentre sono usabili tecniche black-box (es. Partition testing)

# Alcuni problemi nel Testing di Servizi

- Eventuali modifiche ad un Servizio potrebbero invalidare i test già effettuati
- Se si utilizza in binding dinamico (ovvero la ricerca “on-line” del servizio più adatto alla risoluzione di un problema), si dovrebbero rieseguire tutti i test progettati prima di utilizzare effettivamente il servizio
- Le caratteristiche di qualità del servizio (ad esempio velocità, tempo di risposta), dipendono dal carico e non possono essere previste
- Se i servizi sono a pagamento, il loro utilizzo a scopo di test è costoso
- Può essere difficile esercitare le azioni per la gestione delle eccezioni in servizi esterni, giacché esse possono dipendere da fallimenti di altri servizi che non possono essere simulati.



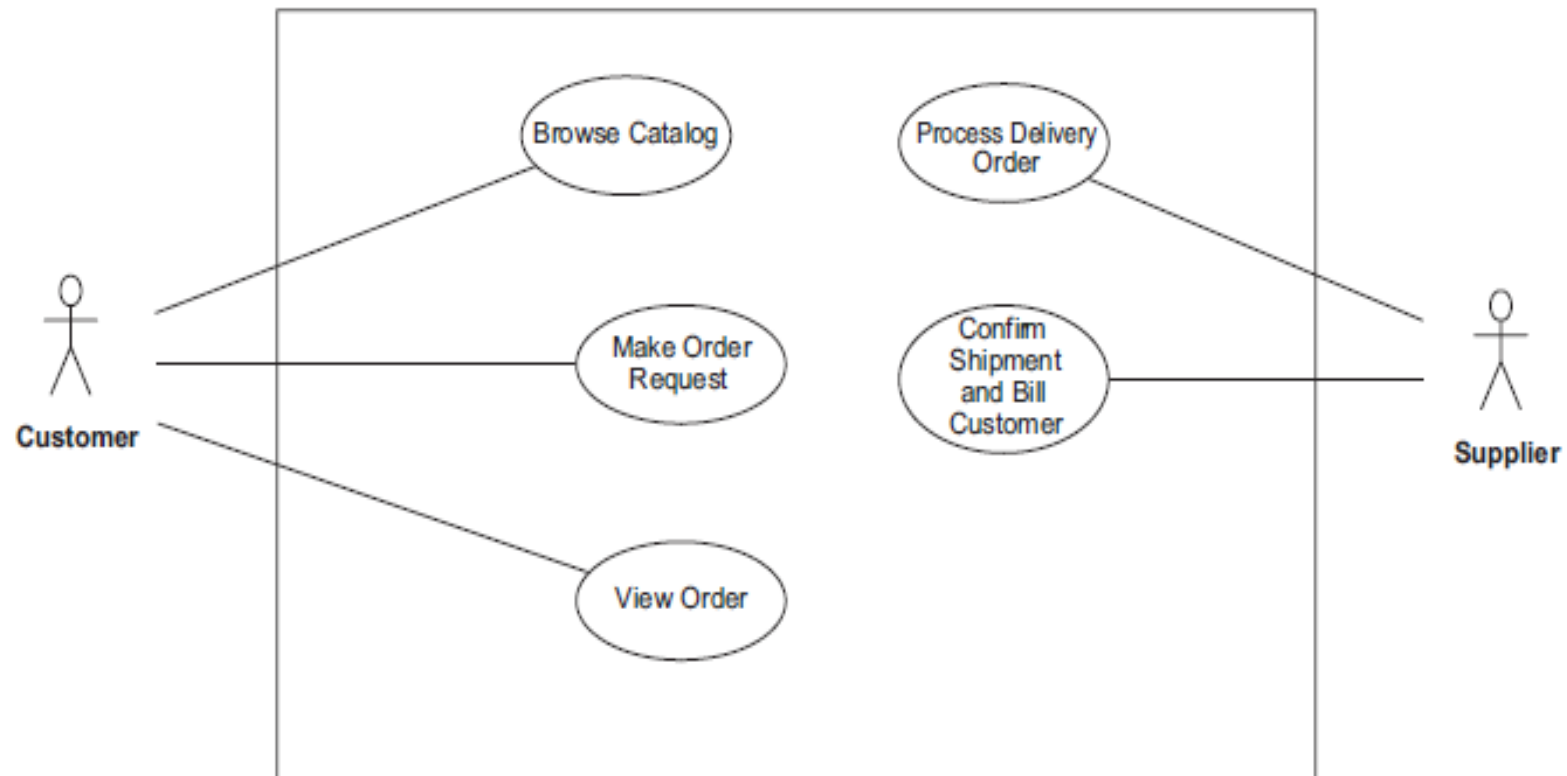


# Dal progetto OO ai Servizi

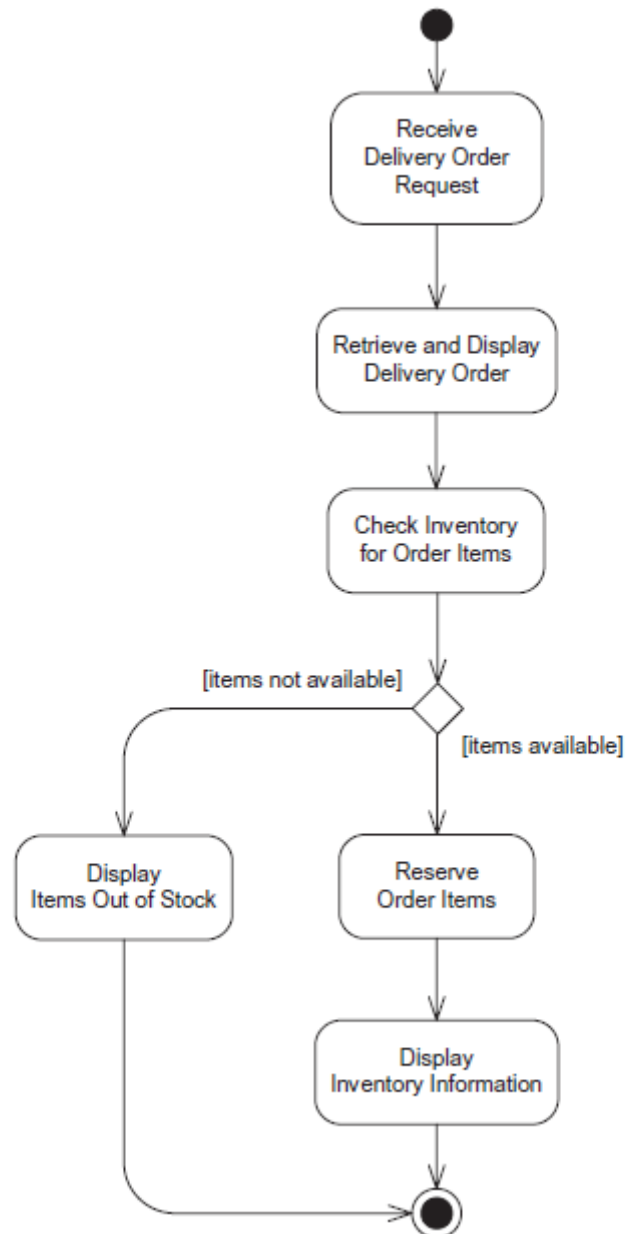
# Progetto delle Interfacce in SOA

- la progettazione delle interfacce dei servizi può partire dai diagrammi per la modellazione della dinamica dell'interazione fra oggetti (communication diagrams).
- Si analizzano i messaggi scambiati con gli oggetti di tipo servizio, per ottenere: nome operazione, input e output.
- Vediamo un esempio (per il Sistema di acquisiti online)

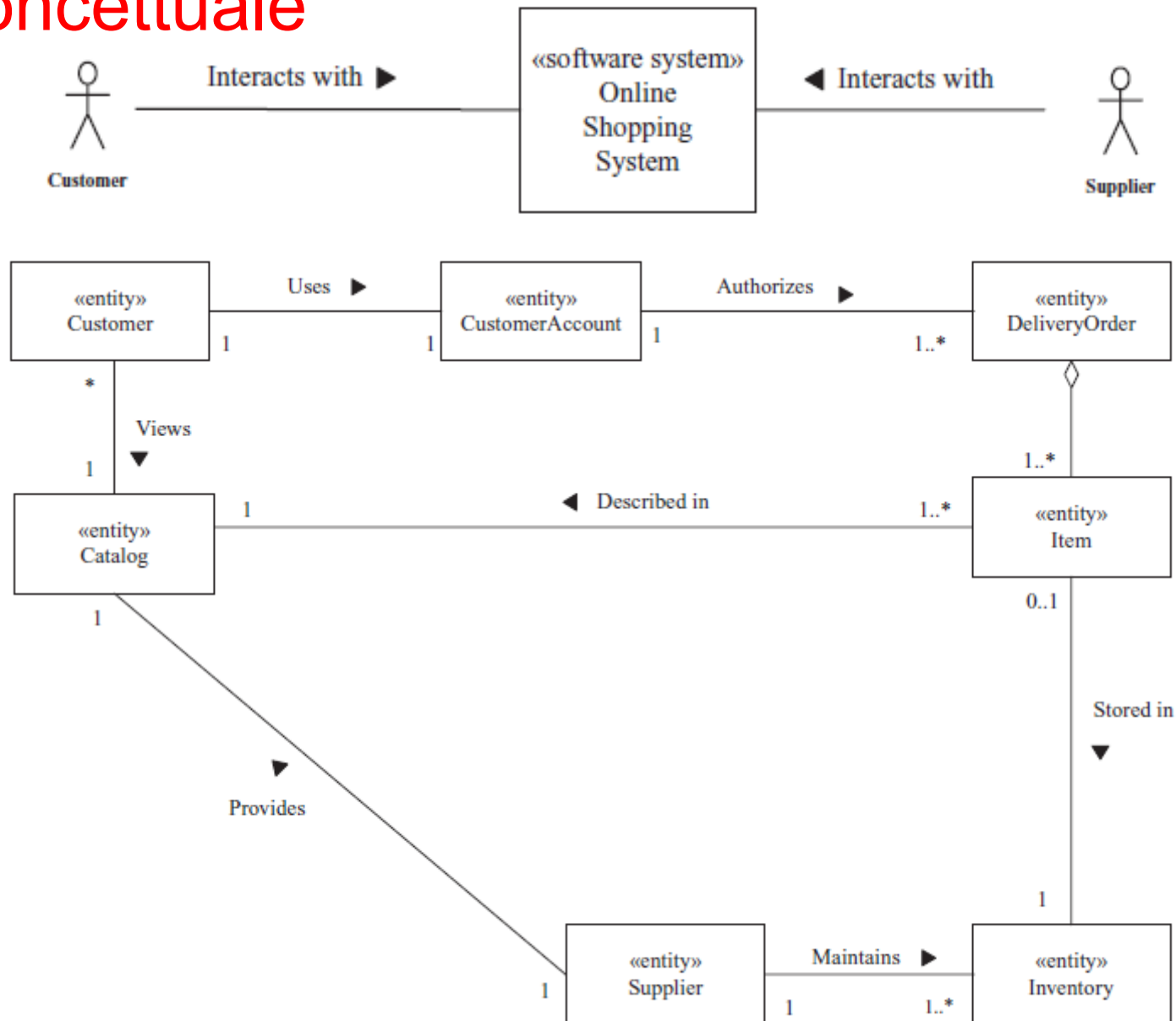
# Esempio: Il Sistema per Acquisti online



