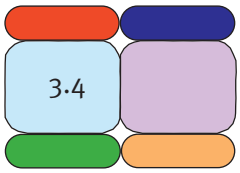
UN'INTRODUZIONE RAGIONATA AL MONDO DEI WEB SERVICE

Barbara Pernici
Pierluigi Plebani

Il paradigma del Service Oriented Computing è visto come una rivoluzione nella comunità informatica e i Web Service una sua realizzazione. La possibilità di vedere il Web come un grande sistema informativo in cui sono forniti innumerevoli servizi offre agli utenti finali un potentissimo strumento che va al di là del mero scambio di informazioni che al momento rappresenta il Web. Si propone, qui, una panoramica sui Web Service, le attuali tecnologie e i problemi aperti su cui la ricerca si sta muovendo.



1. INTRODUZIONE

L'esplosione di Internet, avvenuta negli anni novanta, è andata di pari passo con il forte interesse che le aziende hanno mostrato verso questo nuovo strumento. Il potenziale pubblico che può attraverso Internet relazionarsi con le aziende è vastissimo e, di conseguenza, anche la possibilità di concludere trattative commerciali aumenta in modo considerevole. Si è assistito, quindi, a una integrazione di sistemi informativi sia orizzontalmente sia verticalmente che ha portato alla proliferazione di termini quali *e-commerce*, *e-business*, *e-government*, *e-procurement*. In realtà, già negli anni ottanta, dopo aver visto una forte diffusione dei sistemi informativi all'interno delle aziende, si è sentita la necessità di poter integrare piattaforme differenti. Questo tipo di esigenza sorge non solo tra aziende ma addirittura all'interno della medesima azienda in cui, per motivi storici e organizzativi, diverse divisioni hanno operato scelte tecnologicamente differenti nella realizzazione del proprio sistema informativo.

Ora, complice Internet, le aziende perseguono il medesimo obiettivo in modo da poter automatizzare le filiere produttive o creare dei *marketplace* virtuali. Tale integrazione ha richiesto, e tuttora richiede, notevoli sforzi sia di tipo organizzativo che di tipo tecnologico in grado di mediare tra la necessità delle aziende in questione di scambiarsi informazioni e la necessità opposta di mantenere una certa autonomia.
Dal punto di vista organizzativo è plausibile immaginare che ogni azienda possa fornire servizi e al contempo utilizzarne. In particolare, i servizi forniti, che rappresentano le funzionalità che l'azienda intende esportare verso l'esterno, delineano il confine tra ciò che del proprio sistema informativo aziendale è pubblico e ciò che al contrario rimane privato. D'altro canto, dal punto di vista tecnologico affinché questa interoperabilità sia fattibile, le aziende devono accordarsi su un linguaggio comune di descrizione dei servizi in modo tale da riconoscere cosa un sistema mette a disposizione. Ciò deve essere accompagnato anche da un meccanismo di ricerca dei servi-

zi esistenti e dalla possibilità di utilizzare il Web come canale di trasmissione.
Sebbene gli aspetti organizzativi siano molto importanti per capire le politiche che sottendono alla decisione su quando una funzionalità possa essere esportata e come questo impatta sul *core-business* aziendale, nel presente articolo ci si concentrerà unicamente sugli aspetti tecnologici soffermandoci, quindi, sui Web Service quale soluzione tecnologica adatta all'interoperabilità dei sistemi. Va altresì sottolineato che il ruolo dei Web Service non si limita solo a un discorso di interoperabilità, bensì permette di descrivere nuovi servizi realizzati *ad hoc*, sempre però con l'intento di fornire una soluzione *platform-independent*. Il primo passo è, quindi, quello di separare nettamente, come del resto avviene anche in modelli di programmazione basati su componenti, la logica di presentazione da quella che è la logica applicativa. I Web Service, infatti, si occupano solamente della logica applicativa, e sarà, quindi, il fruitore del servizio a presentare i dati ottenuti utilizzando il servizio con stili e grafica propri.
Si immagini, ad esempio, che l'azienda ACME voglia, all'interno della propria pagina Web, inserire una piccola casella in cui mostrare il valore della quotazione in tempo reale delle proprie azioni. L'ACME per avere questo tipo di informazione dovrà necessariamente richiedere i dati periodicamente al sistema informativo della Borsa utilizzando un servizio appositamente fornito da quest'ultima. Tale servizio richiederà, quindi, in *input* il simbolo dell'azione e restituirà, in *output*, la quotazione attuale. Il Web master di ACME non dovrà fare altro che invocare il servizio e far sì che una porzione della pagina Web dell'azienda sia alimentata dai dati ottenuti in risposta.
Prima però di discutere dei dettagli prettamente tecnici, e per inquadrare meglio lo spazio d'azione, si vuole dare una definizione di più ampio respiro di Web Service per non correre il rischio, come spesso avviene, di limitare la descrizione di questa tecnologia alla descrizione delle tre tecnologie di base quali *Web Services Description Language* (WSDL) [1], *Simple Object Access Protocol* (SOAP) o [7] e *Universal Description, Discovery and Integration* (UDDI) o [11, 12] che comunque verranno affrontate in seguito. A tal proposito, si partirà da una architettura di riferimento, la *Service Oriented Architecture* (SOA) [3, 4], grazie alla quale è possibile identificare gli scopi, gli utilizzi e gli sviluppi futuri che caratterizzano i Web Service e si potrà notare come l'architettura si inserisce perfettamente all'interno delle considerazioni fatte in precedenza, in particolare di quelle relative al mondo *business-to-business* (B2B).
Focalizzando l'attenzione sul concetto di servizio è ovvio immaginare, anche alla luce di quanto detto finora, come gli attori in causa siano necessariamente il fornitore e il richiedente. Questo tipo di paradigma è il medesimo che si riscontra nella tipica interazione di tipo *client-server*. Attraverso la SOA questa interazione viene arricchita con un ulteriore attore detto *Service Directory o Service Broker* che, come mostrato in figura 1, si inserisce all'interno della comunicazione tra fornitore e fruitore del servizio. Di seguito, viene riportata una descrizione approfondita del ruolo di ognuno dei tre attori coinvolti nella SOA:

