- ***Lezione 23***

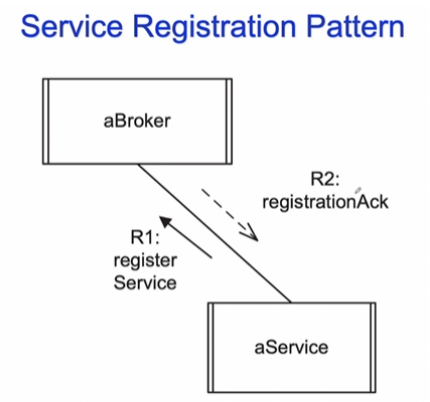
Esistono dei pattern/protocolli di interazione tra le varie entità che giocano un ruolo nelle architetture service oriented usati a livello di progettazione per capire
come tali protocolli possano esser concretamente utilizzati facendo uso di tecnologie implementative.
Partiamo dai pattern di interazione di base. Nell'archiettura service oriented ritroviamo i due ruoli introdotti con il client/server: il consumatore di servizio (client) e il service provider (server).
Tra le due entità si interpone il service broker che regola e permette la comunicazione tra i due.
Si parla quindi di Software Architectural Broker Patterns.
Per poter utilizzare il broker il service provider, una volta realizzato un servizio, deve registrarsi presso il broker al fine di rendere il servizio disponibile (per soddisfare il principio di Discoverability!) ed il service consumer utilizzerà il broker (registro di servizi) per capire se esiste un servizio che fa al caso suo.
Dopo questa fase iniziale esistono diverse modalità di interazione tra i due: una modalità diretta per cui il consumer comunica direttamente con il provider una volta acquisite le informazioni necessarie dal broker oppure una interazione mediata per cui ogni richiesta da parte del consumer passa attraverso il broker.
In tutto ciò gioca un ruolo fondamentale il concetto di trasparenza, due tipologie:

- Platform Transparency: l'utilizzo del servizio deve essere possibile per qualsiasi piattaforma (sistema operativo etc..) e non si necessita di conoscere i dettagli implementativi che permettono l'esecuzione del servizio sul suo ambiente
- Location Transparency: se il provider decide si spostare il servizio su una porta/interfaccia di rete differente, i consumatori non necessitano di essere informati ma soltanto il broker

Si parla in questo senso di brokered communication in quanto al fine di garantire tale trasparenza deve essere il broker a gestire il tutto (es. se cambia interfaccia di rete sarà informato il broker che provvederà affinché il consumer possa continuare a fare richieste normalmente). Ma come anticipato, per utilizzare la brokered communication, è necessario introdurre il pattern di base che è il Service Registration Pattern (il provider deve registrare le informazioni sul servizio presso il broker).
Il Service Registration Pattern rappresenta semplicemente una richiesta che il service provider fa al broker per registrare il proprio servizio tramite un messaggio dove il provider comunica nome del servizio, una descrizione e l'interfaccia di rete presso cui è disponibile.

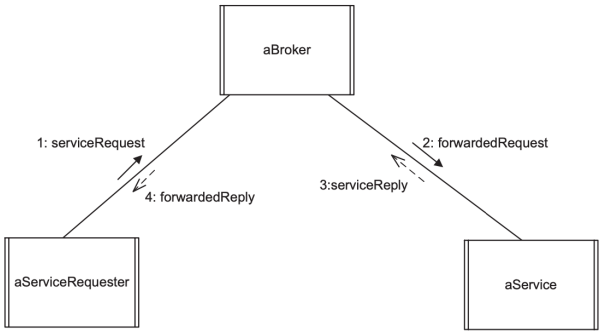
Ciascun pattern sarà illustrato dal punto di vista visuale tramite UML 2.
Si ricordi che quando abbiamo introdotto i diagrammi di interazione UML per descrivere le interazioni tra oggetti abbiamo detto che esistono due tipi di diagrammi che dal punto di vista espressivo sono del tutto equivalenti: sequence diagram e collaboration diagram.
Entrambi sappiamo avere potere espressivo equivalente, ma il collaboration diagram è tipicamente usato in fase di progettazione in quanto è una rappresentazione di interazione tra

oggetti più compatta (e in fase di progettazione molti più oggetti rispetto alla fase di requisiti).