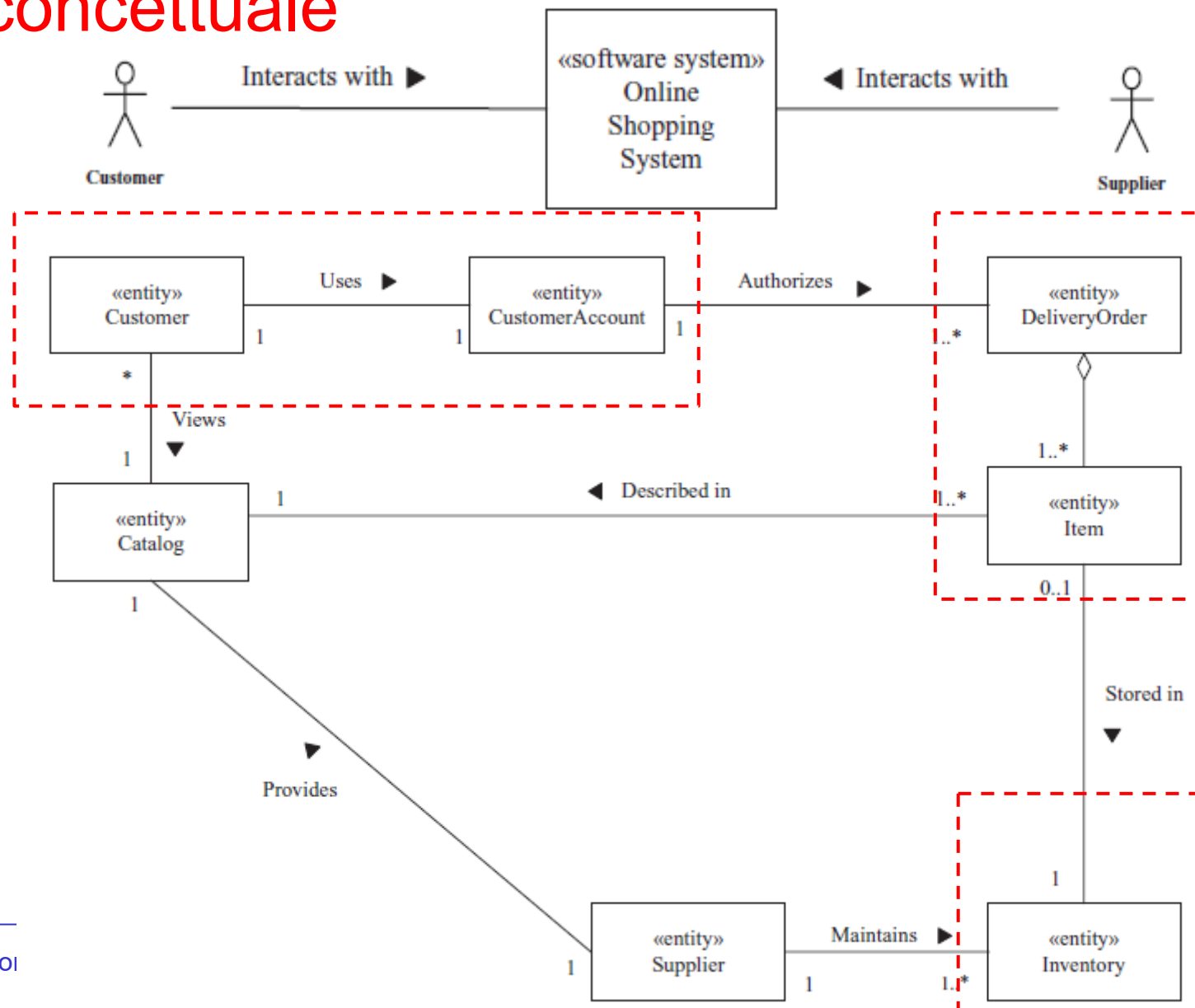
# L'activity diagram per lo UC Processa Ordine