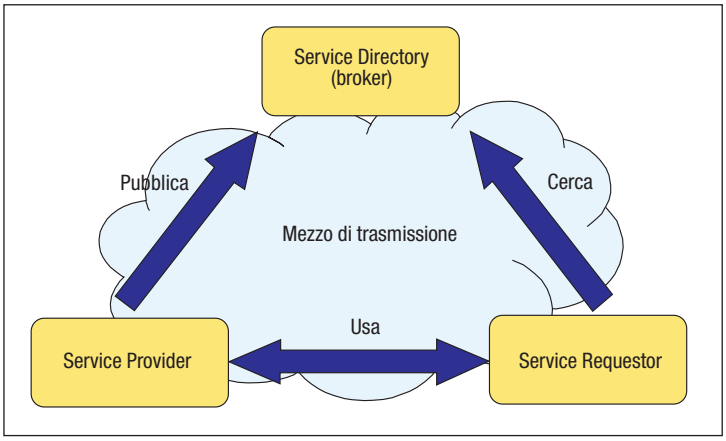


FIGURA 1
Service Oriented Architecture

Service Provider

Chi realizza e mette a disposizione un servizio. Tramite l'operazione di *publish* il servizio viene "pubblicizzato", in quanto le caratteristiche del servizio realizzato vengono memorizzate all'interno di un *registry* accessibile pubblicamente. Il Service Provider rimane, quindi, in attesa che un utente richieda tale servizio.

Service Directory o Service Broker

Questo componente si occupa della gestione del registry, permettendo, a chi ha necessità, di ricercare un servizio sulla base delle caratteristiche con le quali è stato definito e memorizzato. Naturalmente, il Service Directory può seguire politiche di controllo degli accessi sulle interrogazioni in modo da limitare la visibilità sui servizi inseriti. Nel presente lavoro il registry, salvo diversa indicazione, viene considerato totalmente accessibile.

Service Requestor

Rappresenta un potenziale utente che richiede un servizio. A tale scopo, tramite la primitiva di *find* l'utente interagisce con il Service Directory per ottenere il servizio più adatto ai propri obiettivi. Una volta individuato si collega al Service Provider corrispondente (*bind*) e inizia a fruire del particolare servizio (*use*).

La SOA definisce, quindi, "*chi fa che cosa*" all'interno di una serie di interazioni in cui il servizio ricopre il ruolo principale. Va notato che i tre attori interessati possono essere distribuiti sul territorio e possono utilizzare piattaforme tecnologiche differenti, con l'unico vincolo però di dover utilizzare tutti e tre un canale trasmissivo comune. Rimanendo sul mezzo trasmissivo, questo risulta essere un parametro dell'architettura, quindi l'approccio adottato dalle SOA ha il vantaggio di potersi integrare con diversi ambienti quali la telefonia mobile, il Web, o paradossalmente anche la posta ordinaria, permettendo in tal modo di realizzare applicazioni multi-canale, fruibili cioè attraverso diversi dispositivi.

Partendo da questa considerazione si può dire che una architettura per *e-Service* è un'istanza di una SOA dove il mezzo di comunicazione è di tipo elettronico, mentre una architettura per Web Service è un'istanza di una SOA dove il mezzo di comunicazione considerato è il Web. In figura 2 viene individuato il ruolo di SOAP, WSDL e UDDI, ovvero delle tre tecnologie fondamentali.

In particolare, SOAP è un protocollo, basato su XML (*eXtensible Markup Language*) e HTTP (*Hyper Text Transfer Protocol*), in grado di fare interagire componenti remoti attraverso il Web. Nonostante uno dei primi scopi di SOAP sia stato quello di supportare l'RPC (*Remote Procedure Call*) sul Web, questo protocollo è stato studiato anche per avere usi che possono comprendere una interazione di tipo asincrono e, quindi, basata su messaggi. In SOAP la specifica delle chiamate viene descritta in XML, mentre l'HTTP è il protocollo di trasporto su cui poggia. Questa ultima caratteristica pone SOAP in una posizione privilegiata rispetto ad altri meccanismi di invocazione presenti in standard di computazione distribuita quale COM+, Java RMI e CORBA, in quanto questi ultimi mostrano dei limiti nell'uso del Web durante la comunicazione visto che i loro messaggi vengono spesso bloccati dai *firewall*.

2. WSDL (WEB SERVICE DEFINITION LANGUAGE)

Si supponga che una banca voglia fornire un servizio di approvazione automatica prestiti di piccola entità che permette ai correntisti della banca di chiedere un prestito direttamente *on-line*. Dando per assunto che l'utente si sia precedentemente registrato, questi non dovrà far altro che indicare il proprio identificativo che corrisponderà al proprio numero di conto corrente e la somma di denaro richiesta.

Già da questa breve descrizione è possibile identificare come il servizio metta a disposizione dell'utente un'operazione (*approva*, per esempio) che richiederà dei dati in ingresso e in uscita. Attraverso WSDL è possibile formalizzare tutte queste caratteristi-

