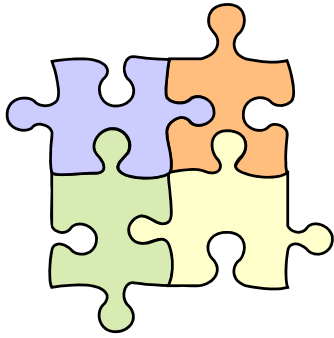


Luca Cabibbo

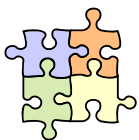


## Architetture Software

# Web Services

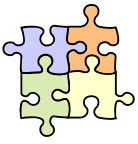
Dispensa ASW 460

ottobre 2011



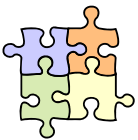
## - Fonti

- [Papazoglou] Web Services – Principles and Technology
- [ACKM] Web Services – Concepts, Architectures and Applications
- [POSA4] Pattern-Oriented Software Architecture – A Pattern Language for Distributed Computing



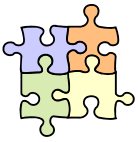
## - Obiettivi e argomenti

- Obiettivi
  - introdurre i web services e i relativi standard
- Argomenti
  - introduzione
  - web services
  - servizi nello stile rest
  - discussione



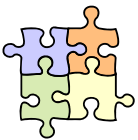
## \* Introduzione

- Il middleware orientato ai **servizi** costituisce la tecnologia emergente per lo sviluppo e l'integrazione di applicazioni distribuite
  - la tecnologia a servizi dominante è rappresentata dai **Web Services**
- L'**architettura orientata ai servizi (SOA)** fornisce il contesto metodologico (e di business) in cui utilizzare al meglio le tecnologie basate su servizi
  - l'architettura orientata ai servizi sarà discussa in un'altra dispensa



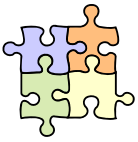
## Intuizione: Dal Web ai Web Services

- Il **Web** consente l'accesso a dati/servizi tramite pagine web
  - l'accesso è pensato per client umani
  - difficile l'accesso diretto da parte di client software
- I **Web Services** hanno l'obiettivo di fornire servizi sul web a client software
  - 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



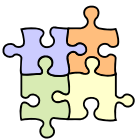
## Web Services e SOA

- **Web Services**
  - costituiscono un modo di esporre le funzionalità di un sistema informatico, 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



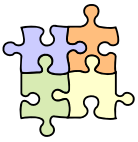
## \* Web Services

- Il paradigma di interazione basato su *servizi* (*services* o *e-services*) – con il relativo middleware – è una tecnologia per lo sviluppo e l'integrazione di applicazioni distribuite
  - evoluzione delle architetture distribuite
  - con l'obiettivo di supportare l'interoperabilità tra componenti in esecuzione su piattaforme diverse
  - con una costante attenzione agli aspetti di business
  - enfasi sull'interoperabilità tra componenti eterogenei, sulla base di protocolli standard aperti e universalmente accettati
  - generalità dei meccanismi di comunicazione – sia sincroni che asincroni
  - flessibilità nell'organizzazione dei suoi elementi (servizi)
  - la tecnologia a servizi “dominante” è, attualmente, quella dei *Web Services* (WS)



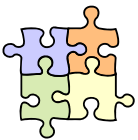
## WS e interoperabilità

- 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 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



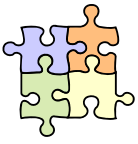
## WS e standard

- I Web Services sono basati su un insieme di standard tecnologici per l'interoperabilità – indipendenti dalle piattaforme e neutrali rispetto ad essi
  - in precedenza sono state 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 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** 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



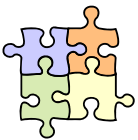
## Che cos'è un Web Service

### □ Un *Web Service* [Papazoglou]

- è un modulo o componente software, auto-contenuto e auto-descrittivo, accessibile mediante Internet, in modo indipendente dalla piattaforma
- 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”

si noti la doppia caratterizzazione:  
- la prima è una caratterizzazione tecnica  
- la seconda è di business

con i WS e le SOA, c'è sempre  
un'attenzione a entrambi gli aspetti



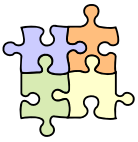
## Che cos'è un Web Service

### □ Un *Web Service* [Papazoglou]

- è un modulo o componente software, auto-contenuto e auto-descrittivo, accessibile mediante Internet, in modo indipendente dalla piattaforma
- 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”

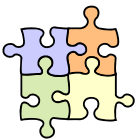
### □ Ad esempio, un WS potrebbe rappresentare

