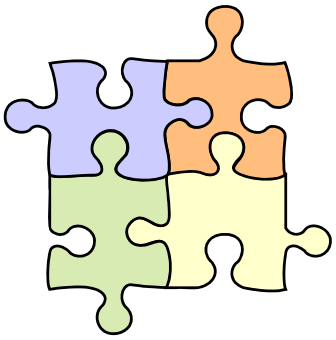
Luca Cabibbo



Architetture Software

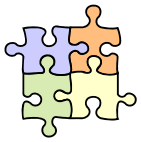
Web Services

Dispensa ASW 450

ottobre 2014

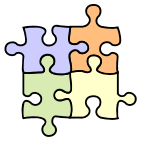
La cosa bella degli standard
è che ce ne sono così tanti
tra cui scegliere.

Andrew S. Tanenbaum



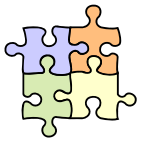
- Fonti

- [Papazoglou] Papazoglou, Web Services – Principles and Technology, 2008
- [ACKM] Alonso, Casati, Kuno, Machiraju, Web Services – Concepts, Architectures and Applications, 2004
- [POSA4] Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing, 2007
- [SAP] Chapter 6, Tactics for Interoperability
- [Fielding] Fielding, Architectural Styles and the Design of Network-based Software Architectures, PhD Thesis, 2000

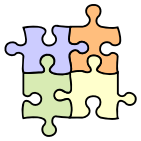


- Obiettivi e argomenti

- Obiettivi
  - introdurre i web services
  - introdurre i web services SOAP (con i relativi standard) e REST
- Argomenti
  - introduzione
  - web services
  - web services SOAP
  - web services REST
  - discussione

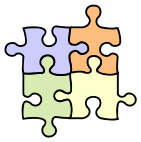


- Wordle



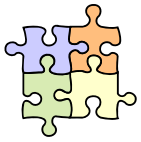
* Introduzione

- Nelle *architetture a servizi* ci sono due aspetti fondamentali
  - la tecnologia (middleware) a *servizi* è una tecnologia moderna per lo sviluppo e l'integrazione di applicazioni distribuite
    - in questo campo, i *Web Services* costituiscono la tecnologia a servizi dominante – con diverse implementazioni
  - l'*architettura orientata ai servizi* (*SOA*) fornisce il contesto metodologico (e di business) in cui utilizzare al meglio le tecnologie basate su servizi
- Questa dispensa si concentra soprattutto sulla tecnologia dei web services
  - l'architettura orientata ai servizi è discussa in un'altra dispensa
  - anche la programmazione dei web services è discussa in un'altra dispensa



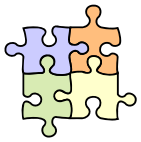
Interoperabilità

- Uno degli obiettivi fondamentali delle tecnologie a servizi è sostenere l'interoperabilità tra sistemi diversi – dunque, anche in presenza di varie forme di eterogeneità
  - l'*interoperabilità* ha a che fare con il grado in cui due o più sistemi possono interagire in modo effettivo, scambiando tra loro informazioni significative, tramite interfacce, in un contesto particolare
  - questa qualità riguarda
    - la capacità di scambiare dati – direttamente o indirettamente
    - la possibilità di interpretare correttamente i dati scambiati
  - alcuni possibili motivi per desiderare l'interoperabilità
    - rendere un servizio fruibile da parte di altri sistemi (anche di terzi, e dunque sconosciuti) – ad es., Google Maps
    - realizzare le funzionalità di un sistema in termini di capacità e servizi offerti anche da altri sistemi



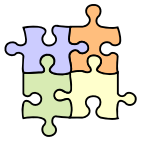
Tattiche per l'interoperabilità

- In generale, progettare per l'interoperabilità richiede di prendere in considerazione almeno i seguenti aspetti
  - interfacce dei servizi – le interazioni avverranno tramite le interfacce, e dunque queste devono essere descritte in modo opportuno (sintassi e semantica) e indipendente dalla tecnologie con cui sono realizzati i partecipanti alle interazioni
  - scoperta dei servizi – il consumatore di un servizio deve poter scoprire identità, locazione e interfaccia del servizio di interesse – questo può avvenire prima del runtime, ma talvolta anche a runtime
  - gestione delle richieste e delle risposte
    - deve essere effettivamente possibile invocare i servizi
    - inoltre, per favorire la composizione di servizi, devono essere possibili diversi tipi di scenari di interazione, ad es., richiesta-risposta o sulla base di flussi di messaggi



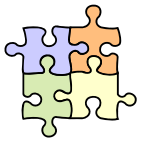
Intuizione: Dal Web ai Web Services

- Il *Web* consente l'accesso a dati/servizi tramite pagine web
  - nel web 1.0 (1990), solo i webmaster possono contribuire alle pagine web – gli altri utenti possono solo vederle
  - nel web 2.0 (2004), tutti gli utenti possono generare contenuti e contribuire alle pagine web
  - in ogni caso, l'accesso al web è pensato per client umani
  - difficile l'accesso diretto al web da parte di client software – ad es., mediante di tecniche di "screen scraping"
- I *Web Services* hanno l'obiettivo di fornire servizi a client software sulla base dei protocolli del web
  - sono un approccio standard (basato su standard) per far sì che componenti software siano resi disponibili e accessibili sul web
  - il riferimento a standard ha lo scopo di aumentare le possibilità di interoperabilità tra componenti/applicazioni



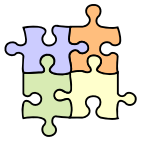
Web Services e SOA

- *Web Services*
  - costituiscono un modo di esporre le funzionalità di un componente o sistema software, per renderle disponibili tramite tecnologie Web standard
- *SOA* (*Service-Oriented Architecture*)
  - un servizio (in una SOA) è una funzionalità di business con un'interfaccia esposta, che può essere invocato dai suoi consumatori mediante messaggi
  - SOA è un'architettura concettuale di business in cui le funzionalità di business (logica applicativa) vengono esposte agli utenti SOA (consumatori) come servizi riusabili e condivisi in rete



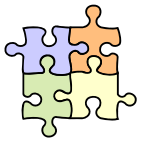
* Web Services

- Il paradigma di interazione basato su *servizi* (*services* o *e-services*) – con le relative tecnologie e middleware – sostiene lo sviluppo e l'integrazione di applicazioni distribuite
  - un obiettivo fondamentale (disatteso dalle tecnologie a componenti) è supportare l'interoperabilità tra componenti software in esecuzione su piattaforme diverse
  - ma anche con un'attenzione costante agli aspetti di business
  - generalità dei meccanismi di comunicazione – sia scambio di documenti/notifiche che chiamata di procedure remote
  - enfasi sull'interoperabilità tra componenti eterogenei, sulla base di protocolli standard aperti e universalmente accettati
  - flessibilità nell'organizzazione dei suoi elementi (servizi)
  - più in generale, sostegno ai requisiti posti dalle SOA
  - la tecnologia a servizi "dominante" è quella dei *Web Services* (WS)



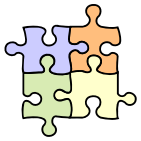
WS e interoperabilità

- I WS costituiscono l'ultimo passo (per ora) nello sviluppo di middleware per l'integrazione di applicazioni distribuite
  - le tecnologie a componenti (come .NET e Java EE) sono eccellenti per costruire o integrare applicazioni nell'ambito di una singola organizzazione
    - ma su che cosa basare lo sviluppo di applicazioni distribuite su larga scala – per collegare processi di business eseguiti da più organizzazioni indipendenti – indipendentemente dalle piattaforme tecnologiche utilizzate?
  - un obiettivo fondamentale della tecnologia dei WS è di offrire una soluzione a problemi pragmatici di interoperabilità, che nascono per differenze di piattaforma e di linguaggi di programmazione
    - i WS accettano la natura eterogenea delle organizzazioni attuali (e delle loro soluzioni informatiche), e comprendono che questa eterogeneità non diminuirà nel futuro



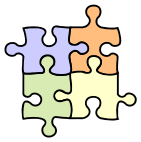
WS e standard

- I Web Services sono basati su un insieme di standard tecnologici per l'interoperabilità – indipendenti dalle piattaforme e neutrali rispetto ad essi
  - in precedenza erano state proposte e realizzate anche altre tecnologie per l'interoperabilità (in particolare, CORBA) – che però non sono state accettate/sostenute dai principali produttori di software
  - viceversa, il successo dei Web Services è legato anche all'importante fatto che la maggior parte dei produttori di software ha deciso di sostenere la tecnologia dei Web Services – per lo meno, per ciò che concerne gli standard fondamentali