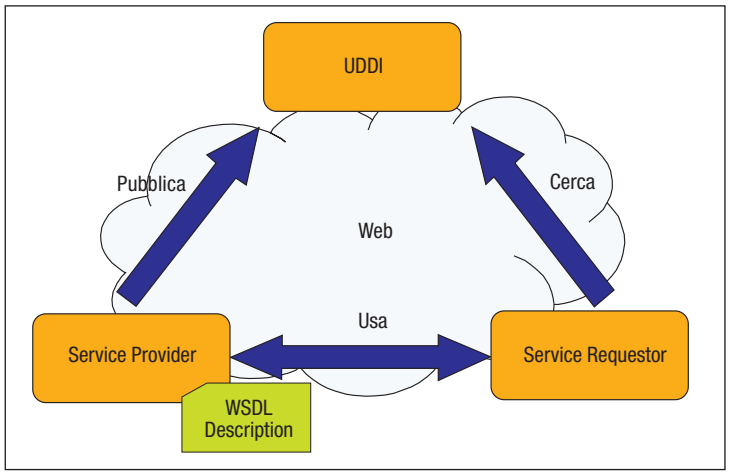


FIGURA 2
Web Service nella SOA

```
<definitions targetNameSpace="http://tempuri.org/services/approvaPrestiti"
             xmlns:tns="http://tempuri.org/services/approvaPrestiti"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    ...
  </types>
  <message name="RichiedenteMsg">
    <part name="cc" type="xsd:integer" />
    <part name="ammontareRichiesto" type="xsd:integer" />
  </message>
  <message name="erroreMsg">
    <part name="errorCode" type="xs:integer" />
  </message>
  <message name="approvazioneMsg">
    <part name="risposta" type="xs:string" />
  </message>
  <portType name="approvaPrestitoPT">
    <operation name="approva">
      <input message="tns:richiedenteMsg" />
      <output message="tns:approvazioneMsg" />
      <fault name="errore" message="tns:erroreMsg" />
    </operation>
  </portType>
  <binding name="approvaPrestitoSOAPBinding" type="tns:approvaPrestitoPT">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="approva">
      <soap:operation soapAction="http://tempuri.org/services/approva"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="approvaPrestitoService">
    <documentation>
        Servizio di approvazione prestiti automatico
    </documentation>
    <port name="approvaPrestito" binding="approvaPrestitoSOAPBinding">
      <soap:address location="http://tempuri.org/services/approva"/>
    </port>
  </service>
</definitions>
```

TABELLA 1
Documento WSDL per il servizio di approvazione prestito

che del servizio secondo uno schema che non differisce molto dalla specifica delle API (*Application Program Interface*) di un sistema. Come si può notare nella tabella 1, WSDL è un linguaggio basato su XML e permette di specificare quelle caratteristiche del servizio che prima sono state descritte solo a parole.

Si comincia con l'identificare le componenti fondamentali di un *file* WSDL partendo da *service* che identifica un insieme logico di servizi (un insieme di servizi e non solo uno!) ognuno dei quali è specificato da una *port*. È fondamentale ricordare che una port identifica l'indirizzo a cui il servizio risponde e il tipo di protocollo supportato dal servizio stesso e non le caratteristiche del servizio. Per questo motivo a ogni port è associata una *portType* il cui compito è quello di specificare la reale sintassi del servizio. La portType ha, quindi, il ruolo di dire cosa il servizio fa, mentre la port come il servizio possa essere acceduto. Questo tipo di specializzazione viene specificato attraverso il *binding* che data una portType descrive in che modo questa si inserisce all'interno di un particolare protocollo quale SOAP, HTTP o SMTP. Riferendosi, quindi, all'esempio, viene specificato un solo servizio (*ApprovaPrestito*) così come appare nel *tag port*. Questo servizio è accessibile all'indirizzo *http://tempuri.org/services/approva* e per invocarlo è necessario che il *client* inoltri le proprie richieste secondo il protocollo SOAP.
Nel dettaglio, una portType è composta da una serie di *operation* che rispecchiano le funzionalità fornite dal servizio e che interagiscono con l'utente e che possono basarsi su uno dei quattro pattern predefiniti:

- *One_way*: l'operazione è composta da un solo messaggio in ingresso al fornitore del servizio.
- *Request_response*: l'operazione prevede una risposta del fornitore del servizio successiva a un messaggio ricevuto dall'utente.
- *Solicit_Response*: l'operazione prevede l'attesa da parte del fornitore del servizio, di una risposta a fronte di una richiesta effettuata dal fornitore stesso.
- *Notification*: l'operazione è composta da un solo messaggio in uscita al fornitore del servizio.

Indipendentemente dal pattern, ogni interazione cliente-fornitore del servizio avviene attraverso uno scambio di messaggi opportunamente identificati in WSDL attraverso il *tag message*, grazie al quale è possibile specificare il formato del messaggio. In particolare, ogni messaggio è visto come un insieme

di una o più port ognuna delle quali aderente a un tipo di dato che può essere uno di quelli primitivi di XML (per esempio, *int* e *string*) oppure un tipo di dato definito in *types*; quest'ultimo non utilizzato all'interno dell'esempio visto che tutti i dati sono definiti secondo un tipo di base.
Al di là delle specifiche sintattiche di WSDL, soffermandosi ulteriormente sulla relazione che esiste tra port e portType è possibile dire che una port può essere vista come una specializzazione di una portType operata secondo un particolare protocollo di comunicazione. È quindi possibile che all'interno del medesimo file WSDL la stessa portType, quindi il servizio, possa essere reso accessibile, grazie ai *binding*, su diversi protocolli di trasporto. Al momento accanto alla specifica di WSDL sono stati specificati i binding su SOAP, HTTP e SMTP (*Simple Mail Transfer Protocol*) quindi è possibile solo specificare Web Service che comunicano attraverso questi protocolli. Sulla base di ciò un buon modo di operare prevede che una specifica WSDL di un servizio sia composta da almeno due file: il primo chiamato WSDL *Interface document* composto solo dalla specifica dei tipi, dei messaggi e delle portType, quindi per ogni tipo di binding verrà definito un documento apposito detto anche WSDL *implementation document*.
Sebbene risulti molto generico, WSDL è al tempo stesso molto esauriente e in grado di descrivere servizi di diverse tipologie. L'utilizzo dei quattro pattern di comunicazione permette, infatti, di specificare sia servizi asincroni che sincroni. Nel primo caso saranno utilizzati solo operazioni di tipo *one-way* e *notification*, mentre nel secondo caso di tipo *request-response* e *solicit-response*.
WSDL però propone solo una fotografia del servizio dandone quindi una visione statica. Si supponga, infatti, di aggiungere al WSDL l'operazione di *login*, come mostrato in tabella 2, in cui l'utente inserisce i propri *userID* e *password* che gli dà diritto di invocare l'operazione di richiesta prestito. In questo caso, solo una certa esperienza suggerisce che le operazioni previste devono essere invocate secondo un ordine preciso e a precise condizioni. L'utente, infatti, deve prima fare il login e solo dopo l'autenticazione richiedere un prestito. Ciò che manca, quindi, è la dinamica del servizio chiamato anche *comportamento*. Attraverso il comportamento è possibile sapere come un servizio funziona e quali sono le operazioni ammissibili in accordo con il suo stato interno. Per questo tipo di problema sono stati sviluppati diversi linguaggi. WSCL (*Web Service Conversation Language*) per esempio descrive, secondo una specifica XML, come un potenziale utente può conversare con il servizio descrivendolo come una macchina a stati finiti dove le operazioni rappresentano gli stati e, in aggiunta, vengono specificate le transizioni tra di essi. Un progetto più ambizioso, BPEL4WS (*Business Process Execution Language for Web Service*), definisce un linguaggio di composizione tra servizi che può anche essere utilizzato come specifica del comportamento di un singolo servizio.

```
...
<message name="loginReqMsg">
    <part name="userId" type="xsd:string" />
    <part name="password" type="xsd:string" />
</message>
<message name="loginResMsg">
    <part name="authRes" type="xsd:string" />
</message>
<portType name="approvaPrestitoPT">
    <operation name="login">
        <input message="tns:loginReqMsg" />
        <output message="tns:loginResMsg" />
        <fault name="errore" message="tns:erroreMsg" />
    </operation>
    <operation name="approva">
        ...
    </operation>
</portType>
<binding name="approvaPrestitoSOAPBinding"type="tns:approvaPrestitoPT">
    <soap:binding style="document"
                transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="login">
        <soap:operation soapAction="http://tempuri.org/services/login"/>
        <input>
            <soap:body use="literal"/>
        </input>
        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
    ...
</binding>
...
```