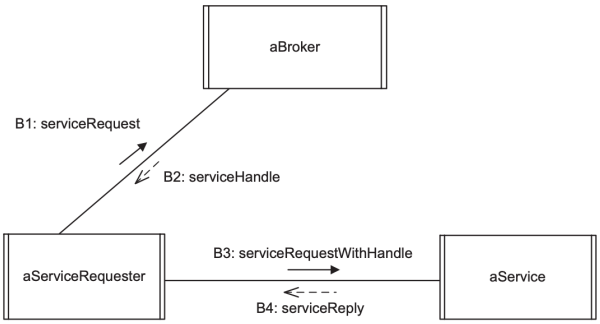


Qui un esempio di collaboration diagram, dove vengono visualizzati solo gli oggetti coinvolti nell'interazione e i messaggi che si scambiano. Manca una componente fondamentale che invece è presente nel sequence diagram: l'ordine temporale (nei sequence diagram messaggi scambiati dall'alto verso il basso).
Sappiamo però che sono rappresentazioni equivalenti, quindi è immediato passare da sequence a collaboration ma più difficile fare il contrario se manca ordine temporale.
Quindi alla specifica di ogni messaggio scambiato è associato un sequence number (R1 e R2 in figura) per ricostruire anche l'ordine temporale e passare al sequence.
Ma perché usiamo UML 2? Gli oggetti sono rappresentati normalmente da rettangoli dove tramite sintassi si può anche risalire alla classe che ha creato l'oggetto (nome_oggetto riferimento_oggetto: nome_classe). Qui manca la specifica della classe in quanto facciamo riferimento a istanze di un servizio e i rettangoli hanno barre laterali: ciò sta a significare che i due elementi (broker e service) vengono eseguiti in modo concorrente (in UML 1 non era possibile rappresentare ciò)
Inoltre in UML 2 il collaboration diagram è stato rinominato comunication diagram.
Nel caso in figura il servizio viene registrato presso il broker e una volta ricevuto l'ACK dal broker allora il servizio è stato aggiunto all' "archivio" del broker.
Una volta registrato il servizio i potenziali consumatori possono interagire con il broker e capire dalla descrizione se il servizio è di loro interesse o meno.
A questo punto allora vi sono le due possibilità di comunicazione diretta o intermediata.

Broker Forwarding Pattern (pagine bianche): il broker si interpone tra provider e consumer anche dopo la fase iniziale per l'utilizzo del servizio.
Si parla di pagine bianche perché il meccanismo alla base è simile al meccanismo che si utilizzava con gli elenchi telefonici dove si poteva recuperare il numero di una persona a partire da dettagli come nome e cognome.
Lo stesso avviene per questo protocollo: si assume che il consumer conosca il nome del service provider ma non sappia l'interfaccia di rete per usare il servizio, quindi lo chiede al provider.
In particolare il client manda un messaggio identificando il servizio richiesto, il broker riceve la richiesta e recupera (in base alle informazioni presenti nel suo archivio dove sono registrati i servizi) l'interfaccia di rete presso cui il servizio è reso disponibile, inoltra quindi la richiesta al servizio, prende la risposta e la inoltra infine al consumer.

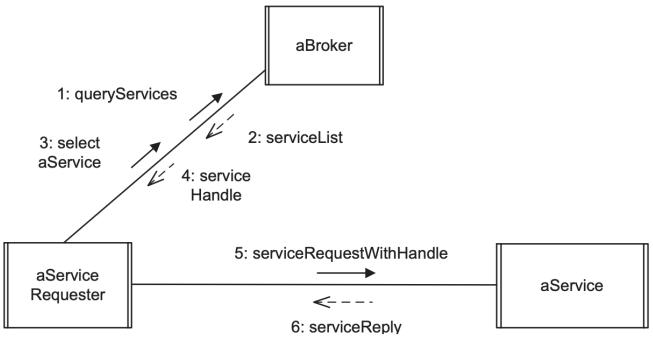


Un altro tipo di pattern basato sempre sullo scenario di pagine bianche è il Broker Handle Pattern. In questo caso ancora il broker è inizialmente coinvolto per ottenere le informazioni della location del service provider, ma invece di inoltrare il tutto manda le informazioni al consumer affinché la successiva comunicazione sia diretta con il provider