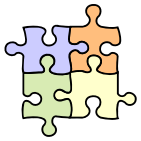
Web Services

- Dunque, i *Web Services* (*servizio web*) sono una tecnologia, basata su un insieme di standard tecnologici, per favorire l'accesso/ l'integrazione/ la composizione di servizi di business mediante tecnologie web
  - una forma di middleware
  - enfasi su apertura e obiettivi di interoperabilità
  - con caratteristiche tali da soddisfare molti requisiti derivanti dalle SOA
  - tecnologia e standard supportati dalla maggior parte dei produttori di software



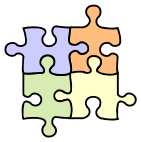
Che cosa sono i Web Services

- I servizi web hanno di solito una doppia caratterizzazione
  - un *web service* è un modulo o componente software, auto-contenuto e auto-descrittivo, accessibile mediante Internet, in modo indipendente dalla piattaforma
  - un *web service* ha lo scopo di svolgere un compito, risolvere un problema, o condurre transazioni per conto di un utente o applicazione – ovvero, di incapsulare un "servizio"
- La prima caratterizzazione è tecnica, la seconda di business
  - con i WS e le SOA, c'è sempre un'attenzione ad entrambi gli aspetti



Che cosa sono i Web Services

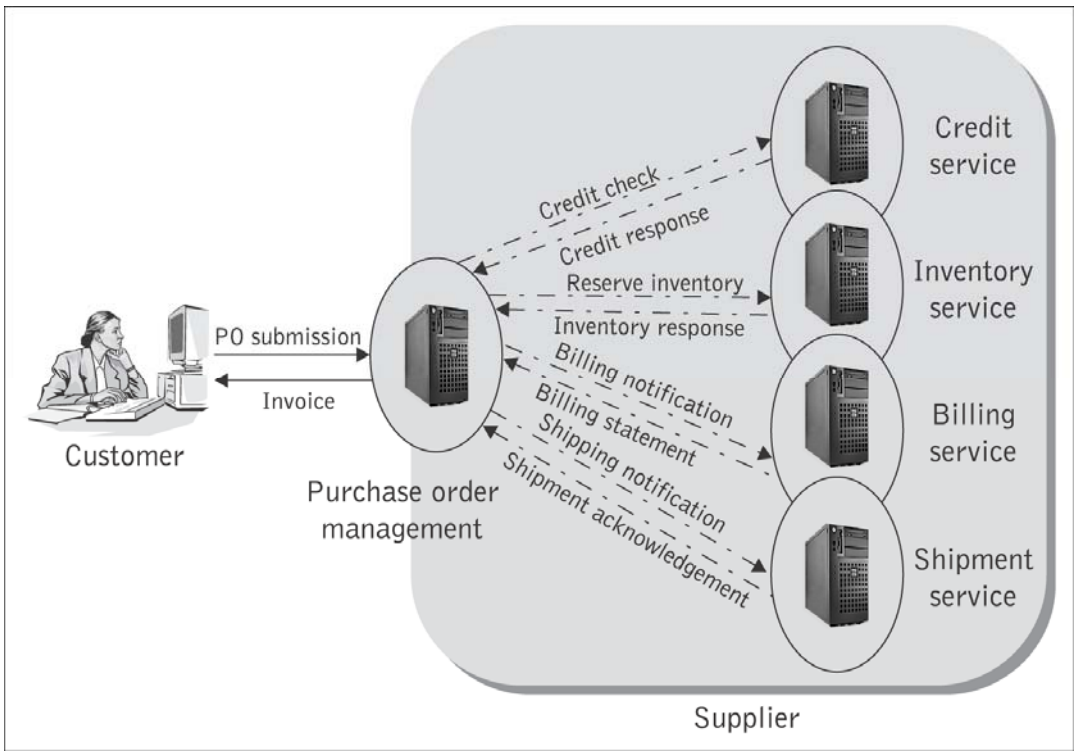
- Ecco alcuni esempi di “servizi” che possono essere rappresentati da un web service
  - un compito di business auto-contenuto – ad es., effettuare un bonifico bancario
  - l’accesso (come servizio) a una risorsa – ad es., a cartelle cliniche
  - un intero processo di business – ad es., l’acquisto automatizzato di prodotti e forniture per l’ufficio
  - un’applicazione – ad es., gestione di assicurazioni
    - include, ad es., un processo per la gestione di pratiche di rimborso
    - la gestione di una pratica di rimborso potrebbe richiedere di effettuare un bonifico – mediante un servizio erogato da una banca



Componibilità dei Web Services

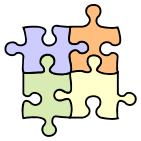
- Un requisito fondamentale è che i web services devono poter essere messi in corrispondenza e composti
  - una delle caratteristiche fondamentali della tecnologia dei web services è proprio la componibilità dei servizi
    - la composizione dei servizi può avvenire in modo più flessibile che non con le tecnologie a componenti
  - la tecnologia dei web services favorisce l’integrazione di servizi, per creare processi di business completi, con un costo di sviluppo ridotto
    - sia nell’ambito di una singola organizzazione (EAI) che tra diverse organizzazioni (integrazione di e-business)
  - i web services vanno progettati e definiti come elementi costitutivi per la creazione di applicazioni distribuite
    - un buon servizio web deve poter essere impiegato nella realizzazione di più applicazioni

Un esempio – gestione di ordini d'acquisto



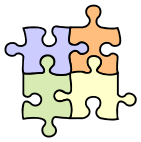
Capacità dei Web Services

- Le capacità fondamentali offerte della tecnologia dei web services sono motivate dagli obiettivi di interoperabilità e di integrazione che questa tecnologia si propone di perseguire
  - descrivere servizi – in modo dettagliato e indipendente dalla piattaforma
  - scoprire servizi – mediante opportuni strumenti di ricerca, per trovare i servizi adatti a risolvere un certo problema
  - invocare servizi – anche in modo dinamico
  - comporre servizi – per definire dei nuovi servizi
  - queste funzionalità sono fornite agli sviluppatori sulla base di opportuni (e numerosi) standard
  - ci sono due famiglie principali di standard per web services, che condividono molti obiettivi, ma presentano anche numerose variazioni – i web services SOAP e quelli REST

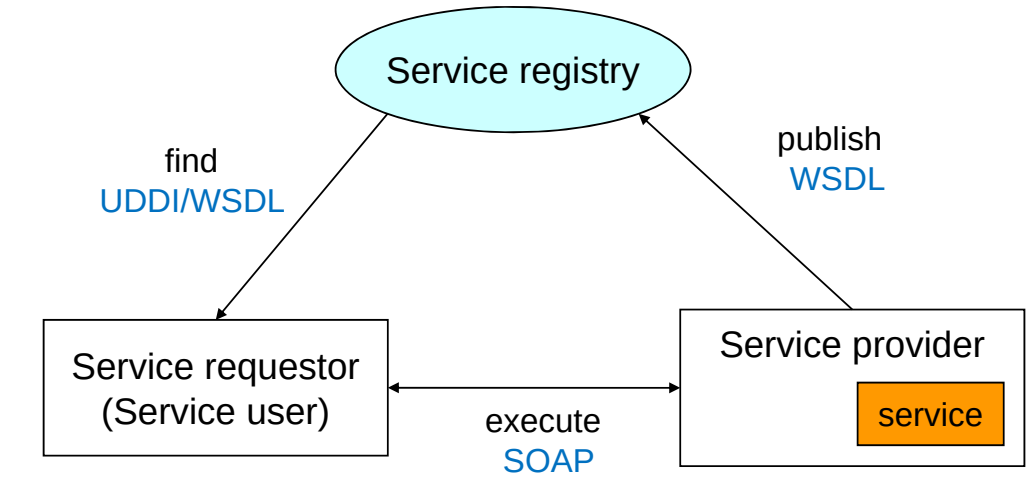


* Web Services SOAP

- La tecnologia principale nel contesto dei web services è costituita dai *web services SOAP* – detti anche *web services WS-** – con riferimento agli standard utilizzati
  - i web services SOAP sono basati su numerosi standard – che riguardano sia funzionalità fondamentali per l'utilizzo dei WS – ad es., definizione dell'interfaccia, invocazione, registry per servizi, composizione di servizi ... – che altre qualità
    - gli standard principali sono WSDL, UDDI, SOAP – tutti basati su XML
    - inoltre ci sono BPEL e le estensioni WS-*
  - i principali responsabili degli standard per i Web Services SOAP sono il consorzio OASIS (*Organization for the Advancement of Structured Information Standards*) e il W3C (*World Wide Web Consortium*)