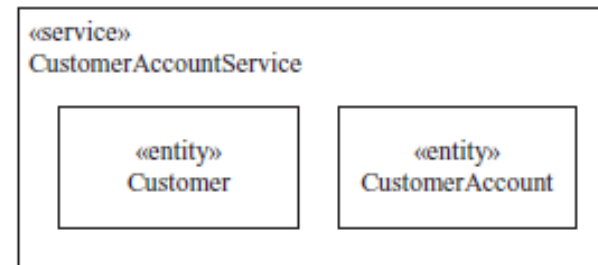
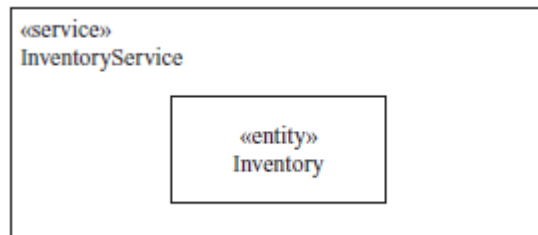
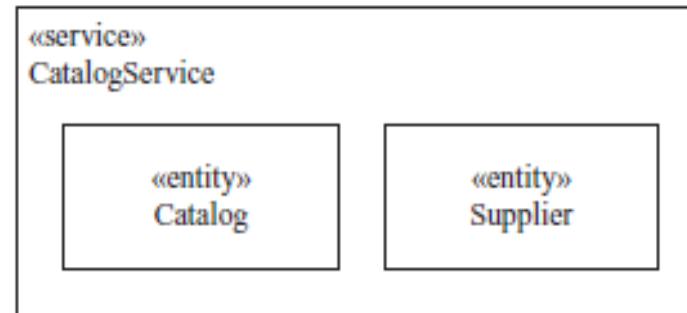
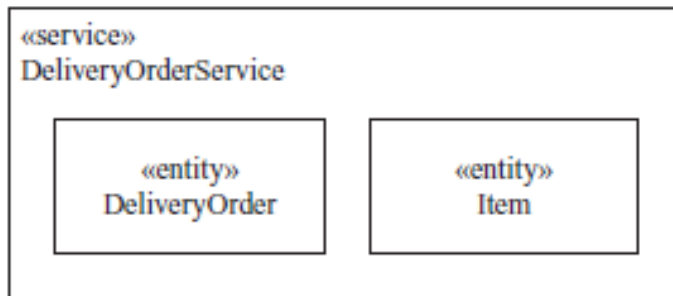
# Diagramma di Contesto e Modello concettuale



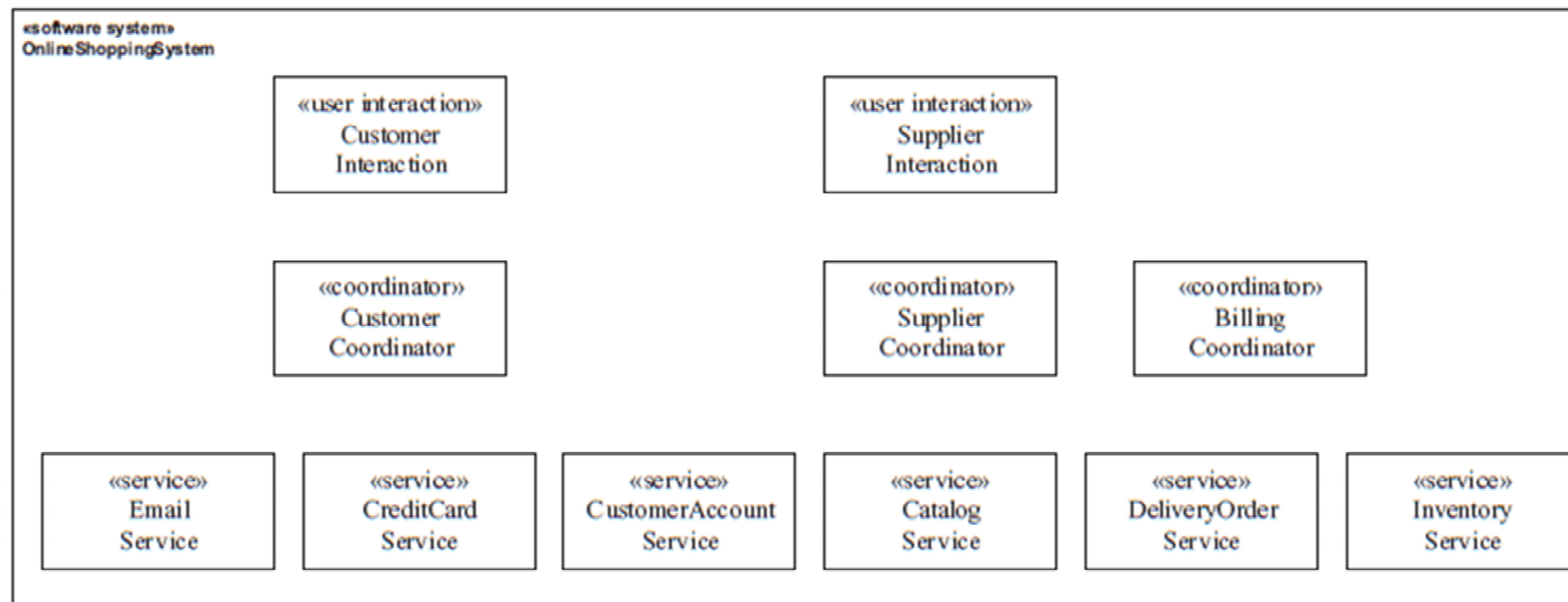
# Diagramma di Contesto e Modello concettuale