TABELLA 2
Operazioni di autenticazione per il servizio di approvazione prestito

3. BPEL4WS (BUSINESS PROCESS EXECUTION LANGUAGE FOR WEB SERVICES)

L'esigenza di coordinare in modo automatico attività svolte da attori diversi per il raggiungimento di un obiettivo comune quale, per esempio, l'approvazione di una richiesta, la concessione di un prestito e il rimborso di danni a un assicurato è caratteristica di processi con un elevato numero di istanze, regole di esecuzione precise e ripetibili. Le classiche applicazioni *software* per il supporto di flussi di lavoro con queste caratteristiche sono i sistemi di gestione di *workflow* (WfMS, *WorkFlow Management Systems* o), che consentono la gestione contemporanea di un numero, anche elevato, di istanze di processi seguendo schemi di processo predefiniti.

In questo ambito, il processo viene definito come una rete di attività e di relazioni esistenti tra di esse, criteri per indicare l'inizio e la fine di un processo, e informazioni riguardo alle singole attività, quali: i partecipanti, le applicazioni IT (*Information Technology*) associate, i dati, e così via. La rappresentazione di un processo in una forma che consente l'esecuzione automatica delle attività automatizzabili e la gestione automatica del passaggio tra un'attività e quelle che seguono nel flusso di lavoro è basata su alcuni costrutti fondamentali quali:

- *sequenza*: le attività vengono svolte l'una di seguito all'altra: una attività viene attivata solo al termine di un'attività precedente;
- *parallelo* (*AND split*): dopo la terminazione di una attività vengono attivate più attività in parallelo;
- *alternativa* (*OR split*): dopo la terminazione di una attività vengono attivate più attività in alternativa; vengono specificate le condizioni di attivazione delle attività oppure può essere effettuata una scelta non deterministica;
- *join* (*AND oppure OR*): per proseguire nel flusso di lavoro devono essere terminate tutte le attività precedenti (caso AND), o almeno una delle attività precedenti.

Sulla base di questi costrutti è possibile rappresentare reti di attività di tipo generale. Nei sistemi di gestione di workflow le attività vengono schedulate, attivate e gestite sulla base delle risorse disponibili e grazie a ciò è possibile ottenere l'automazione completa o parziale di un *business process* in cui documenti, informazioni o compiti sono passati da un partecipante a un altro per svolgere attività, secondo un insieme di regole procedurali. I prodotti sul mercato consentono, in genere, di coordinare processi all'interno di una sola organizzazione, seguendo una logica essenzialmente centralizzata, anche se le singole attività possono essere svolte dagli operatori su postazioni di lavoro diverse.

L'affermarsi dei Web Service rende naturale l'estensione dei concetti alla base dei sistemi di gestione di workflow anche per coordinare servizi in rete forniti da più organizzazioni. In questo ambito, l'obiettivo principale è quello di comporre più servizi forniti da fornitori diversi al fine di creare nuovi servizi a valore aggiunto. Tale composizione richiede però la definizione di standard per modellare le interazioni tra i servizi, e a tali standard stanno lavorando numerosi *vendor* e ricercatori.

Al momento, la letteratura propone due principali approcci al coordinamento dei servizi in rete, che vengono designati sotto le denominazioni di "orchestrazione" e "coreografia". Nell'orchestrazione, si fa riferimento a un processo di business che può interagire con altri servizi, interni o esterni. L'interazione avviene tramite messaggi scambiati tra i servizi. L'orchestrazione però presuppone il controllo sul processo da parte di una sola organizzazione e consente di gestire processi in esecuzione anche di lunga durata.

La coreografia indica, invece, un approccio più collaborativo, in cui ogni partecipante descrive il proprio ruolo nell'interazione e il compito del sistema di coreografia è principalmente quello di tenere traccia delle interazioni avvenute tra le parti.