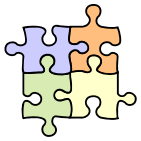
Interazione tra servizi



elemento che invoca servizi forniti da altri

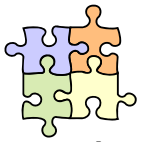
elemento che offre servizi che possono essere usati da altri

i due ruoli non sono mutuamente esclusivi:
un elemento può sia offrire che invocare servizi



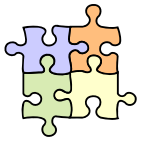
Standard per web services

- I web services SOAP (per semplicità chiamati solo WS in questa sezione) sono basati su un insieme di standard
  - *XML*
    - l'adozione di XML come "sintassi" e "linguaggio di trasporto" sostiene l'indipendenza dei vari standard per WS dalla piattaforma e dai linguaggi di programmazione
  - *SOAP*
    - definisce l'organizzazione per lo scambio di dati e messaggi strutturati
  - *WSDL – Web Services Description Language*
    - un linguaggio per la definizione dell'interfaccia di WS
  - *UDDI – Universal Description, Discovery and Integration*
    - standard per la ricerca di WS



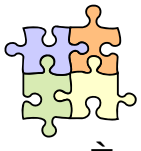
Standard per web services

- È possibile comprendere le specifiche dei WS ragionando sui problemi che i WS intendono affrontare
  - ci concentriamo soprattutto sulla descrizione degli standard di base, che definiscono un'infrastruttura minimale per i WS
  - sopra a questa infrastruttura sono state definite numerose estensioni – a cui ci si riferisce come WS-*



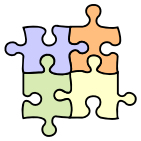
- Sintassi

- Gli standard per i WS sono definiti usando una sintassi comune
  - per sostenerne portabilità ed estendibilità
- La sintassi scelta è *XML* – *eXtensible Markup Language*
  - il vantaggio principale è la sua generalità e flessibilità
  - lo svantaggio principale è l'overhead introdotto da XML nella codifica, trasmissione, decodifica – legato al fatto che le informazioni sono comunemente codificate in XML in modo "prolisso"



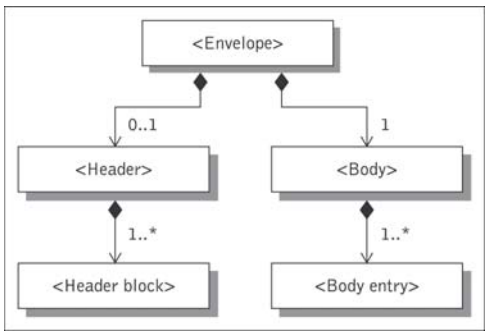
- Interazione con web services e SOAP

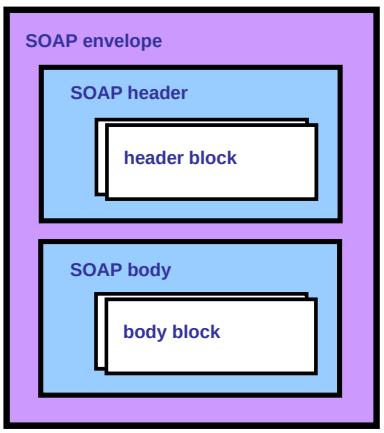
- È necessario un meccanismo per consentire l'interazione tra un web service e i suoi "client" – ci sono tre aspetti da considerare
  - formato comune per i messaggi scambiati
  - supporto per diverse forme specifiche di interazione – ad es., nello stile procedurale (RPC) e nello stile del messaging
  - compatibilità con diverse modalità di trasporto dei messaggi – ad es., basato su HTTP, TCP, SMTP, JMS, ... – ma, allo stesso tempo, indipendenza dalla modalità di trasporto
- Le interazioni con/tra WS sono basate su *SOAP*
  - la comunicazione è basata sullo scambio di *messaggi*
  - SOAP è, di per sé, un protocollo stateless e one-way
    - è però possibile anche una comunicazione nello stile richiesta-risposta, definita sulla base di coppie di messaggi
  - inizialmente un acronimo, oggi SOAP ha significato autonomo

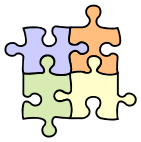


Messaggi SOAP

- Un *messaggio SOAP* è costituito da una busta (*envelope*), che contiene un'intestazione e un corpo
  - il corpo (*body*) racchiude le informazioni che il messaggio vuole effettivamente trasmettere – ad es., un documento, oppure il nome di un metodo invocato e i relativi parametri
  - l'intestazione (*header*) contiene metadati e altre informazioni "non funzionali" – ad es., credenziali di autenticazione oppure l'id della transazione



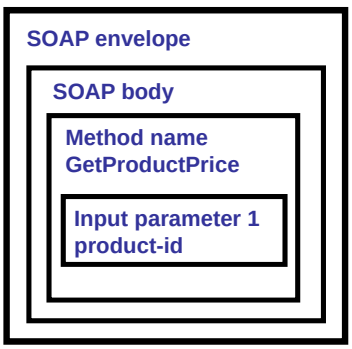




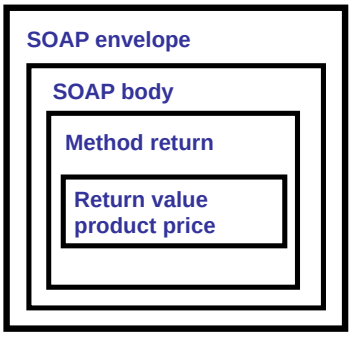
Messaggi SOAP

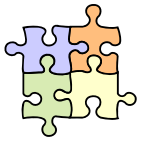
- Ad es., coppia di messaggi SOAP in un'interazione in stile RPC

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
            <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
                env:mustUnderstand='1'>
               512
            </tx:Transaction-id>
    </env:Header>
    <env:Body>
            <m:GetProductPrice>
               <product-id>  450R6OP  </product-id >
            </m:GetProductPrice >
    </env:Body>
</env:Envelope>
```



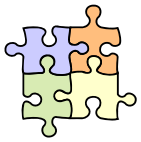

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
            <--! – Optional context information -->
    </env:Header>
    <env:Body>
            <m:GetProductPriceResponse>
               <product-price>   134.32  </product-price>
            </m:GetProductPriceResponse>
    </env:Body>
</env:Envelope>
```



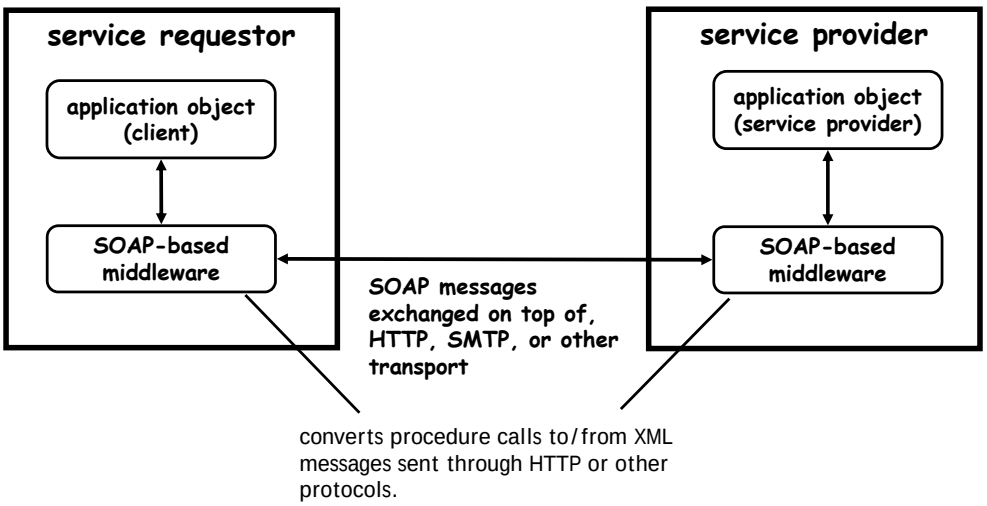


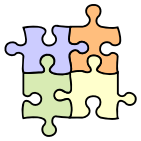
Modelli di comunicazione

- Il modello di comunicazione di SOAP prevede diverse varianti, basate su due proprietà
  - *stile di comunicazione* – *RPC* oppure *document*
    - lo stile *RPC* consente di definire messaggi in corrispondenza alla richiesta di esecuzione di un'operazione o alla relativa risposta
    - lo stile *document* (*message*) consente di scambiare documenti (messaggi) che contengono dati XML
  - *stile di codifica (encoding)* – *encoded* oppure *literal*
    - lo stile *encoded* (solitamente usato con lo stile RPC) è basato su una codifica standard dei dati trasmessi, e consente ad esempio la serializzazione di grafi di oggetti
    - lo stile *literal* (solitamente usato con lo stile document) non prevede invece nessuna codifica dei dati trasmessi