# Strutturazione degli oggetti- integrazione delle classi entità in una SOA mediante classi servizi

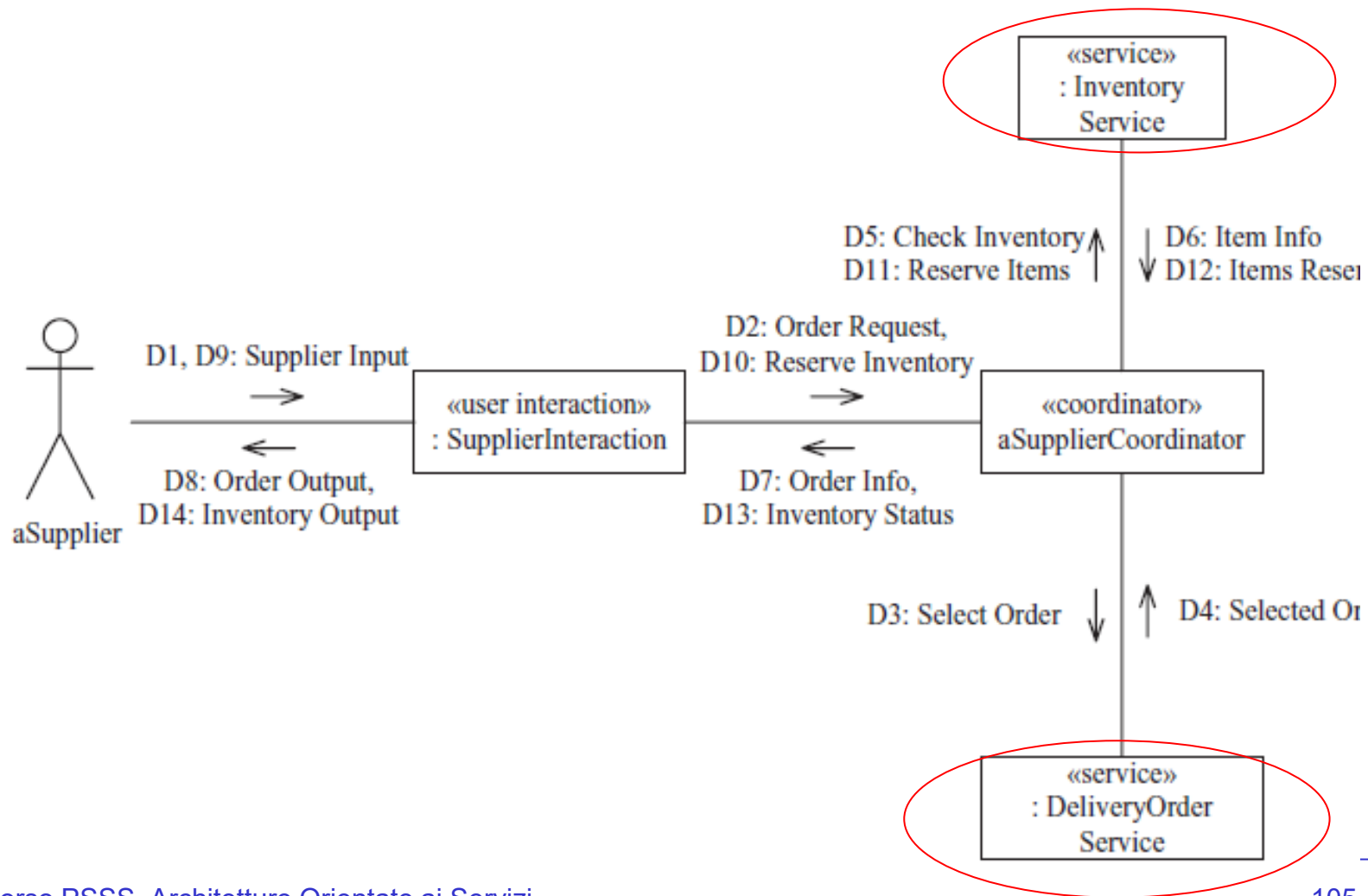


# Le classi strutturate del sistema





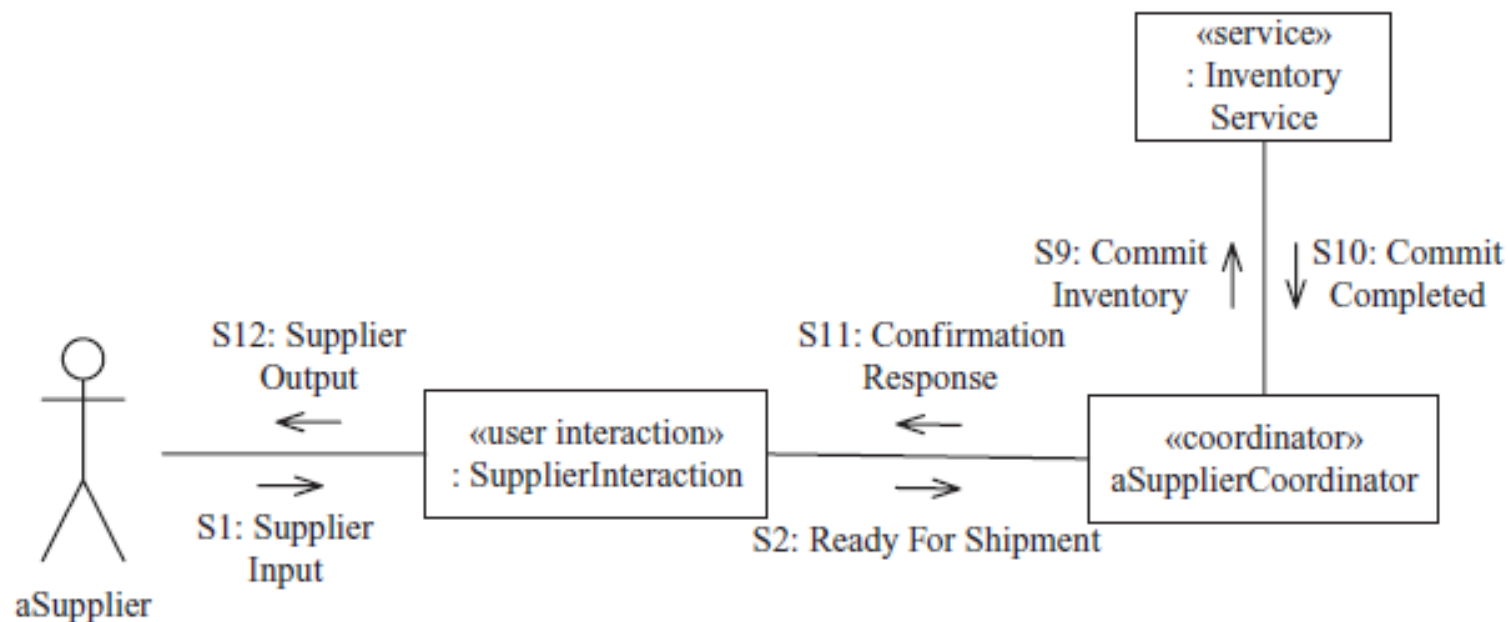
# Il communication diagram per il caso d'uso **processa ordine di consegna**



# Servizio Inventario

- Presenta le seguenti operazioni:
- **checkInventory** (in: itemId , out: inventoryStatus)  
(per controllare che gli articoli presenti nell'ordine siano disponibili in Inventario)