- un compito di business auto-contenuto – ad es., effettuare un bonifico bancario
- un intero processo di business – ad es., l'acquisto automatizzato di forniture d'ufficio
- un'applicazione – ad es., gestione di assicurazioni sulla vita
- l'accesso (come servizio) a una risorsa – ad es., a cartelle mediche

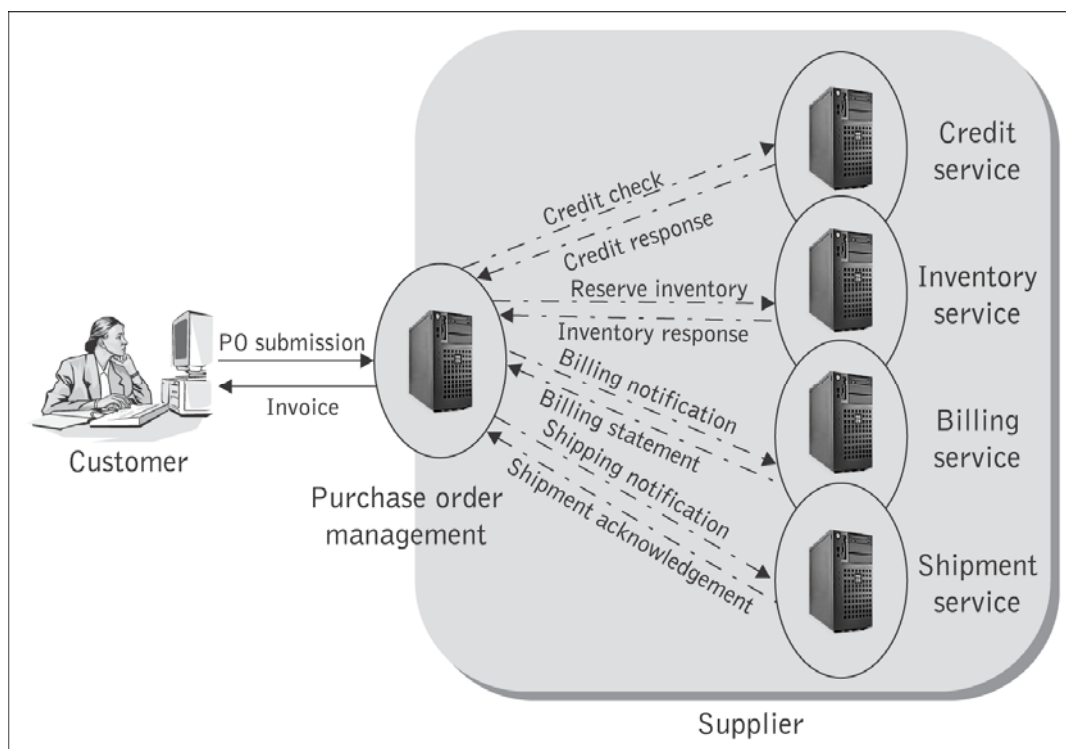


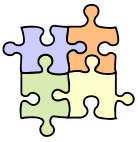
## Che cosa sono i Web Services

- I Web Services formano gli elementi costitutivi per la creazione di applicazioni distribuite
  - una caratteristica fondamentale dei web services è che possono essere messi in corrispondenza e composti
    - un buon web service può essere impiegato nella realizzazione di più applicazioni
  - i web services favoriscono 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)



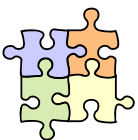
## Un esempio - gestione di ordini d'acquisto





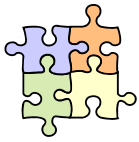
## Che cosa sono i Web Services

- 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 più servizi – per definire un nuovo servizio
- queste funzionalità sono fornite agli sviluppatori sulla base di opportuni (e numerosi) standard