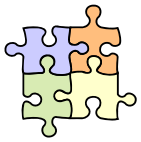
Interazione con SOAP





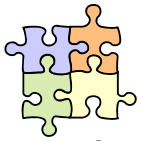
Osservazioni su SOAP

- SOAP è un “wire protocol” – ovvero, si occupa della forma con cui sistemi o applicazioni distribuite possono scambiarsi dei dati
  - tuttavia, SOAP non è un “transport protocol” – ovvero, non si occupa di come i dati possono essere concretamente scambiati tra sistemi o applicazioni
  - lo scambio di messaggi SOAP avviene sulla base di altri protocolli – ad es., HTTP – l’aspetto del binding di messaggi SOAP è considerato e descritto nel contesto di WSDL



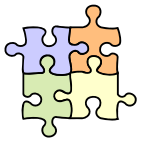
Osservazioni su SOAP

- SOAP definisce un semplice meccanismo per codificare dati e consentirne lo scambio tra applicazioni distribuite
  - tuttavia, non definisce nessuna semantica
    - questo lo rende adatto a essere usato in una varietà di sistemi – da RPC al messaging
  - inoltre, SOAP non si occupa di aspetti come il routing o lo scambio affidabile di messaggi
    - ma è adatto a contesti in cui è necessario scambiare informazioni in modo flessibile ed estensibile



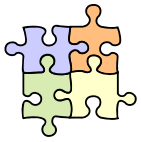
- Descrizione di servizi e WSDL

- È necessario anche un meccanismo per la descrizione di servizi
  - per descrivere l'interfaccia di un servizio – come avviene con l'IDL di altri middleware
  - nonché per descrivere indirizzamento (locazione del servizio) e modalità di trasporto
- La descrizione di un Web Service avviene mediante *WSDL* – *Web Service Description Language*
  - una specifica WSDL di un WS descrive
    - *che cosa* fa il WS – ovvero, quali operazioni fornisce
    - *come* è possibile invocare il WS – ovvero, il dettaglio dei formati dei dati e dei protocolli per accedere alle operazioni del servizio
    - *dove* risiede il WS – ad es., mediante URI (URL o URN), e la modalità di trasporto (ad es., HTTP)



Contenuto di un documento WSDL

- Un documento WSDL contiene
  - una *parte astratta* – descrizione dell'interfaccia
    - *data types* – dichiarazione dei tipi di dato per i dati scambiati, ad es., per i parametri e i risultati delle operazioni
    - *message* – il formato di un possibile messaggio scambiato – struttura del corpo di un possibile messaggio SOAP
    - *operation* – descrizione astratta di una singola operazione del servizio
    - *port type* – un tipo astratto di servizio e l'insieme delle sue operazioni
  - una *parte concreta* – descrizione dell'implementazione
    - lega la definizione astratta del servizio con degli indirizzi di rete concreti, un protocollo specifico e delle strutture di dati specifiche
    - in termini di uno o più binding



```
<wsdl:definitions name="PurchaseOrderService"
    targetNamespace="http://supply.com/PurchaseService/wsdl"
    xmlns:tns="http://supply.com/ PurchaseService/wsdl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://supply.com/PurchaseService/wsdl"
      <xsd:complexType name="CustomerInfoType">
          <xsd:sequence>
              <xsd:element name="CusNamer" type="xsd:string"/>
              <xsd:element name="CusAddress" type="xsd:string"/>
           </xsd:sequence>
      </xsd:complexType>
       <xsd:complexType name="POType">
            <xsd:sequence>
                <xsd:element name="PONumber" type="integer"/>
                <xsd:element name="PODate" type="string"/>
            </xsd:sequence>
       </xsd:complexType>
       <xsd:complexType name="InvoiceType">
            <xsd:all>
                <xsd:element name="InvPrice" type="float"/>
                <xsd:element name="InvDate" type="string"/>
            </xsd:all>
       </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="POMessage">
      <wsdl:part name="PurchaseOrder" type="tns:POType"/>
      < wsdl:part name="CustomerInfo" type="tns:CustomerInfoType"/>
  </wsdl:message>
  <wsdl:message name="InvMessage">
      <wsdl:part name="Invoice" type="tns:InvoiceType"/>
  </wsdl:message>
  <wsdl:portType name="PurchaseOrderPortType">
      <wsdl:operation name="SendPurchase">
          <wsdl:input message="tns:POMessage"/>
          <wsdl:output message="tns:InvMessage"/>
       </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

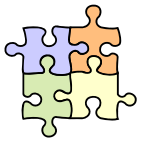
Abstract data type definitions

Data that is sent

Data that is returned

Port type with one operation

An operation with request (input) & response (output) message



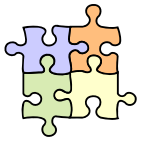
```
<wsdl:definitions> . ...
    <import namespace="http://supply.com/PurchaseService/wsdl"
     location="http://supply.com PurchaseService/wsdl/PurchaseOrder-interface.wsdl"/>
  <!--  location of WSDL PO interface from Listing-1-->
  <!-- wsdl:binding states a serialisation protocol for this service -->
  <!--  type attribute must match name of portType element in Listing-1-->
  <wsdl:binding name="PurchaseOrderSOAPBinding"
                    type="tns:PurchaseOrderPortType">

    <!-- leverage off soapbind:binding synchronous style -->
    <soapbind:binding style="rpc"
                          transport="http://schemas.xmlsoap.org/soap/http/"/>

    <wsdl:operation name="SendPurchase">

      <!-- again bind to SOAP -->
      <soapbind:operation
        soapAction="http://supply.com/PurchaseService/wsdl SendPurchase" style="rpc"/>

      <!-- furthur specify that the messages in the wsdl:operation use SOAP -->
       <wsdl:input>
          <soapbind:body  use="literal"
            namespace="http://supply.com/PurchaseService/wsdl"/>
       </wsdl:input>
       <wsdl:output>
           <soapbind:body  use="literal"
             namespace="http://supply.com/PurchaseService/wsdl"/>
        </wsdl:output>

    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="PurchaseOrderService">
      <wsdl:port  name="PurchaseOrderPort" binding="tns:PurchaseOrderSOAPBinding">
          <!-- give the binding a network endpoint address or URI of service  -->
          <soapbind:address location="http://supply.com:8080/PurchaseOrderService"/>
      </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Bind an abstract operation to this implementation and

map the abstract input and output messages to these concrete messages

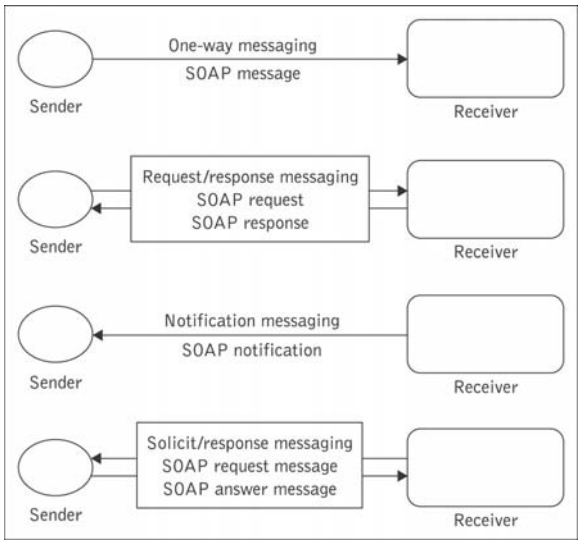
Service name

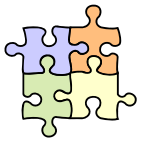
Network address of service



Message Exchange Pattern

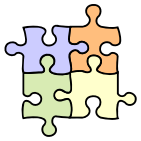
- WSDL consente di definire operazioni basate su quattro modalità di interazione – in relazione ai possibili pattern per lo scambio di messaggi tra servizi (*Message Exchange Pattern* o *MEP*)
  - ogni operazione può avere un messaggio di input e/o un messaggio di output – pertanto i MEP possibili sono quattro