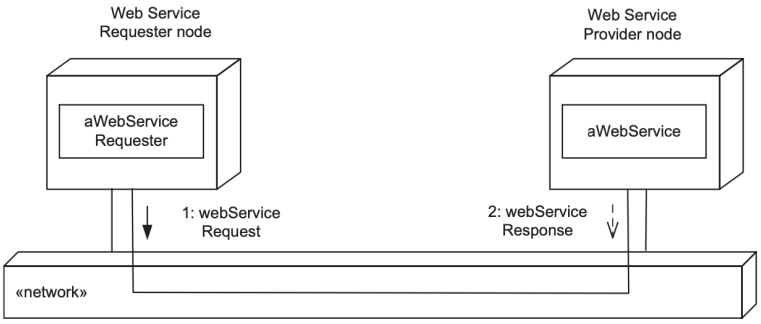


Il vantaggio nell'utilizzo del primo è che è garantita la location trasparency: se cambia interfaccia di rete lo viene a sapere il broker che aggiornando il registro di servizi nasconderà automaticamente il tutto al consumer.
Ciò non funziona più con il broker handle pattern, se si cambia l'interfaccia le successive richieste dal consumer che non lo saprà andranno a vuoto e sarà necessaria una nuova comunicazione con il broker. Il vantaggio del secondo però è che si scambiano meno messaggi, maggiore efficienza (ogni interazione due messaggi uno richiesta uno risposta contro i quattro del Broker Forwarding Pattern)
Se utilizzo il servizio poche volte conviene usare il primo approccio, ma se lo uso molto frequentemente conviene il secondo.
Diverso lo scenario del tipo pagine gialle. Mentre nelle pagine gialle ad es. ordine alfabetico per nome, cognome e indirizzo, con le pagine gialle si introducono le categorie di servizi (es. idraulici, elettricisti etc… scelgo quello che fa al caso mio magari perché vicino a casa mia e prendo il numero di telefono).
Quindi la differenza è che non so esattamente il servizio specifico che mi serve ma il tipo di servizio, dalle pagine gialle ottengo un elenco di quella tipologia e potrò scegliere quello che voglio.
Si parla di Service Discovery Pattern (pagine gialle), dove la differenza sta nell'introduzione di una serie ulteriore di messaggi.
Il consumer richiede al broker una tipologia di servizio (queryServices), il broker restituisce una lista di servizi che soddisfano la richiesta (sempre tramite l'archivio interno dove i

servizi sono registrati), il service requester sceglie il servizio da utilizzare. Da questo punto in poi si può interagire nelle due modalità viste prima: in figura Broker Handle in quanto il broker manda l'interfaccia di rete e il client potrà interagire direttamente con il provider, altrimenti si può utilizzare broker forwarding.