BPEL4WS (o BPEL, in breve) è un linguaggio, nato grazie all'esperienza maturata sulla base di XLANG, una proposta di Microsoft, e WSFL (*Web Service Flow Language*), si colloca come la principale proposta in letteratura per l'orchestrazione di Web Service, che può essere vista come una na-

turale evoluzione dei modelli e sistemi per la gestione di workflow. La versione 1.1 di BPEL è stata progettata da Microsoft, IBM, Siebel Systems, BEA e SAP e rilasciata nel maggio 2003.
Essendo l'obiettivo di BPEL o quello di specificare un modello di comportamento dei servizi Web durante un processo di business, questo linguaggio si pone a un livello più alto rispetto ai linguaggi esaminati fin qui. Per questo BPEL presuppone che i servizi che dovrà orchestrare siano descritti secondo WSDL ed eventualmente memorizzati in un registry UDDI.
Dal punto di vista prettamente tecnico, BPEL esprime la logica di funzionamento del processo attraverso una grammatica basata su XML interpretabile da un *orchestration engine* opportunamente progettato per supportare questo linguaggio e il cui compito sarà quello di coordinare i vari servizi che compongono il processo.
Una delle particolarità di BPEL è la sua visione ricorsiva di composizione di Web Service. Infatti, Il processo definito come insieme di Web Service collaboranti, può essere esso stesso un Web Service. Questo nuovo servizio a valore aggiunto, così creato, sarà visibile all'esterno come un servizio semplice di cui è definita l'interfaccia WSDL. Ciò che avviene è, quindi, una divisione del processo secondo due punti di vista (Figura 3): quello dell'utente del processo, a sinistra, che può supporre di interagire con un singolo servizio, e quello dei servizi cooperanti all'interno del processo, a destra, che, descrivendo le proprie funzionalità attraverso WSDL, verranno invocati dall'*orchestration engine* nei tempi e nei modi definiti all'interno del processo.
Entrando più in dettaglio, il processo essenzialmente è composto da attività che possono gestire richieste da parte del client del processo (attività di *receive*) oppure inviare messaggi di risposta al client medesimo (attività di *reply*). Per poter soddisfare tale richiesta il processo si basa, come si è detto, su una serie di Web Service esterni chiamati *partner*, invocati attraverso l'attività di *invoke* che potrà corrispondere a un servizio sincrono o asincrono che dipende dalle caratteristiche del Web Service chiamato. Come è facile intuire, il paradigma sottostante basato sui messaggi ben si coniuga con i quattro modelli di interazione (*one-way, notification, request-response, solicit-response*) definiti in WSDL.

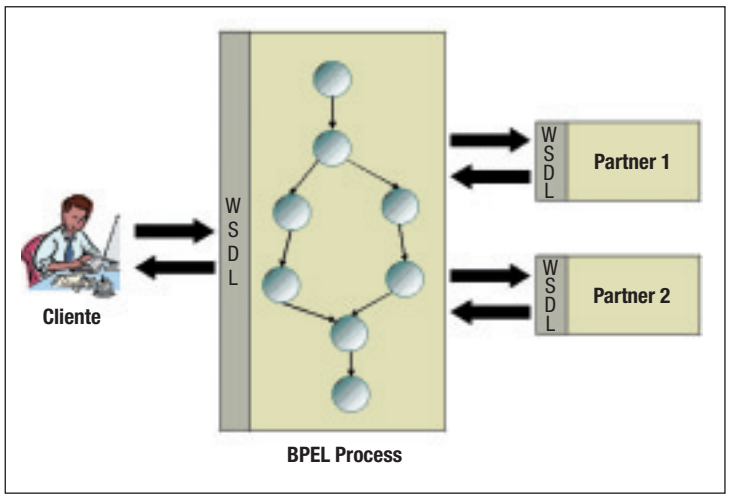


FIGURA 3
Modalità di interazione con un processo BPEL

I tre tipi attività descritti rappresentano le attività fondamentali che il processo deve svolgere per interagire con l'esterno. Tali attività dovranno essere svolte secondo uno schema ben preciso che rappresenta appunto il processo vero e proprio. A tal proposito, possono essere utilizzati i principali costrutti, precedentemente descritti, per i modelli di workflow e cui corrispondono i costrutti *sequence* per la sequenza, *flow* per l'esecuzione di attività in parallelo, *switch* e *pick* per le alternative e *while* come costrutto iterativo.
A supporto del mantenimento dello stato tra le diverse chiamate, il processo può, inoltre, contemplare la definizione di variabili che consentono di memorizzare i dati corrispondenti alle richieste ricevute o alle risposte di invocazioni di servizi. È anche possibile effettuare manipolazioni di tali variabili all'interno del processo, attraverso l'attività di *assign*, il che rende il linguaggio di descrizione del processo simile a un linguaggio di programmazione.
Partendo dall'esempio sul servizio di prestito illustrato per descrivere WSDL è possibile notare come questo può essere utilizzato all'interno di un processo BPEL. In prima battuta si consideri l'esempio di figura 4 dove si ipo-

tizza che una banca abbia definito un proprio processo per la gestione delle richieste di prestito [14].
Tale processo, definito in BPEL, delega l'approvazione del prestito a un servizio esterno che corrisponde all'*ApprovaPrestito* definito precedentemente in WSDL. In questo caso, il processo può essere visto come un semplice *mediator* che, ricevuta una richiesta la inoltra al servizio di competenza e, una volta ricevuta una risposta da quest'ultimo la inoltra al cliente. Il servizio visto in precedenza in poche parole potrebbe essere visto come un servizio che ora non è più invocabile direttamente ma che, supponendo sia fornito da un consulente finanziario, può essere utilizzato solo da istituti bancari.
Un esempio più complesso è mostrato in figura 5.
In questo caso, il processo risulta maggiormente ricco e si appoggia su due diversi servizi. Va notato che dal punto di vista del cliente, il processo precedente e quello attuale sono assolutamente identici. Ciò che cambia è come la banca gestisce le varie richieste di prestito. In questo caso infatti, sfruttando i costrutti tipici dei workflow, la banca è in grado di definire un processo BPEL in cui, dopo aver ricevuto la richiesta di prestito, seleziona uno dei due servizi di supporto sulla base dell'istanza di richiesta stessa. Nell'esempio, se l'ammontare del prestito richiesto è inferiore a 10.000 euro, la richiesta viene valutata da un "servizio di valutazione richiedenti", mentre il ricorso all'istituzione finanziaria avviene solo nei casi in cui il prestito sia di importo maggiore o il richiedente sia considerato a rischio. La risposta viene inviata, come nell'esempio precedente, al cliente, che in seguito potrà inviare la richiesta di attivazione del prestito. Quest'ultimo messaggio consente di esaminare una caratteristica di BPEL assente in generale nei sistemi di workflow: la seconda, *receive*, è anch'essa la ricezione di un messaggio, nel quale non viene effettuato un riferimento esplicito all'istanza attiva del processo. Il concetto di *correlationSet* di BPEL consente di definire una sorta di chiave per il processo, che permette di associare i messaggi alle istanze di processo attive. In questo esempio, si può fare l'ipotesi che venga gestita una richiesta di prestito alla volta da parte di uno stesso cliente e che, quindi, la chiave sia il nominativo del cliente. Pertanto, finché la richiesta di prestito precedente rimane attiva, ulteriori messaggi da parte dello stesso cliente verranno interpretati come continuazione dell'interazione nell'ambito del processo attivo. Per cui il cliente potrà inviare la richiesta di attivazione (o la rinuncia) senza avere la necessità di conoscere dettagli sulla modalità di gestione della sua pratica. Il meccanismo sopra descritto è particolarmente interessante poiché consente di gestire anche