- **reserveInventory** (in: itemId, in: amount) (che corrisponde alla prima fase di un protocollo two-phase commit e serve a bloccare gli articoli)

# Il caso d'uso per la conferma della spedizione



*Il communication diagram per il caso d'uso Conferma Spedizione*

# Ulteriori operazioni del servizio Inventory

- **commitInventory** (in: itemId in:amount ) che corrisponde alla seconda fase del protocollo two-phase commit e completa la transazione
- **abort** (se l'ordine è cancellato)
- **update** (per aggiornare l'inventario).

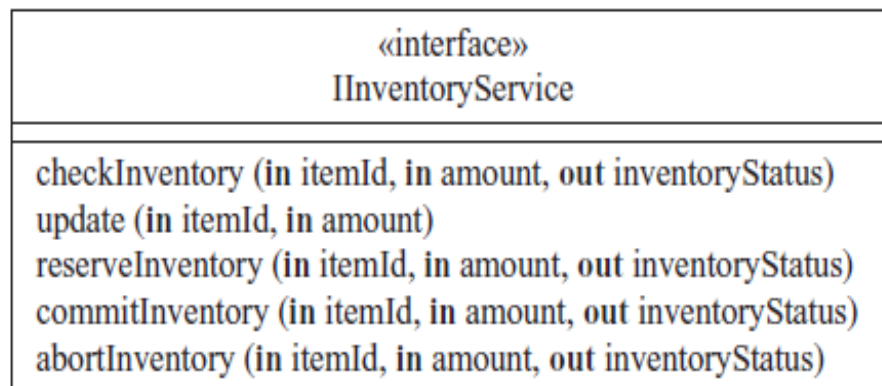
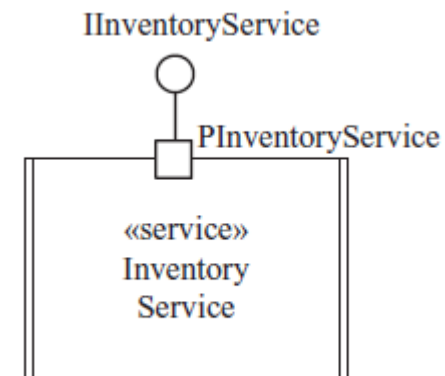