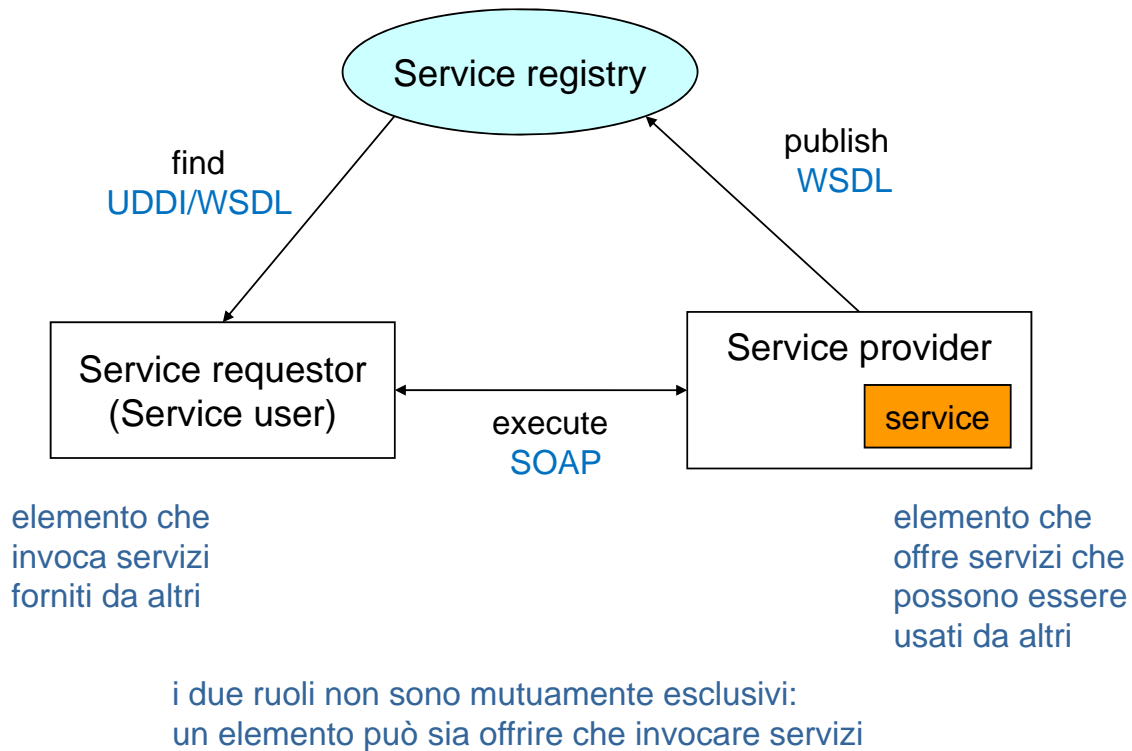


## - Standard per Web Services

- Nella tecnologia dei Web Services, gli standard rivestono un ruolo fondamentale
  - esistono numerosi standard per i web services, 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 – BPEL e le estensioni WS-\*
  - i principali responsabili degli standard per i Web Services sono il consorzio OASIS (*Organization for the Advancement of Structured Information Standards*) e il W3C (*World Wide Web Consortium*)

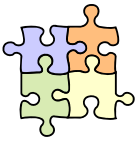


## Interazione tra servizi



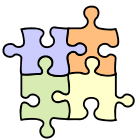
## Standard per WS

- I WS sono basati su un insieme di standard, basati su **XML**
  - l'adozione di XML come “linguaggio di trasporto” sostiene l'indipendenza dei vari standard dalla piattaforma e dai linguaggi di programmazione
- I tre standard fondamentali per i WS – basati su XML
  - **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



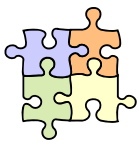
## Standard per WS

- E' possibile comprendere le specifiche dei WS ragionando sui problemi che i WS intendono affrontare
  - questo consente di descrivere un'infrastruttura minimale per i WS
  - in realtà, sopra a questa infrastruttura sono state definite numerose estensioni – a cui ci si riferisce come WS-\*



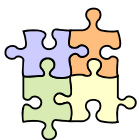
## - Sintassi

- Gli standard sui 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
  - c'è però anche uno svantaggio, l'overhead introdotto da XML nella codifica, trasmissione, decodifica – legato al fatto che le informazioni sono comunemente codificate in XML in modo “proliso”



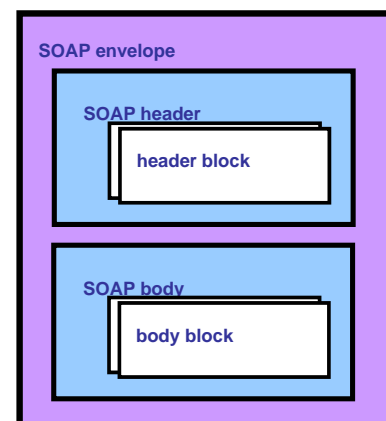
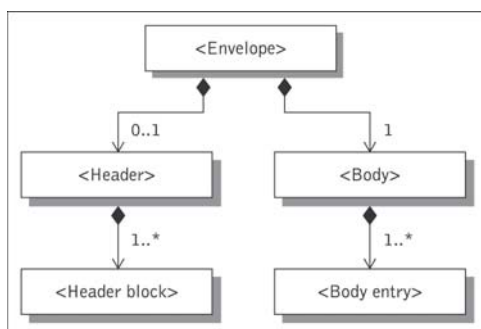
## - Interazione con i Web Services e SOAP