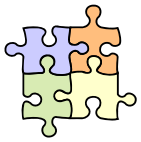
Message Exchange Pattern

- *Request/response*
  - un'operazione in cui il servizio riceve un messaggio, e poi invia una risposta al suo client – è una forma di chiamata di procedura remota
- *One way*
  - un'operazione in cui il servizio riceve un messaggio, ma non invia risposta al suo client – è dunque una forma di messaging asincrono – ad esempio, la sottomissione di un ordine



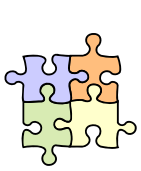
Message Exchange Pattern

- *Notification*
  - un'operazione in cui il servizio invia un messaggio a un client, senza attendere risposta
  - è un meccanismo di notifica di eventi ai client – i client interessati (subscriber) si devono registrare al servizio per ricevere notifiche relative ad eventi di un certo tipo
- *Solicit/response*
  - un'operazione in cui il servizio invia un messaggio a un client, e poi ne riceve una risposta – è dunque complementare a request/response
  - è simile alla notification (richiede la registrazione), ma prevede la risposta dei client – ad esempio, il servizio invia informazioni sullo stato dell'ordine di un cliente (e vuole una ricevuta)



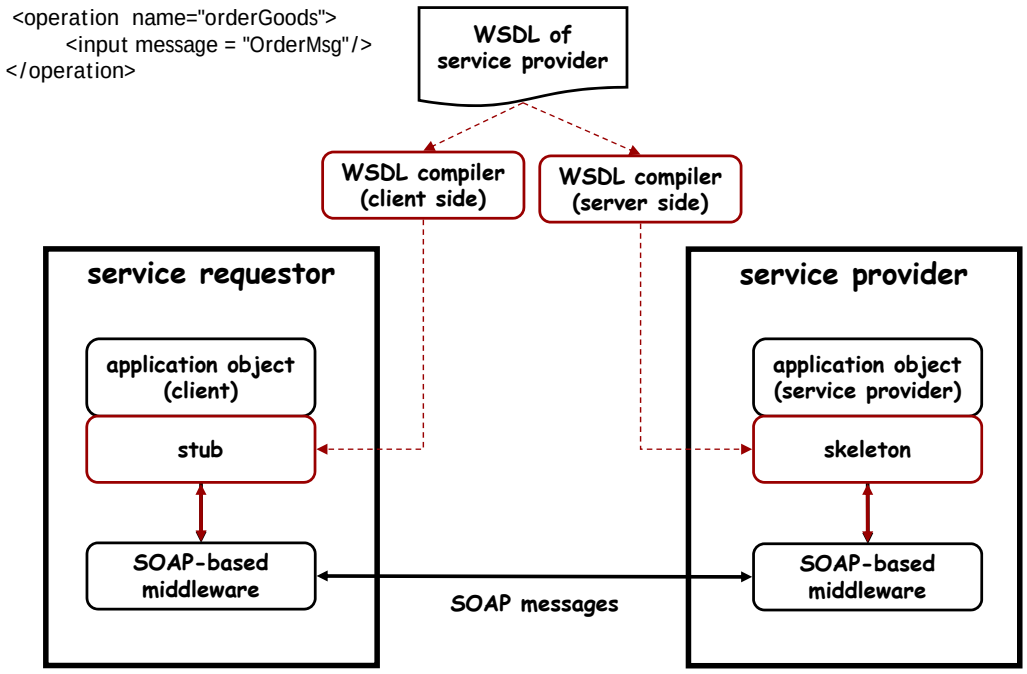
Uso di WSDL e SOAP

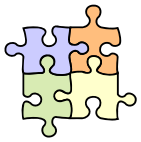
- I software development kit per la programmazione di web services possono utilizzare WSDL e SOAP per costruire automaticamente degli oggetti proxy – sia lato del servizio che lato client
  - in modo tale che il programmatore possa codificare la richiesta di servizi come invocazioni locali – usando il linguaggio di programmazione preferito



Uso di WSDL e SOAP

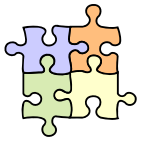
```
<operation  name="orderGoods">
      <input message = "OrderMsg"/>
</operation>
```





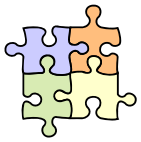
Uso di WSDL e SOAP

- Alcune funzionalità
  - generazione top-down del servizio – contract-first
    - da una specifica WSDL, viene generato il codice (skeleton) dell'implementazione del servizio (da completare)
  - generazione bottom-up del servizio – contract-last
    - dall'implementazione del servizio (ad esempio, un enterprise bean stateless), viene definito il servizio e generata la specifica WSDL
  - generazione del proxy lato client
    - da una specifica WSDL, viene generato il codice (stub) del proxy del servizio

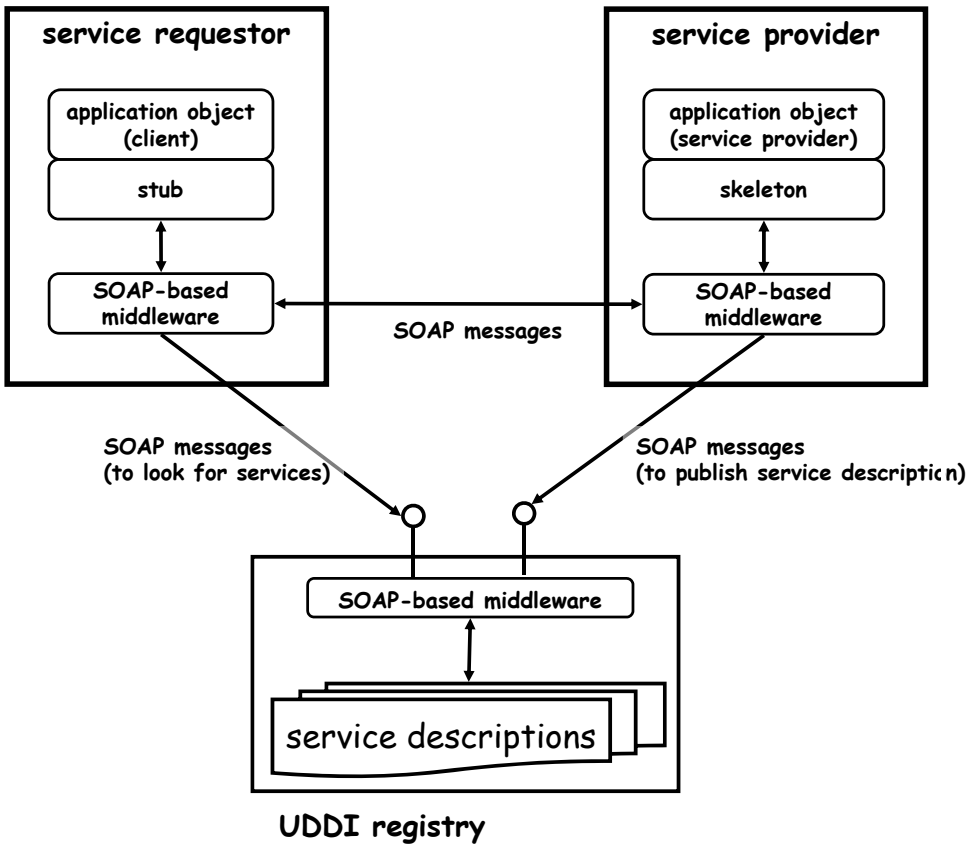


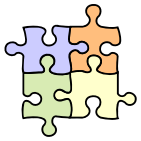
- Servizio di directory e UDDI

- Per rendere possibile un uso diffuso dei web services, è utile un meccanismo (possibilmente standardizzato anche questo) per il service registry, ovvero per pubblicare e ricercare servizi
- Un servizio di directory per i Web Services può essere basato su *UDDI – Universal Description, Discovery, and Integration*
  - due elementi
    - il registry
      - pagine bianche (indirizzi e contatti), pagine gialle (basate su classificazioni industriali), pagine verdi (con informazioni tecniche sui servizi)
    - API per l'accesso al registry