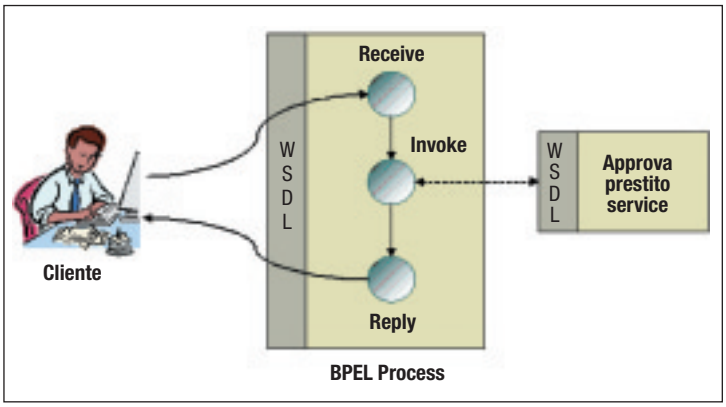


FIGURA 4
Esempio di interazione semplice con un processo BPEL

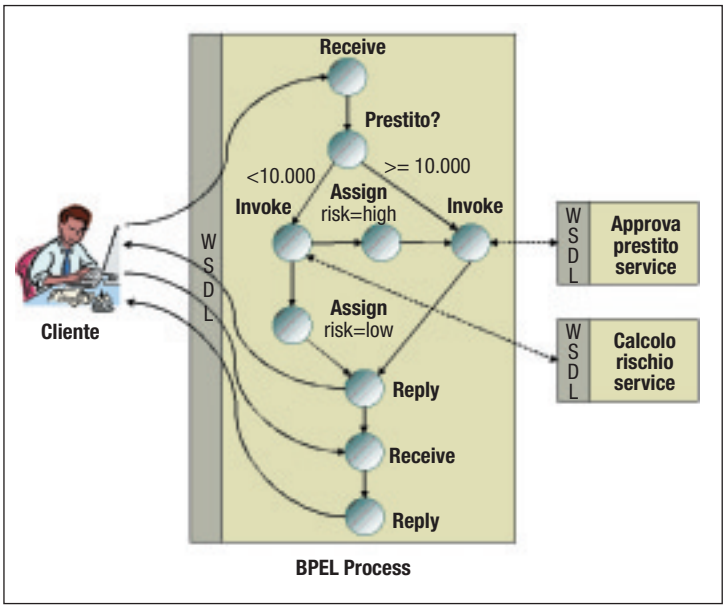


FIGURA 5
Processo BPEL con costrutti decisionali

processi e transazioni con lunga durata nel tempo.
Tra le funzionalità più avanzate, BPEL fornisce anche meccanismi per la gestione di transazioni ed eccezioni, basandosi sulle specifiche WS-coordination and WS-Transaction, sviluppate da IBM, Microsoft e BEA. Le eccezioni vengono gestite analogamente al linguaggio Java tramite costrutti di *throw* e *catch* identificando, quindi, anche degli *scope* all'interno dei quali la gestione delle eccezioni ha validità. Le eccezioni possono essere sollevate al verificarsi di particolari eventi, anche temporali. Poiché in un processo a lunga durata le eccezioni possono portare a interrompere transazioni in esecuzione, BPEL a supporto in particolare delle *long-transaction*, permette di specificare anche attività mirate alla compensazione e al *failure-recovery*.

4. UDDI (UNIVERSAL DESCRIPTION DISCOVERY AND INTEGRATION)

Dopo aver visto come un servizio può essere descritto secondo la visione chiamata statica e dinamica, va sottolineato come la probabilità di successo nella realizzazione di sistemi basati su Web Service dipende fortemente dalla facilità di reperimento dei servizi stessi. È di fondamentale importanza quindi, avere a disposizione una piattaforma in grado di reperire Web Service sulla base di diverse tipologie di ricerca.
UDDI è un progetto nato per questo scopo e iniziato da un gruppo di compagnie del mondo IT quali Microsoft, IBM e Ariba e al quale ora aderiscono circa 300 aziende.
Per meglio comprendere il ruolo di UDDI all'interno dell'ambiente tecnologico di interesse, occorre definire, in modo preciso, il dominio applicativo all'interno del quale si colloca. In prima analisi, questo scenario di riferimento può essere scomposto in tre fasi operative.
1. Un'azienda, o un gruppo di aziende, descrive le caratteristiche che una certa tipologia di servizio deve possedere. A questo livello ci si riferisce alla *tipologia di un servizio* come per esempio vendita, acquisto o noleggio e non a un servizio preciso quale può essere la vendita di software o l'acquisto di fiori.
2. Una qualunque azienda può, a questo punto, realizzare una delle tipologie di servizio definite, fornendo in tal modo un *servizio*.
3. L'insieme dei servizi, così come quello delle tipologie di servizio, deve essere consultabile. Questo permette, a chi vuole fruirne, di reperire il servizio desiderato, e agli sviluppatori di trovare le specifiche di una tipologia di servizio.
Partendo da questo scenario, UDDI affronta le problematiche di pubblicazione e reperimento di servizi, proponendo una architettura che permette di affrontare le tre problematiche riassunte nell'acronimo UDDI, e cioè:
- l'accesso alla descrizione di servizi, di tipologie di servizi e di fornitori di servizi secondo una struttura dati ben definita;
- l'astrazione dalla tecnologia utilizzata nella realizzazione del servizio. Questo al fine di permettere l'integrazione tra servizi realizzati in modo tecnologicamente differente;
- la ricerca di un servizio secondo differenti chiavi di ricerca.