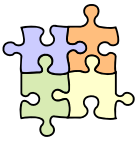
- E' necessario un meccanismo per consentire l'interazione tra un web service e i suoi "client" – ci sono tre aspetti da considerare
  - formato comune per i messaggi scambiati
  - supporto per diverse forme specifiche di interazione – ad es., nello stile procedurale (RPC) e nello stile del messaging
  - compatibilità con diverse modalità di trasporto dei messaggi – ad es., basato su TCP/IP, HTTP, SMTP, ... – 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

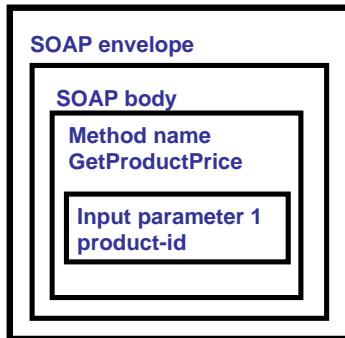
- Un **messaggio SOAP** è costituito da una busta (**envelope**), che contiene un'intestazione e un corpo
  - il corpo (**body**) racchiude le informazioni che il messaggio vuole effettivamente trasmettere – ad es., un documento, oppure il nome di un metodo invocato e i relativi parametri
  - l'intestazione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



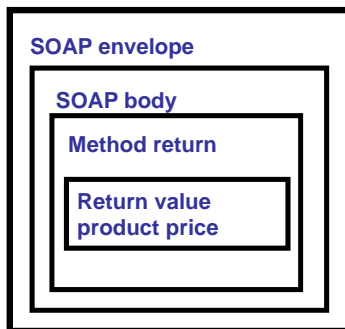


## Messaggi SOAP

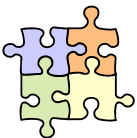
- Ad es., coppia di messaggi SOAP in un'interazione in stile RPC



```
<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> 450R60P </product-id >
    </m:GetProductPrice >
  </env:Body>
</env:Envelope>
```

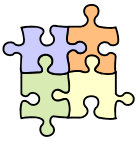


```
<env:Envelope
  xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
  xmlns:m="http://www.plastics_supply.com/product-prices">
  <env:Header>
    <!-- - Optional context information -->
  </env:Header>
  <env:Body>
    <m:GetProductPriceResponse>
      <product-price> 134.32 </product-price>
  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

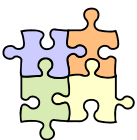
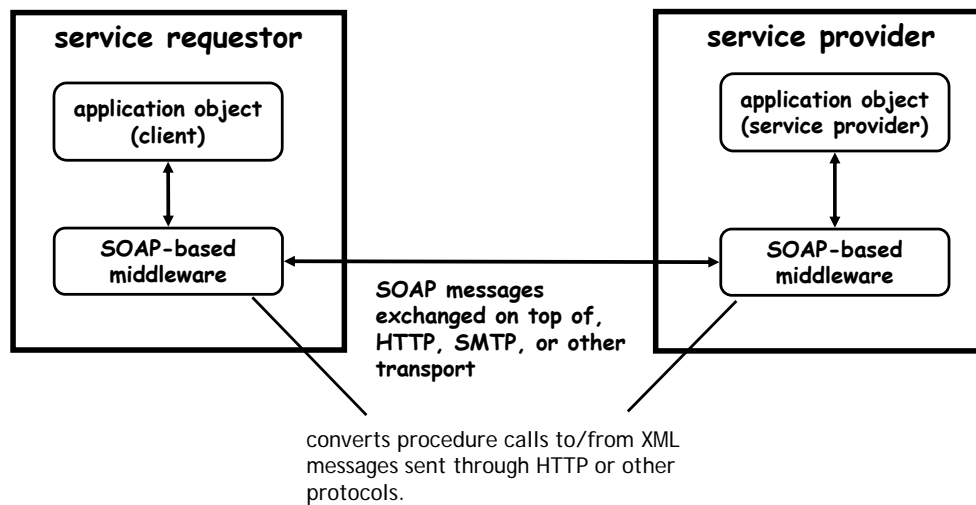


## Modelli di comunicazione

- Il modello di comunicazione di SOAP prevede diverse varianti, basate su due proprietà
  - **stile di comunicazione**
    - lo stile *RPC* consente di definire messaggi in corrispondenza alla richiesta o alla risposta relativa all'esecuzione di un'operazione
    - lo stile *document (message)* consente di scambiare documenti (messaggi) che contengono dati XML
  - **stile di codifica (encoding)**
    - 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

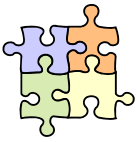


## Interazione con SOAP



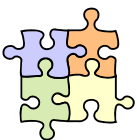
## Osservazioni su SOAP

- 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 – di questo aspetto si occupa WSDL
- 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 e estensibile



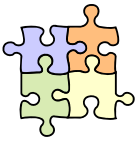
## - Descrizione di servizi e WSDL

- E' 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