Figure 16.14. Service interface for Inventory Service

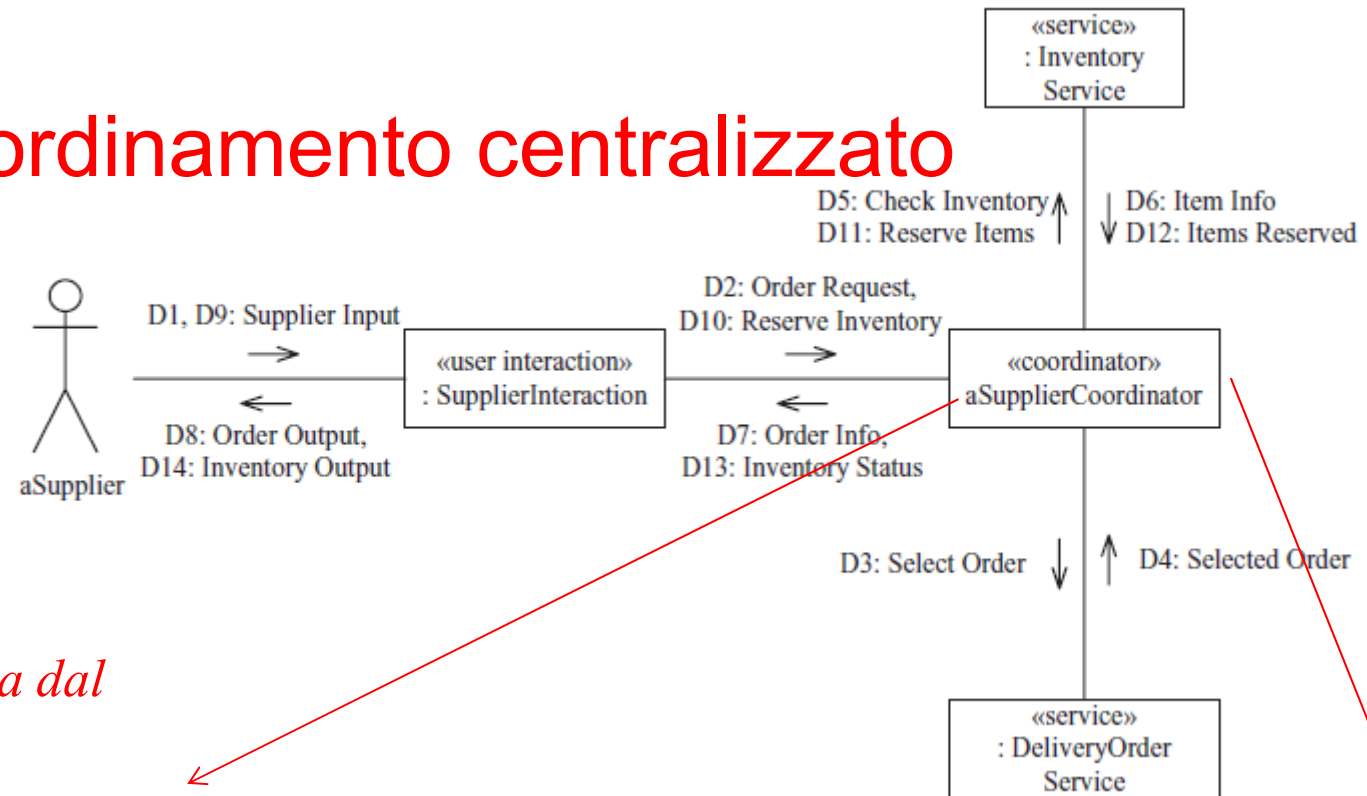


*Il servizio presenta un **port** detto **PInventoryService**, che supporta l'interfaccia offerta **IInventoryService**.*

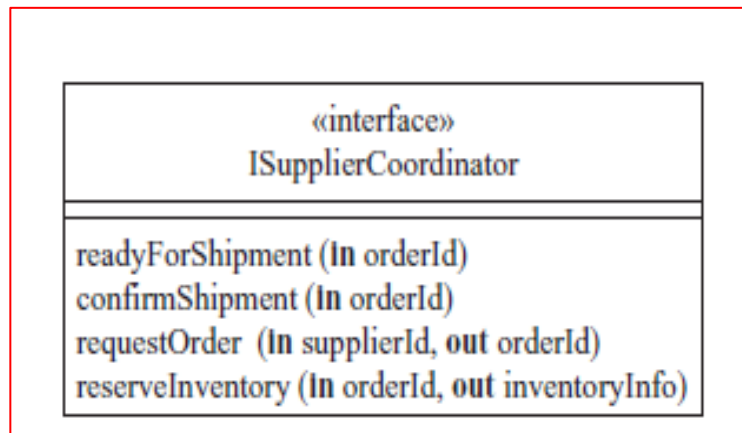
# Il coordinamento di Servizi nelle SOA

- In applicazioni SOA che coinvolgono più servizi, è importante la funzione di coordinamento, che può essere di due tipi:
- **Orchestrazione**: prevede una logica di coordinamento del workflow *controllata centralmente*.
- **Coreografia**: prevede un *controllo distribuito*, necessario quando si devono coordinare servizi provenienti da diverse organizzazioni.
- Gli oggetti di tipo «coordinatore» svolgeranno questo ruolo di coordinamento, ricevendo le richieste dei client e servendosi delle interfacce esposte dai servizi richiesti.