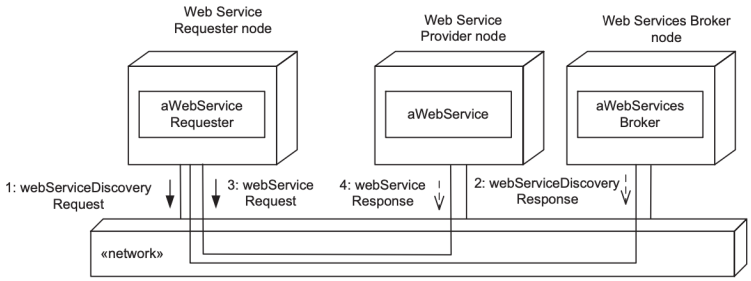


Ma in base a cosa potrei scegliere uno specifico servizio da una lista generica?
In base a vari criteri al di là della funzionalità, come QoS (quality of service).
Vediamo ora quale potrebbe essere un buon supporto tecnologico dal punto di vista implementativo orientato alla realizzazione su applicazioni basate su service oriented.
Si fa riferimento alla tecnologia Web Services.
Un Web Service fa riferimento all'uso di questi servizi facendo uso di protocolli standard internet e usando come linguaggio per lo scambio di dati (richieste, risposte etc..) tramite l'uso di protocolli XML.
Una tecnologia implementativa per architettura service oriented deve garantire tutte le funzionalità viste a livello architetturale come broker, descrizione del servizio…
Si introducono quindi degli standard (come SOAP) per mettere a disposizione queste funzionalità di base, che come detto fanno uso di XML come formato di serializzazione e di protocolli standard internet, in particolare tutte le richieste mediate dal broker tra consumer e provider fanno uso dello standard HTTP e la relativa porta 80.
Si evitano quindi quei problemi evidenziati con tecnologie come CORBA che fanno uso di protocolli proprietari (dove in ambiente distribuito facendo uso di firewall era difficile permettere la comunicazione).
In particolare viene citato il protocollo SOAP Simple Object Access Protocol, utilizzato per permettere al consumer di inviare un messaggio al provider come richiesta di esecuzione di una certa operazione.
SOAP come detto è basato sul formato XML e consiste in tre parti: una "busta" che definisce un framework per descrivere cosa c'è nel messaggio e come processarlo, un insieme di regole per codificare i tipi di dato scambiati tra consumer e provider e un modo per rappresentare queste chiamate di procedura remota.
L'uso del Web Services ha come idea di base di rendere disponibili i servizi tramite interfaccia di rete raggiungibile tramite una piattaforma web (non solo usando HTTP come vedremo ma nella maggior parte dei casi si).

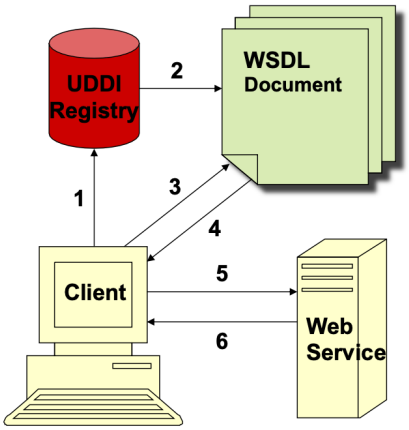


Qui un esempio di utilizzo della tecnologia Web Service. La richiesta viene veicolata al service provider e allo specifico servizio messo a disposizione tramite l'uso del protocollo SOAP.
Il diagramma in figura (su) fa riferimento a un diagramma di implementazione che ancora non avevamo visto, in particolare un deployment diagram dove si combina la descrizione della piattaforma di esecuzione e l'allocazione di componenti software su questi elementi della piattaforma.
In particolare tre nodi (i parallelogrammi) che rappresentano nodi di esecuzione il primo quello che esegue il consumer (contenente l'oggetto in esecuzione, la componente awebservice requester), il secondo del provider (contenente l'oggetto … aWebService) e quello sotto quello che rappresenta l'interfaccia di rete che collega i due (basata ovviamente sul web e quindi protocolli internet, in particolare HTTP).
Quando il consumer ha le informazioni necessarie per identificare il servizio che necessita (oggetto aWebService Requester) grazie al protocollo SOAP posso confezionare la richiesta da inviare al servizio, che potrà elaborare la richiesta e rispondere ancora usando SOAP.
Ma come ottenere queste informazioni necessarie (la sintassi affinché possa avvenire la richiesta)? Posso farlo grazie alla descrizione del servizio, che sappiamo essere inviata al broker dal service provider all'atto di registrazione del servizio.

Si introduce quindi nell'ambito della tecnologia WebService un ulteriore standard per scrivere la descrizione del servizio.
Questo standard si chiama WSDL Web Service Description Language, linguaggio ancora basato su XML che permette di fornire SOLO le informazioni necessarie del servizio tramite un documento WSDL al broker e quindi al potenziale consumatore (principio di discovery).
Il documento WSDL oltre alla descrizione contiene ovviamente anche il broker handle, il riferimento che il requester che il consumer può utilizzare per interagire direttamente con il provider.
Quindi la tecnologia WebService mette a disposizione un protocolo per definire i messaggi da scambiare (SOAP) e un linguaggio per descrivere i servizi (WSDL).
Ci manca il terzo elemento, come realizzare il concetto di service registry (broker)?
Viene introdotto un particolare framework, UDDI Universal Description Discovery and Integration.
Tale framework realizza quindi tutte le funzioni base che deve svolgere il broker.
Quindi in generale quando parliamo di tecnologia implementativa a supporto di architetture service oriented si fa riferimento a come realizzare il broker, come descrivere il servizio e come interagire con il servizio, nel caso web services UDDI, WSDL e SOAP.