```

<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>

```

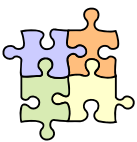
Abstract data type definitions

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="Purchasing"
    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"/>

      <!-- further 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>

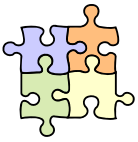
```

Bind an abstract operation to this implementation and

map the abstract input and output messages to these concrete messages

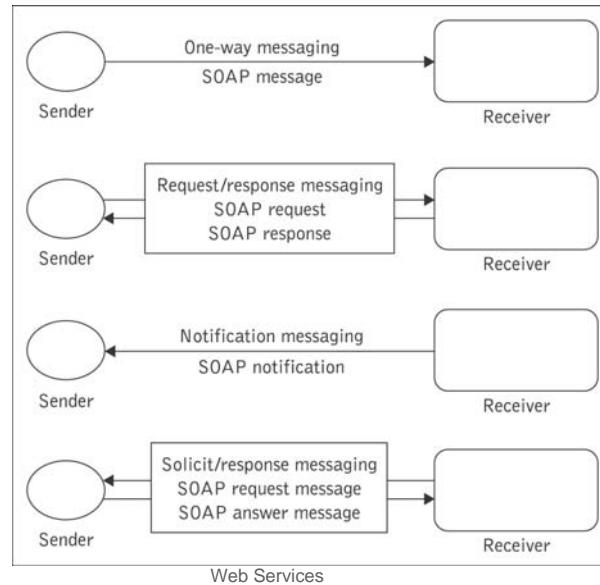
Service name

Network address of service



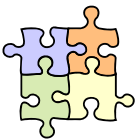
## Message Exchange Pattern

- 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*)
  - poiché ogni operazione può avere un messaggio di input e/o uno di output, i MEP possibili sono quattro



31

Luca Cabibbo – ASw



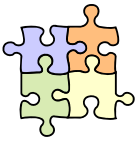
## Message Exchange Pattern

- *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

32

Web Services

Luca Cabibbo – ASw



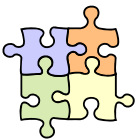
## Message Exchange Pattern

### □ *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

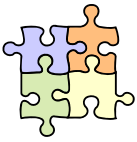
### □ *Solicit/response*

- un'operazione in cui il servizio invia un messaggio a un client, e poi ne riceve una risposta – è dunque l'opposto di 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)