UDDI, quindi, non è un contenitore di servizi o di descrizioni di servizi, bensì uno strumento che, utilizzando opportune strutture dati, tiene traccia della dislocazione dei servizi e delle loro descrizioni. Operando, inoltre, una classificazione sulle informazioni raccolte, UDDI permette l'esecuzione di ricerche efficaci ed efficienti.
Riguardo alla sua architettura UDDI prevede la creazione di un ambiente distribuito *peer-to-peer* in cui i vari *nodi*, che contengono una parte dei servizi disponibili, possano interoperare tra di loro allo scopo di soddisfare le richieste di pubblicazione e ricerca di servizi.
Per garantire l'interoperabilità, ogni nodo dell'infrastruttura, gestito da una figura che assume il ruolo di *operatore*, deve essere realizzato secondo le specifiche o rilasciate dal gruppo di lavoro UDDI. Tali specifiche definiscono sia la struttura delle informazioni che un registry UDDI deve memorizzare, sia il set minimo di API da implementare per l'accesso alle informazioni stesse.
Sulla base di queste specifiche ogni nodo può essere rappresentato secondo quanto schematizzato in figura 6, dove si possono

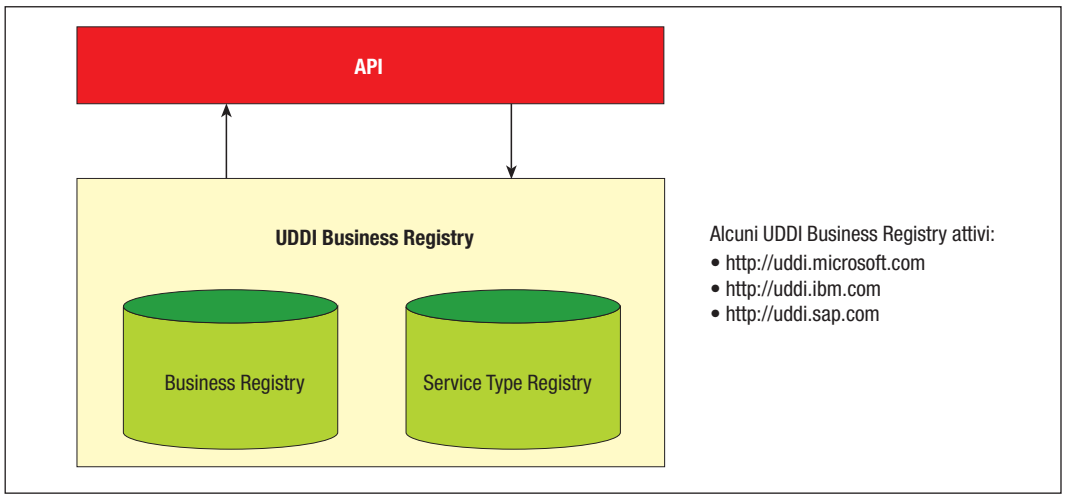


FIGURA 6
Struttura di un UDDI Registry

individuare le due componenti principali:

- *UDDI Business Registry* ulteriormente suddiviso in:
  - *Service Type Registry*: contenente un insieme di dati in grado di descrivere e localizzare le tipologie di servizio. Tali informazioni sono organizzate secondo una struttura detta *tModel*;
  - *Business Registry*: contenente informazioni sulle aziende che forniscono servizi e si sono registrate al nodo. Ogni entry di questo registry è detta *businessEntity*;
- libreria API per l'accesso, la ricerca e la manutenzione delle registrazioni.

Riprendendo, quindi, le tre fasi operative descritte nell'introduzione che caratterizzano il dominio applicativo di interesse, si può affermare che UDDI permette, grazie ai *tModel* memorizzati nel *Service Type Registry*, di ottenere le descrizioni di una tipologia di servizio, senza che esista alcun vincolo sul linguaggio di specifica utilizzato per redigere la descrizione stessa. Una volta realizzato un servizio, la sua descrizione può essere pubblicata su un nodo UDDI all'interno del *Business Registry*. Quindi UDDI, tramite il gruppo di API, mette a disposizione un meccanismo per la ricerca di servizi o tipologia di servizi secondo diverse chiavi di ricerca. In particolare, sono messe a disposizione tre tipologie di ricerca secondo una classificazione a *White page*, a *Yellow page* e a *Green page*. Nel primo caso, la metodologia di ricerca possiede una forte analogia con quella offerta dagli elenchi telefonici è possibile, quindi, conoscendo il nome del fornitore avere informazioni sui servizi forniti. Nel secondo caso, l'analogia si sposta verso le pagine gialle aziendali, la ricerca, quindi, avviene in prima istanza utilizzando una categorizzazione delle tipologie di servizio. Quello che invece è caratteristico di UDDI è il terzo tipo di classificazione che, basandosi sulle informazioni memorizzate nei tModel, permette di conoscere i fornitori di un servizio date le caratteristiche di alcune funzionalità. Inoltre, sfruttando le tassonomie e le classificazioni inserite nella *businessEntity*, è possibile organizzare l'elenco dei fornitori di servizio, in modo da permettere ricerche basate, per esempio, su l'ambito in cui si colloca il servizio, la localizzazione geografica dei fornitori oppure l'ambito in cui si colloca il fornitore del servizio.