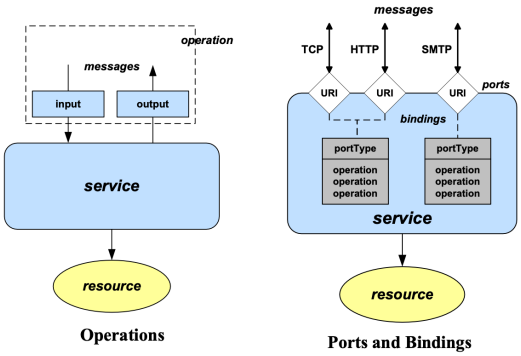


Qui un esempio di Web Service in cui interviene anche il broker su un altro nodo remoto, il consumatore chiede al broker informazioni sul servizio per ottenere eventualmente il documento WSDL e interagire quindi col provider.
Tutto il processo riassunto in questa slide:



1. **Client queries UUDI registry to locate service**
2. **Registry refers client to WSDL document**
3. **Client accesses WSDL document**
4. **WSDL provides data to interact with web service**
5. **Client sends SOAP-message request**
6. **Web service returns SOAP-message response**

WSDL mette a disposizione solo gli elementi di base per utilizzare il servizio (quindi niente roba al di là dell'interfaccia tipo QoS), ossia l'insieme di operazioni che possono essere invocate (del tutto simile all'interfaccia pubblica di una classe che mi dice i metodi che posso invocare, la differenza sta nel fatto che in un'applicazione object oriented la richiesta è veicolata nello stesso spazio di memoria mentre nel secondo caso la richiesta è remota), la network location dove presso cui raggiungere il servizio ed inviare la relativa richiesta SOAP.



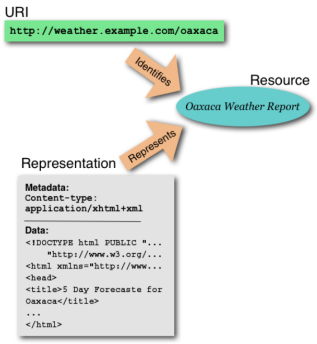
Un documento WSDL come si vede è diviso in due parti separate (livello di astrazione):

- a sinistra livello di dettaglio relativo alle operazioni. WSDL mette a disposizione le operazioni in termini di messaggi input e output (esiste in realtà anche il messaggio di fault oltre a input e output se errore). Questa parte descritta in XML.
- a destra invece si mostra come le operazioni sono concretamente messe a disposizione sull'interfaccia di rete: es. la richiesta è inviata tramite HTTP e raggiunge uno specifico indirizzo di rete URI e lo strumento utilizzato per mettere a disposizione le informazioni a livello di interfaccia di rete è il portType.

Un portType è null'altro che un insieme di operazioni messe a disposizione su uno o più URI (endpoint). Come detto si utilizza maggiormente HTTP (sincrono) per condividere questi messaggi ma si potrebbero utilizzare anche altri protocolli come TCP o SMTP (asincrona).
Nel tempo sono stati introdotti anche ulteriori contributi che non sono alternativi alla tecnologia Web Service ma che si affiancano ad essa, uno di questi è REST Representational State Transfer.
Questo più che essere una tecnologia implementativa (come WebService) è uno stile architetturale (come gestire l'architettura di un sistema software distribuito usando sempre protocolli standard internet, in particolare HTTP).
REST mette a disposizione un interfaccia di rete (API) per accedere ad un insieme di risorse messe a disposizione sulla piattaforma web (queste risorse sono di diversi tipi e possono anche essere servizi web).