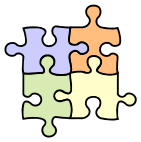
Directory per Web Services





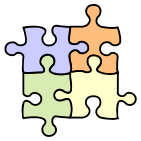
Uso del servizio di directory

- Alcuni possibili utilizzi del registry di servizi
  - uso “statico”
    - il registry è un repository di tutti i servizi software – condiviso tra più centri di sviluppo/utilizzo/gestione del software
      - ad es., descrive le caratteristiche fondamentali di ogni servizio, come modalità d’uso, locazione, SLA, statistiche, ...
    - applicato in sede di sviluppo (o deployment) di un’applicazione basata su servizi
    - fondamentale per consentire l’uso dei servizi web in più applicazioni
  - uso “dinamico”
    - usato a runtime, ad es., per supportare il brokering di servizi e consentire la selezione dinamica tra servizi



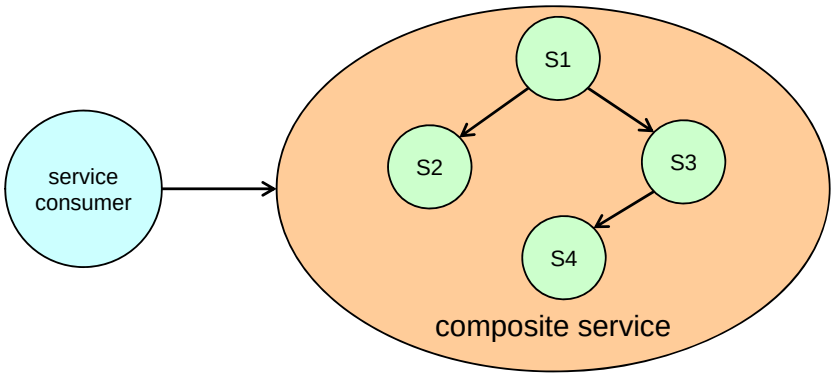
Servizi “statici”, “dinamici” e “semantici”

- Per utilizzare un servizio, un consumatore (client) deve determinare l’interfaccia del servizio, localizzarlo e poi invocarlo
  - nel binding (collegamento) statico, interfaccia e locazione del servizio sono determinate durante l’implementazione o il deployment del consumatore del servizio
  - nel binding dinamico, la locazione del servizio è determinata a runtime, con l’ausilio di un service registry
    - l’accoppiamento tra produttore e consumatore è ridotto
      - è possibile ad esempio che la locazione del servizio cambi – senza impatto sul consumo del servizio
    - c’è un overhead sulle prestazioni
  - talvolta, anche l’interfaccia del servizio è determinata a runtime
    - è però necessario che il consumatore e il fornitore del servizio abbiano un accordo predefinito sulla sintassi e la semantica delle interfacce

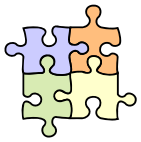


- Composizione di web services

- Un'altra caratteristica fondamentale dei WS è la possibilità di definire servizi web come *composizione* di altri servizi web

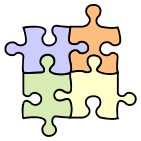


  - ad es., un servizio di prenotazione di viaggi organizzati può essere definito sulla base di altri servizi di prenotazione alberghiera, aerea, noleggio automobili, ...
    - questi servizi potrebbero anche essere offerti da più organizzazioni – tra loro indipendenti



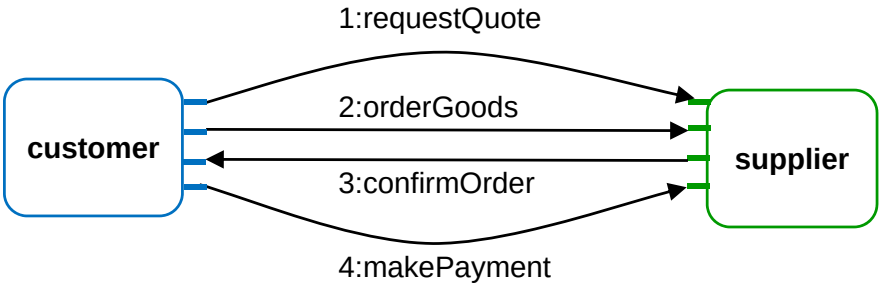
Composizione di web services

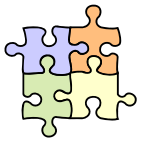
- La composizione di un insieme di servizi è basata sul coordinamento, la gestione e il sequenziamento dell'invocazione di tali servizi
  - per svolgere dei compiti complessi, i sistemi coinvolti devono infatti interagire in modo complesso
  - questa attività è ancora più complicata se i diversi sistemi devono essere debolmente accoppiati o addirittura completamente indipendenti tra di loro
- La composizione di servizi richiede la definizione di attività di collaborazione e scambio di messaggi tra i servizi componenti



Composizione di web services

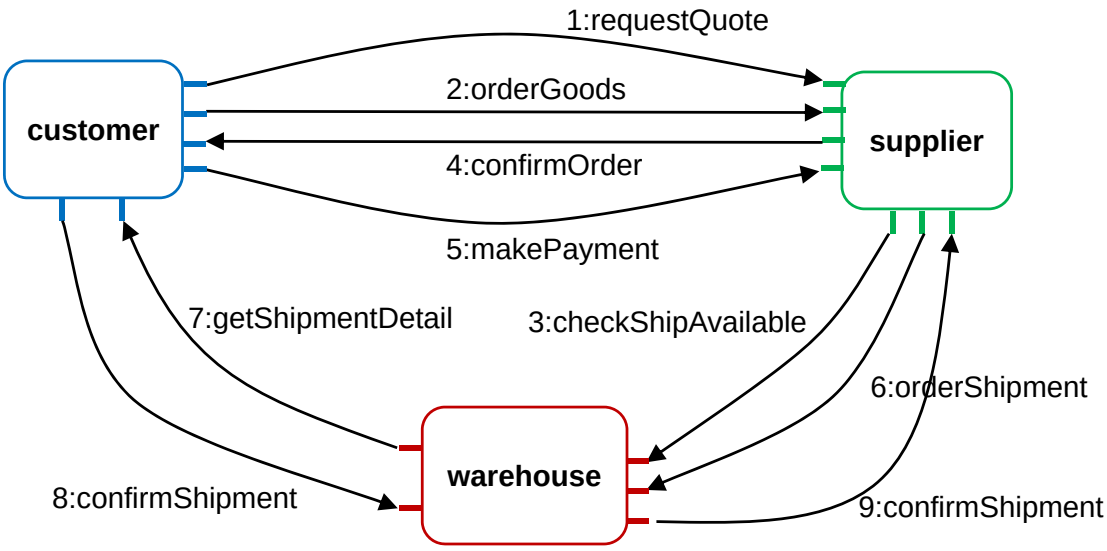
- La composizione di servizi richiede la definizione di attività di collaborazione e scambio di messaggi tra i servizi componenti
  - ad esempio, la conversazione tra due servizi per gestire l'approvvigionamento di prodotti

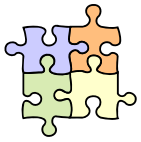




Composizione di web services

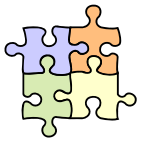
- La composizione di servizi richiede la definizione di attività di collaborazione e scambio di messaggi tra i servizi componenti
  - una conversazione più complessa: il fornitore delega la consegna al magazzino – il magazzino interagisce con il cliente per accordarsi sulla spedizione





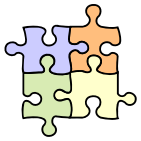
Composizione di web services

- La composizione di servizi richiede la definizione di attività di collaborazione e scambio di messaggi tra i servizi componenti
  - la composizione di servizi può essere definita e gestita sulla base di linguaggi di programmazione tradizionali – ad es., Java o C#
  - tuttavia, è più comune ed efficace l'uso di linguaggi specializzati per la composizione di servizi – in particolare, BPEL
  - più importante, la composizione può essere definita in termini di specifiche visuali – che possono essere fornite direttamente da "programmatori" di business (non tecnici) – e poi automaticamente tradotte in BPEL