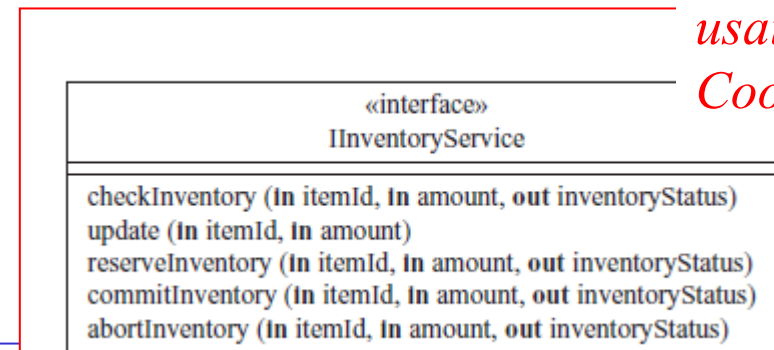
# Es. Coordinamento centralizzato



*Interfaccia offerta dal Coordinatore*



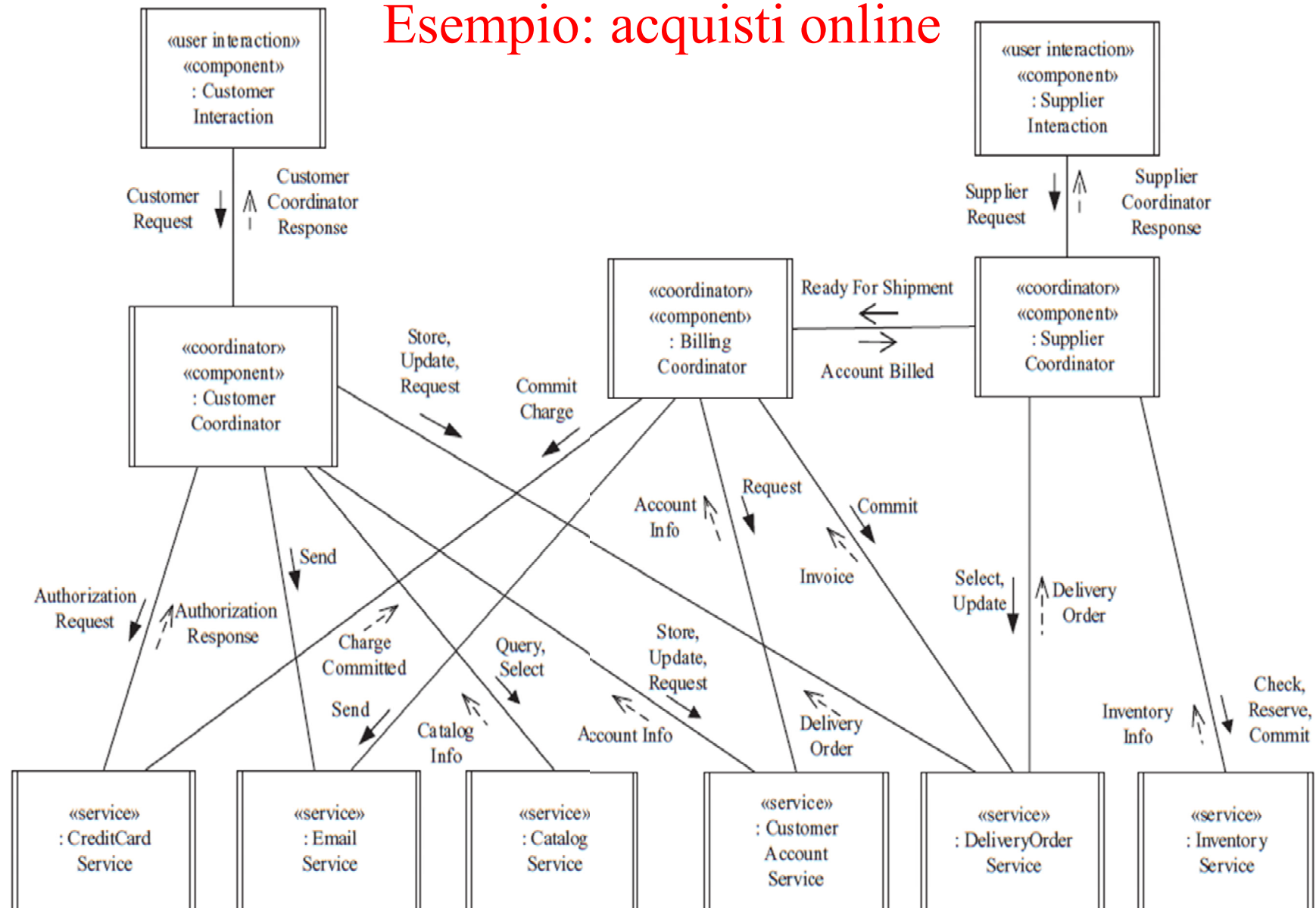
*Interfaccia usata dal Coordinatore*



# Progetto di applicazioni SOA

- A partire dal communication diagram concorrente integrato, si possono definire le interfacce di comunicazione fra componenti e servizi. Per la comunicazione, è possibile usare sia la comunicazione sincrona con Reply, che asincrona con Callback.
- Nella comunicazione peer-to-peer (ad es. Fra coordinatori) si può usare quella asincrona.

## Esempio: acquisti online





# Dettagli dell'architettura finale

- L'architettura finale prevede 2 componenti per l'interazione utente, 3 coordinatori (Customer Coordinator, Supplier Coordinator, e Billing Coordinator) e 6 servizi.
- Gli scambi di messaggio sono soprattutto di tipo request/response ossia basati sul pattern «Synchronous Message Communication with Reply» (tipico di SOA) (v. messaggi tra coordinatori e servizi).
- La comunicazione fra coordinatori è asincrona

# Riuso di Servizi

- La progettazione di servizi dovrebbe puntare a produrre servizi riusabili, quindi self-contained, privi di dipendenze da altri servizi.
- I servizi riusabili dovrebbero presentare solo l'interfaccia *Provides*, e non l'interfaccia *Requires*