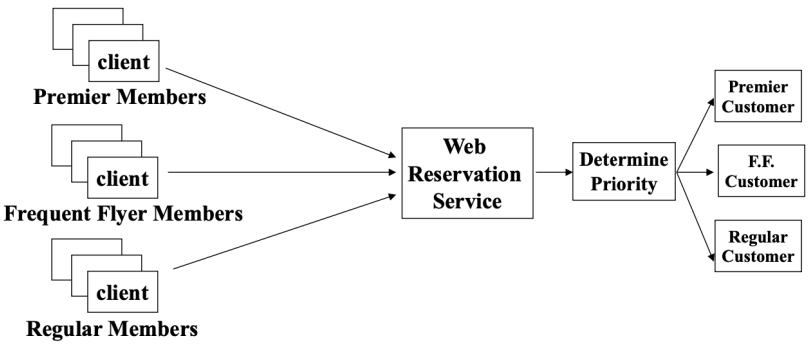
La cosa fondamentale è che in questo caso non si usa HTTP solo per garantire lo scambio di messaggi tra i partecipanti all'architettura distribuita, ma anche per veicolare queste interazioni → si fa uso solo delle operazioni di base HTTP per accedere e utilizzare la risorsa.
Caratteristiche: ancora una volta client/server quindi un server mette a disposizione una risorsa con cui si può interagire facendo uso solo di HTTP, ambiente Stateless in quanto non si salva lo stato nell'interazione c/s, vengono però messe a disposizione delle cache per migliorare l'efficienza in una rete basata su REST, interfaccia uniforme in quanto le interazioni sono basate solo e soltanto sui 4 verbi di base HTTP ossia GET PUT POST (aggiornare) DELETE (CRUD!), REST si fonda sul concetto di risorsa che deve essere identificata con URL specifico (o URI), inoltre le rappresentazioni delle risorse sono interconnesse usando gli URL.

Tutto si basa sulle risorse web e ogni entità distinguibile è risorsa (sito web, pagina html, documento xml, web service, etc…) e sono identificate da un URL.
La risorsa è tipicamente rappresentata facendo uso di un documento XML.



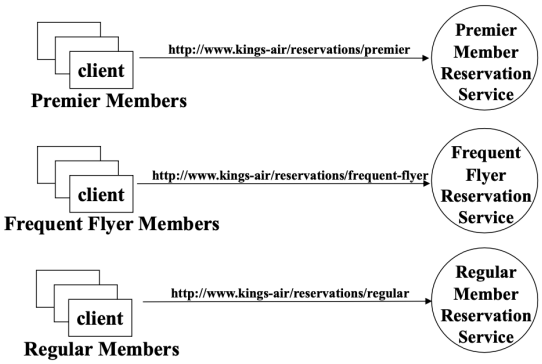
Per questo abbiamo detto che REST non rappresenta un'alternativa a Web Service (egli stesso li fornisce attraverso non più un'interfaccia proprietaria ma un meccanismo diverso). Per interagire con le risorse si utilizzano le RESTful API che usano le 4 operazioni CRUD HTTP in base al "contesto": se si deve prendere una Collection di risorse (es. /users nell'URL) o un singolo Item (es. /users/{id}). In particolare Post può essere usato solo nelle collezioni per aggiungervi un item mentre Get sia sulle collezioni che sui singoli item (nel primo caso per ottenere la lista di elementi e nel secondo il singolo elemento), invece PUT e DELETE utilizzabili solo negli item.
Ma cosa mi cambia se uso il design convenzionale per Web Services o se ci aggiungo REST? Vediamolo con un esempio concreto dove confrontiamo una progettazione basata su una e sull'altra.
Si vuole realizzare un servizio di prenotazione biglietti aerei online, e si vuole fornire un servizio di qualità differente in base al tipo di utenti (premier member servizio immediato, frequent flyer members servizio spedito, e gli altri servizio regolare).

Nell'approccio convenzionale si fa uso di un singolo URL per mettere a disposizione il servizio che offra le tre tipologie di accesso.
L'approccio basato su REST invece fa uso di più URL.



Nel primo caso regular, frequent e premier accedono tutti allo stesso URL e in base al tipo si determina la priorità della richiesta e inoltra quindi la richiesta alle parti di servizio che gestiscono la priorità.

Il problema di ciò è che si sta assumendo che usare tre URL diversi è più costoso, e l'affidabilità del servizio non è supportata visto che a questo punto quel singolo URL è un central point of failure e bottleneck.



Nel secondo caso invece non c'è l'esigenza di porre attenzione sul numero di URL che si utilizza → si evitano i problemi.