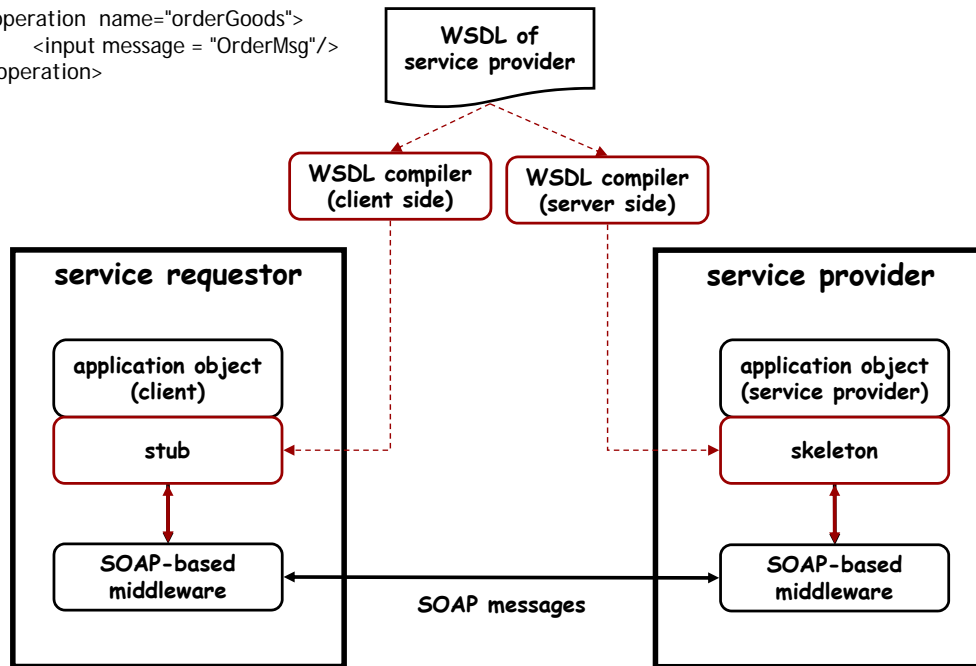
## Uso di WSDL e SOAP

- I software development kit per WS 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

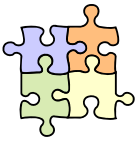
```
<operation name="orderGoods">  
  <input message = "OrderMsg"/>  
</operation>
```



## Uso di WSDL e SOAP

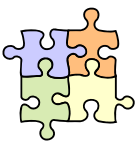
### □ Alcune funzionalità

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

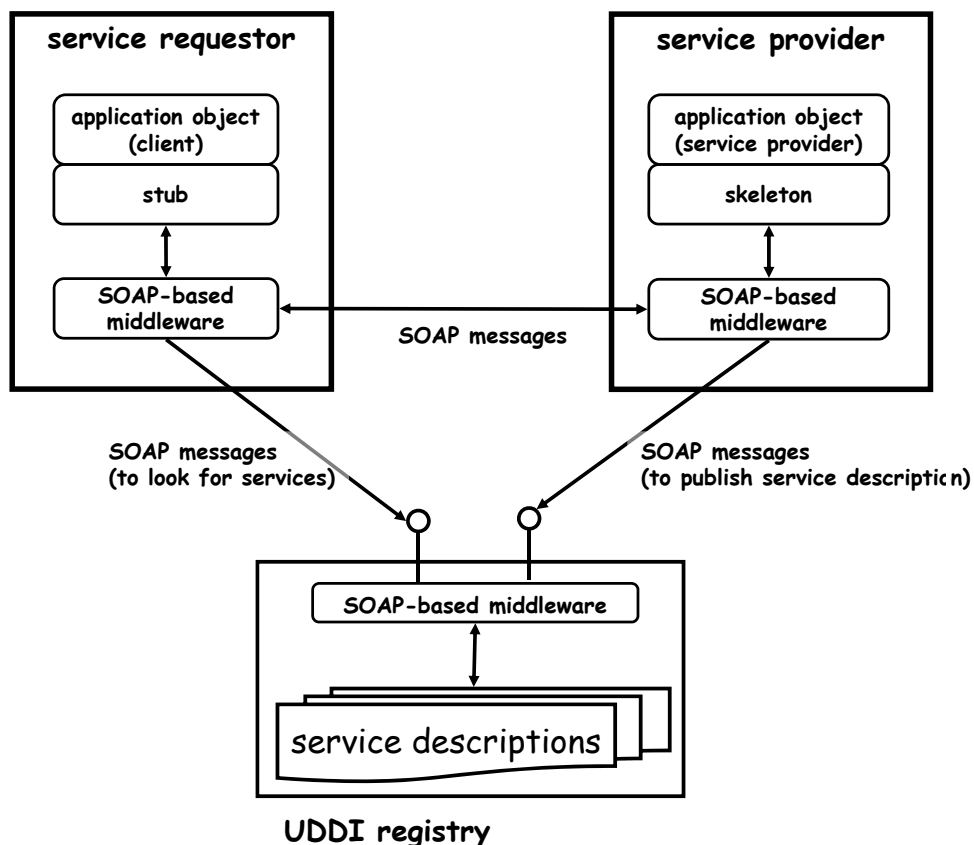


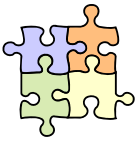
## - Servizio di directory e UDDI

- Per rendere possibile un uso diffuso dei WS, può essere 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



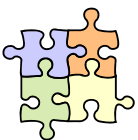
## Directory per Web Services





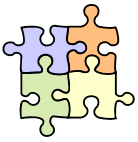
## Uso del servizio di directory

- 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 in più applicazioni
  - uso “dinamico”
    - usato a runtime, ad es., per supportare il brokering di servizi e consentire la selezione dinamica tra servizi