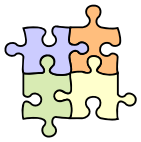
BPEL

- *Business Process Execution Language* (*for Web Services*)
  - un linguaggio di programmazione/scripting per WS
    - definisce primitive per la fruizione di servizi – come "invoke", "receive", "reply", "throw" e "wait"
    - contiene costrutti di controllo tradizionali – assegnazione, sequenza, istruzioni condizionali e ripetitive
    - contiene anche un costrutto per l'esecuzione concorrente di attività – "flow"
    - la sintassi è basata su XML
  - il codice BPEL può essere eseguito da un motore BPEL – ad es., eseguito da un application server – che coordina l'invocazione di servizi, usando il codice BPEL come uno script
  - esistono strumenti per la programmazione visuale di processi – che possono essere utilizzati direttamente da "programmatori" di business (non tecnici) – e possono generare codice BPEL

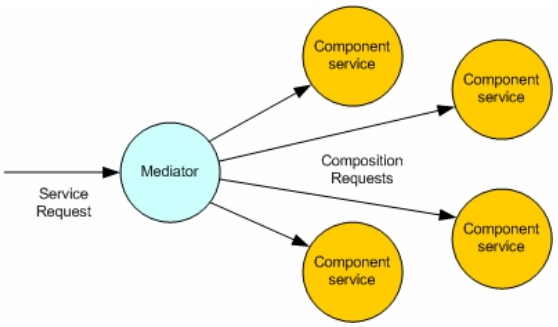


Modalità di composizione

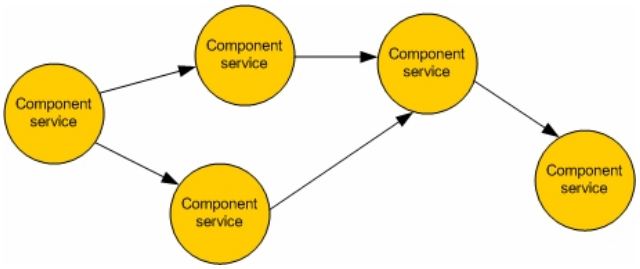
- Due modalità di composizione di servizi
  - *orchestrazione* – basata sull'uso di un mediatore
    - un processo centrale (il mediatore, che di solito è un web service) controlla e coordina l'esecuzione di operazioni che riguardano altri web services
    - i web services partecipanti potrebbero non sapere di agire nel contesto di un servizio più ampio
  - *coreografia* – di tipo peer-to-peer, non c'è un coordinamento centralizzato
    - la coreografia è uno sforzo collaborativo – i partecipanti sono a conoscenza del processo più ampio al quale stanno contribuendo
    - i web services partecipanti devono sapere come interagire tra loro e come coordinarsi



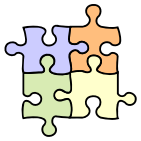
Orchestrazione e coreografia



orchestrazione

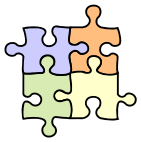


coreografia



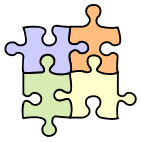
- Estensioni

- Oltre a SOAP, WSDL e UDDI, esistono numerosi standard (oltre 100!) per WS – indicati complessivamente come *WS-**
  - alcuni di questi standard sono relativi ad aspetti specifici dell'interazione tra web services – ad esempio
    - i WS sono solitamente stateless – ma è possibile definire servizi stateful sulla base di WS-Resource
    - WS-Addressing si occupa di indirizzamento, in modo neutrale rispetto ai meccanismi di trasporto
  - molte estensioni sono relative alla gestione delle qualità dei servizi – ad esempio
    - sicurezza – WS-Security
    - affidabilità della comunicazione – WS-ReliableMessaging
    - transazioni – WS-Coordination, WS-Transaction
    - requisiti, capacità e preferenze – WS-Policy
  - alcune estensioni sono complementari, altre in competizione



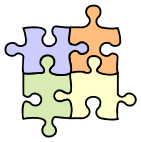
Estensioni e profili

- Diversamente da quanto accade per gli standard fondamentali, le estensioni WS-* non sono accettate o sostenute allo stesso modo dai produttori di software
  - per favorire l'interoperabilità, le estensioni sono state recentemente organizzate in *profili*
  - ad esempio
    - il profilo WS-I Basic – "I" sta per Interoperability – fa riferimento soprattutto a SOAP, WSDL, UDDI, XML, XML-Schema, HTTP, HTTPS e WS-Addressing
    - il profilo Reliable-Secure-Profile – estende il profilo Basic con l'adozione di WS-ReliableMessaging, WS-SecureConversation, WS-MakeConnection e WS-SecurityPolicy



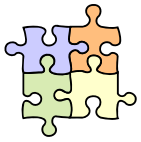
- Infrastruttura runtime per servizi

- L'esecuzione di un'applicazione a servizi richiede la presenza di un'opportuna infrastruttura runtime (middleware) – che implementa o gestisce gli standard di interesse tra quelli descritti
  - l'infrastruttura minimale per i web services può essere fornita da un application server – un AS può normalmente fungere anche da contenitore di web services
  - intuitivamente, però, per far interagire web services realizzati con tecnologie diverse (uno degli obiettivi fondamentali dei web services!) è necessaria anche un'infrastruttura che realizzi il "bridging" dei diversi application server coinvolti
  - in questo caso l'infrastruttura di deployment è di solito chiamata un *Enterprise Service Bus* (*ESB*)



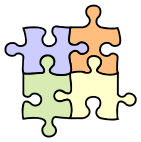
- Web services SOAP: Discussione

- I web services SOAP sono una soluzione tecnologica all'interoperabilità tra componenti/servizi software distribuiti ed eterogenei
  - sulla base di alcuni standard fondamentali – e numerose estensioni
  - è possibile la composizione di servizi – nonché la gestione di diversi attributi di qualità
  - l'uso di SOAP garantisce indipendenza e trasparenza dai protocolli – sia di trasporto, che quelli relativi alle qualità, dichiarate negli header SOAP
  - l'uso di WSDL sostiene ulteriormente l'indipendenza dalla piattaforma per la definizione e la fruizione dei web services
  - la disponibilità di strumenti di sviluppo e motori per l'esecuzione di web services nasconde agli sviluppatori molta della complessità di questi standard



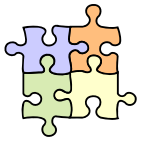
Web services SOAP: Discussione

- I web services SOAP presentano anche alcuni inconvenienti
  - la semplicità con cui è possibile esporre componenti software esistenti come web services può dar luogo ad un uso scorretto di questa tecnologia
  - in pratica, si possono verificare problemi di interoperabilità tra un servizio e i suoi client – ad esempio
    - in caso di interpretazioni diverse degli standard di base
    - in caso di utilizzo di standard diversi dallo stack WS-*
    - se nelle interfacce viene fatto uso di tipi di dati nativi con i web services bottom-up
  - anche l'uso estensivo di XML può porre problemi – ad esempio
    - di efficienza, nella traduzione da/verso strutture di dati native nelle implementazioni
    - XML-Schema è troppo ricco, e non è supportato completamente in tutte le implementazioni



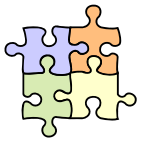
* Web Services REST

- Una delle principali critiche alla tecnologia dei web services SOAP – chiamati *"big"* – è la loro "pesantezza" – con le relative implicazioni su prestazioni e scalabilità
  - esistono però anche delle tecnologie a servizi più "leggere" – che sostengono ancora interoperabilità, offrono migliori garanzie su prestazioni e scalabilità, e sono solitamente più semplici da programmare
    - attenzione, queste tecnologie sono probabilmente peggiori nei confronti di altre qualità – ad es., affidabilità e sicurezza – o in termini di infrastruttura – ad es., rispetto alla composizione di servizi
  - tra queste tecnologie, la più diffusa è quella dei web services REST (o RESTful)