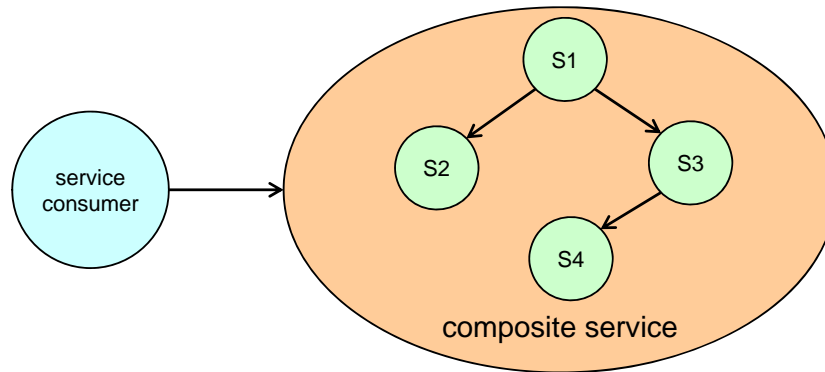
## Web services “statici” e “dinamici”

- 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
  - tuttavia, è possibile anche un binding dinamico, in cui interfaccia e locazione del servizio sono determinate 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 impatto negativo sulle prestazioni – ma il service registry può essere utilizzato per il load balancing
    - è però necessario che il consumatore e il fornitore del servizio abbiano un accordo predefinito sulla sintassi e la semantica delle interfacce

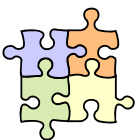


## - Composizione di servizi

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

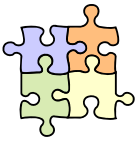


- ad es., un servizio di prenotazione di viaggi organizzati può essere definito sulla base di altri servizi di prenotazione alberghiera, aerea, noleggio automobili, ...
  - questi servizi potrebbero essere anche offerti da più organizzazioni – tra loro indipendenti



## Composizione di servizi

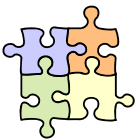
- La composizione di servizi richiede la definizione di attività di collaborazione e scambio di messaggi tra i servizi componenti
  - la composizione di servizi può essere definita e gestita sulla base di linguaggi di programmazione tradizionali – ad es., Java o C#
  - l'uso di linguaggi specializzati per la composizione di servizi – ad es., BPEL – è più efficace e comune
  - 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

### □ **BPEL (BPEL4WS) – Business Process Execution Language for Web Services**

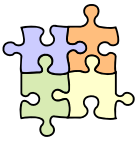
- è un linguaggio di programmazione basato su XML
  - definisce primitive per la fruizione di servizi – come “invoke”, “receive”, “reply”, “throw” e “wait”
  - contiene costrutti di controllo tradizionali – assegnazione, sequenza, istruzioni condizionali e ripetitive
  - contiene un costrutto per l’esecuzione concorrente di attività – “flow”
- 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



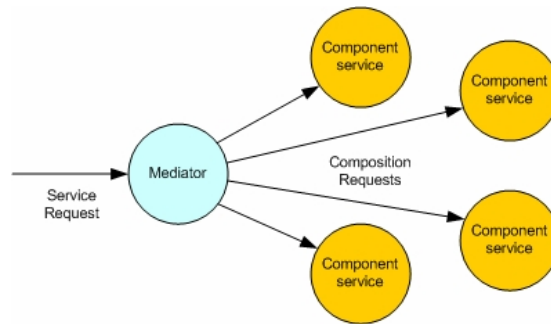
## Modalità di composizione

### □ Due modalità di composizione di servizi

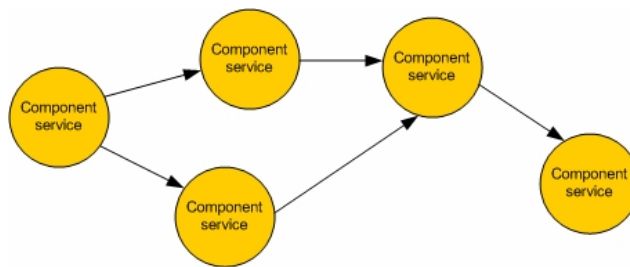
- **orchestrazione** – basata sull’uso di un mediatore
  - un processo centrale (il mediatore, che di solito è un web service) controlla e coordina l’esecuzione di operazioni che riguardano altri web services
  - i web services partecipanti potrebbero non sapere di agire nel contesto di un servizio più 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



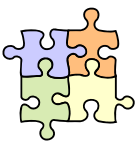
# Orchestrazione e coreografia



orchestrazione

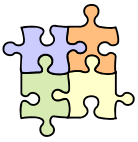


coreografia



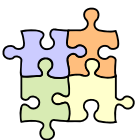
## - Estensioni

- Esistono numerosi standard (oltre 100!) per Web Services oltre a SOAP, WSDL e UDDI – complessivamente indicati come **WS-\***
  - per la gestione di ambiti come
    - sicurezza – WS-Security
    - affidabilità comunicazione – WS-ReliableMessaging
    - transazioni – WS-Coordination, WS-Transaction
    - requisiti, capacità e preferenze – WS-Policy
    - gestione dello stato – WS-Resource
    - ...
  - alcuni sono complementari, altri in competizione
- Attenzione
  - diversamente da quanto accade per gli standard fondamentali, non tutti i produttori di software accettano o sostengono allo stesso modo gli standard WS-\*
  - recentemente organizzati in **profili**, per favorire l'interoperabilità



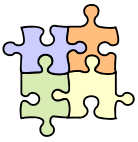
## - Infrastruttura runtime per servizi

- Evidentemente, l'esecuzione di un'applicazione a servizi richiede la presenza di un'opportuna infrastruttura runtime (middleware) – che implementi o gestisca gli standard di interesse tra quelli descritti
  - l'infrastruttura minimale per i web services può essere fornita dagli application server – che normalmente fungono anche come contenitori 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 può essere un *Enterprise Service Bus (ESB)*
    - torneremo su questo argomento nella dispensa sulle SOA



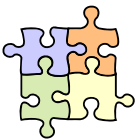
## \* Servizi nello stile REST

- Una delle principali critiche alla tecnologia dei Web Services basata sugli standard visti finora – chiamati Web Services SOAP oppure di tipo “*big*” – è la sua “pesantezza” – con le relative implicazioni sulle prestazioni
  - esistono però anche delle tecnologie a servizi più “leggere” – che sostengono ancora interoperabilità, e offrono migliori prestazioni
    - attenzione, queste tecnologie sono probabilmente peggiori nei confronti di altre qualità, come affidabilità e sicurezza
  - tra queste tecnologie, la più diffusa è quella dei web services nello stile REST



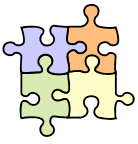
## REST

- **REST** (*Representational State Transfer*) è un paradigma per la realizzazioni di applicazioni Web, che permette la gestione di risorse per mezzo del protocollo HTTP
  
- 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, la risorsa il “corso di Architetture Software” con URL <http://www.uniroma3.it/corsi/asw>
  - un’applicazione client può accedere a una risorsa tramite la sua URL
    - 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



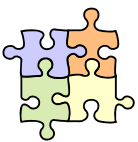
## REST

- Dalle parole di Roy Fielding – uno degli autori di HTTP, che ha coniato il termine REST
  - 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



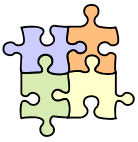
## Caratteristiche dello stile REST

- **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
  - code on demand (opzionale)
  - uso di un'interfaccia uniforme tra componenti – l'accesso uniforme alle risorse sostiene flessibilità



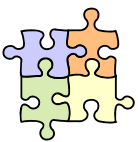
## Principi dell'interfaccia

- **Principi che guidano la definizione di un'interfaccia REST**
  - le risorse sono identificate – le richieste specificano individualmente le risorse di interesse – mediante URI
  - manipolazione delle risorse – le richieste di un client sono in genere relative alla gestione o manipolazione di risorse – a secondo dei permessi di accesso, un client può accedere, creare, modificare oppure cancellare risorse
  - messaggi auto-descrittivi
    - 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
  - rappresentazione ipermediale – se un client deve poter accedere risorse correlate ad una sua richiesta, queste sono comunemente identificate nella rappresentazione restituita



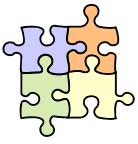
## Servizi nello stile REST

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



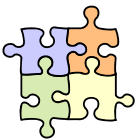
## Esempio di servizio nello stile REST

- 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 per una collezione ha un'URI di base – chiamata *collection URI*
    - ad es., <http://www.uniroma3.it/corsi> e <http://www.uniroma3.it/docenti>
  - ogni istanza di risorsa ha un'URI – chiamata *element URI*
    - ad es., <http://www.uniroma3.it/corsi/asw> e <http://www.uniroma3.it/docenti/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



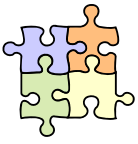
## Esempio di servizio nello stile REST

- 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



## \* Discussione

- La tecnologia dei Web Services, per la sua generalità e il supporto per l'interoperabilità, si è stabilita come tecnologia preferita per molte 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