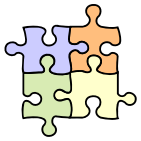
REST

- *REST* (*Representational State Transfer*) è un paradigma per la realizzazioni di applicazioni web, che permette la gestione di risorse per mezzo del protocollo HTTP
  - si noti che lo stile REST è stato definito prima dei web services REST (e indipendentemente da essi)
- Lo stile architetturale REST è stato proposto da Roy Fielding – uno degli autori di HTTP
  - Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use



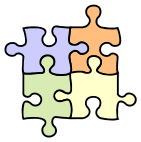
REST

- Il concetto centrale nello stile REST è quello di “risorsa”
  - il web gestisce un insieme di risorse – una *risorsa* è ogni elemento informativo di interesse, con un identificatore univoco
    - ad esempio, una risorsa “corso di Architetture Software” con URI (qui inventata) *http://www.uniroma3.it/corsi/asw*
- Un’applicazione client può accedere a una risorsa tramite la sua URI – in questo caso
  - al client viene restituita una *rappresentazione* della risorsa
  - questa rappresentazione definisce un nuovo *stato* per l’applicazione client
  - ovvero, l’applicazione client cambia (*trasferisce*) il proprio stato ogni volta che accede a una risorsa



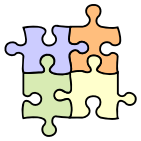
Caratteristiche dello stile REST

- Caratteristiche dello stile architetturale REST
  - è uno stile architetturale di tipo client-server
  - i servizi sono di tipo stateless – sostiene scalabilità e disponibilità
  - è possibile fare caching delle risposte dei servizi – sostiene scalabilità e prestazioni
  - è un'architettura a strati – un client in genere non sa se sta comunicando con un server che eroga effettivamente il servizio oppure con un intermediario
  - uso di un'interfaccia uniforme tra componenti – l'accesso uniforme alle risorse sostiene flessibilità
  - code on demand (opzionale)



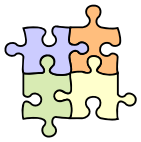
Servizi nello stile REST

- Lo stile architetturale REST consente di descrivere l'architettura generale del World Wide Web – tuttavia, può essere applicato in particolare anche nella definizione di *web services nello stile REST* (*RESTful WS*)
  - attenzione, lo stile REST è adeguato per servizi relativamente semplici, orientati alla gestione di risorse informative



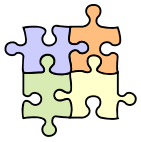
Principi per servizi nello stile REST

- Principi che guidano la definizione di un servizio REST
  - identificazione delle risorse tramite URI
    - un servizio REST espone un insieme di risorse – che sono l'obiettivo delle interazioni con i client – identificate da URI
  - interfaccia uniforme
    - le risorse vengono manipolate in modo uniforme, tramite quattro operazioni predefinite: PUT, GET, POST e DELETE
  - messaggi auto-descrittivi
    - le risorse sono disaccoppiate dalle loro rappresentazioni – e il loro contenuto può essere acceduto sulla base di più formati – ad es., testo, XML, JSON, PDF, JPEG
    - ogni richiesta contiene informazioni sufficienti a descrivere come il server possa elaborare la richiesta
    - ogni risposta contiene informazioni sufficienti a descrivere come il client possa elaborare la risposta



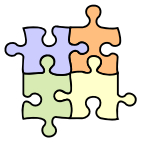
Principi per servizi nello stile REST

- Principi che guidano la definizione di un servizio REST
  - rappresentazione ipermediale
    - se un client deve poter accedere risorse correlate a una sua richiesta, queste sono comunemente identificate nella rappresentazione restituita
  - interazioni stateful basate su collegamenti ipertestuali
    - le interazioni con le risorse sono di per sé stateless
    - tuttavia, sono possibili interazioni stateful, sulla base di un trasferimento esplicito dello stato delle conversazioni – ad es., riscrittura di URI, cookie, campi nascosti



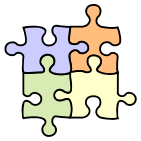
Esempio di servizio nello stile REST

- Un esempio di servizio nello stile REST
  - il servizio gestisce una o più collezioni omogenee di risorse
    - ad esempio, un insieme di corsi e un insieme di docenti
  - il servizio definisce, per ciascuna collezione, un'URI di base – chiamata *collection URI*
    - ad es., *http://www.uniroma3.it/corsi* e *http://www.uniroma3.it/docenti*
  - ciascuna istanza di risorsa ha un'URI – chiamata *element URI*
    - ad es., *http://www.uniroma3.it/corsi/asw* e *http://www.uniroma3.it/docenti/luca.cabibbo*
  - le operazioni offerte dal servizio sono messe in corrispondenza con le operazioni HTTP GET, PUT, POST e DELETE
  - in particolare, la rappresentazione restituita dall'operazione GET è espressa in un formato di interscambio opportuno – ad es., testo, HTML, XML oppure JSON



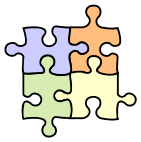
Esempio di servizio nello stile REST

- Un servizio nello stile REST – operazioni riferite a una *collection URI*
  - GET – restituisce un elenco di tutti gli elementi della collezione
  - PUT – sostituisce la collezione con un'altra collezione
  - POST – crea un nuovo elemento della collezione – e gli assegna una nuova URI (e la restituisce)
  - DELETE – cancella l'intera collezione



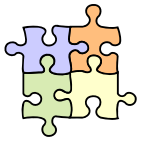
Esempio di servizio nello stile REST

- Un servizio nello stile REST – operazioni riferite a un'*element URI*
  - GET – restituisce una rappresentazione di uno specifico elemento della collezione
  - PUT – crea un nuovo elemento della collezione, oppure lo aggiorna
  - POST – considera l'elemento della collezione come un'altra collezione, e ne aggiunge un elemento (in modo analogo a quanto fa POST con riferimento a una collection URI)
  - DELETE – cancella l'elemento della collezione



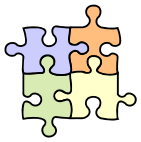
- Web services REST: Discussione

- I web services REST presentano alcuni vantaggi rispetto ai web services SOAP
  - l'adozione dei servizi REST è di solito più semplice che non quella dei servizi SOAP – poiché fanno riferimento a standard noti e le infrastrutture necessarie sono spesso già disponibili
  - in particolare, è molto basso lo sforzo richiesto per scrivere client di servizi REST
  - grazie all'uso di URI e collegamenti ipertestuali, è possibile scoprire risorse web senza l'uso di un registry centralizzato
  - i servizi REST sono più leggeri di quelli SOAP – ad es., JSON richiede un overhead minore di quello di XML
  - le soluzioni per la scalabilità del web – ad es., basate su caching, clustering e load balancing – possono essere applicate anche per servizi REST stateless
  - il vantaggio principale di REST vs. SOAP è la *semplicità*



Web services REST: Discussione

- I web services REST vengono percepiti come più semplici da programmare dei web services SOAP – perché le opzioni possibili sono in genere inferiori – ma la realizzazione di applicazioni REST complesse è di solito più difficile – ad esempio
  - la gestione di diversi attributi di qualità deve essere gestita direttamente dal programmatore di servizi REST
  - i servizi REST vengono acceduti tramite HTTP, in modo sincrono – per questo, l'integrazione di applicazioni è più problematica
  - usando le sole operazioni HTTP, può essere difficile stabilire una buona definizione dell'interfaccia di un servizio REST
  - i servizi REST non hanno un'interfaccia descritta esplicitamente – questo può rendere più difficile la fruizione di servizi REST e la gestione della loro evoluzione
  - il vantaggio principale di SOAP vs. REST è la *completezza*



* Discussione

- La tecnologia dei Web Services, per la sua generalità e il supporto per l'interoperabilità, si è stabilita come tecnologia preferita per lo sviluppo e l'integrazione di applicazioni di tipo enterprise
  - tuttavia, questo non significa che i WS destituiranno le tecnologie precedenti, e in particolare quelle a componenti
  - piuttosto, il ruolo dei Web Services è quello di complementare queste tecnologie di successo – fornendo, ove necessario, meccanismi standard per l'interoperabilità – aggiungendo in questo modo valore alle piattaforme di middleware esistenti
  - infatti, i Web Services, con la loro focalizzazione sull'integrazione, favoriscono il riuso di funzionalità, e riducono il lock-in ("rimanere vincolati") al middleware – consentendo agli sviluppatori di usare il middleware più opportuno per soddisfare le loro necessità, senza però precludere l'interoperabilità con altri sistemi