In questa descrizione di UDDI è emerso come l'obiettivo dell'UDDI Registry sia duplice: rendere pubbliche le descrizioni sia di tipologie di servizio, sia di loro specializzazioni. Ciò si relaziona molto bene con la suddivisione della specifica WSDL esposta in precedenza in cui si suggeriva, per ogni servizio, di descriverlo secondo due documenti: il WSDL *Service Interface Document*, contenente solo i tag *type*, *message*, *portType* e *type* e il WSDL *Service Interface Document* che include il precedente e in più definisce i tag di binding e di service. Fatta questa suddivisione, alla luce delle caratteristiche di UDDI, è utile considerare il WSDL *Service Interface Document* come la descrizione della tipologia del servizio, mentre il WSDL *Service Implementation Document* come la descrizione del servizio vero e proprio.

Sempre in figura 6 sono elencati alcuni dei nodi UDDI al momento attivi. Data la natura ancora sperimentale, attualmente questi registri sono per la maggior parte popolati da servizi non funzionanti o non attivi. Questo sottolinea come i Web Service non siano ancora maturi per un utilizzo massiccio e gioca anche a sfavore di UDDI il fatto che da parte di alcuni venga considerato come un mezzo insufficiente e superato. Ciò è dovuto anche al fatto che purtroppo le funzionalità di UDDI, sebbene molto utili, sono considerate ancora troppo limitative. Un'effettiva ricerca di un servizio deve poter mettere a disposizione dell'utente sistemi di *querying* molto più efficaci del semplice *browsing* attraverso indici precostituiti. Ciò a cui si vuole arrivare è, quindi, una interrogazione di tipo *content-based* a cui la ricerca in questo momento sta lavorando.

5. DIREZIONI FUTURE

I linguaggi di specifica descritti nel presente lavoro sono ancora in una fase di continua evoluzione. Recente è la pubblicazione della proposta di standardizzazione del linguaggio WSDL nella sua versione 2.0, che prevede nuovi modelli di interazione con i Web Service, oltre a ridefinire alcuni costrutti precedentemente definiti. I motori di esecuzione di processi BPEL sono ancora allo stato di prototipi (per esempio, BPEL4J) anche se alcune proposte abbastanza solide, come Collaxa o, sono in commercio. In realtà, anche le specifiche di BPEL stesso sono in evoluzione, unitamente alle specifiche correlate.

Riguardo alla ricerca, particolare attenzione viene posta, attualmente, sui meccanismi di gestione dei servizi e sulle specifiche di caratteristiche dei servizi che consentano di rappresentare aspetti funzionali e non funzionali dei servizi stessi. In questo ambito, si possono prendere in esame alcune direzioni di sviluppo.

- Gestione dei servizi: è necessario ancora definire meccanismi di gestione e controllo per la pubblicazione e la validazione dei servizi che dovrebbero arricchire UDDI e renderlo uno strumento realmente efficace. Accanto a questo sono allo studio meccanismi per la definizione di politiche commerciali associate all'uso dei servizi in rete secondo strategie sviluppate appositamente per questo tipo di fornitura.
- Definizione delle caratteristiche riguardanti la qualità del servizio: tra le caratteristiche non funzionali dei servizi è particolarmente rilevante la definizione di livelli di qualità dei servizi e di meccanismi per il monitoraggio della qualità. Attualmente, sono state definite alcune proposte, quali WSLA o e WSOL [6, 10]. Per la definizione di caratteristiche di qualità dei servizi anche se in realtà sono necessarie ulteriori ricerche per la definizione di meccanismi di negoziazione e ottimizzazione dei parametri di qualità.
- Definizione di modalità di esecuzione dei servizi in ambienti distribuiti e mobili: sono allo studio, tra l'altro, possibili estensioni delle modalità computazionali utilizzate nel *grid computing* per utilizzare le risorse disponibili in rete per l'esecuzione di servizi distribuiti.
- Definizione di caratteristiche funzionali e non funzionali dei servizi per la ricerca in rete: le funzionalità di ricerca dei servizi disponibili in UDDI sono molto elementari. Proposte alternative sono state formulate nell'ambito del progetto Semantic Web del W3C (*World Wide Web Consortium*) per la caratterizzazione dei servizi basata su ontologie.

L'accesso a servizi in rete e la condivisione di risorse che è possibile effettuare tramite il *service oriented computing* è sicuramente un argomento di notevole interesse per lo sviluppo di nuove applicazioni. Tale interesse è stato dimostrato dalla partecipazione di tutti i principali produttori di software nella definizione di nuove specifiche e piattaforme di esecuzione. Di particolare attualità è l'utilizzo delle tecnologie sopra citate in realizzazioni nell'ambito di comunità chiuse, quali distretti virtuali o tra fornitori e clienti in una filiera produttiva. Le tecnologie basate su Web Service consentono un accesso più immediato alle informazioni e l'interazione automatizzata tra i sistemi informativi aziendali, superando le limitazioni dei sistemi proprietari. Pertanto, l'utilizzo delle tecnologie illustrate nel presente lavoro in ambito B2B è sicuramente di attualità e in forte crescita. Rimangono ancora

da esplorare le potenzialità dell'utilizzo di questi paradigmi nell'ambito di una interazione più globale.

Bibliografia

[1] Christensen E., Curbera F., Meredith G., Weerawarana S.: *Web Services Description Language (WSDL)* 1.1. http://www.w3.org/TR/wsdl.

[2] Collaxa: http://www.collaxa.com

[3] Communications of the ACM: *Special issue on Service Oriented Architectures*. Ottobre 2003.

[4] IEEE Computer: *Special issue on Web Services*. Ottobre 2003.

[5] Leymann F., Roller D., Thatte S.: *Goals of the BPEL4WS Specification*. xml.coverpages.org/BPEL4WS-DesignGoals.pdf

[6] Ludwig H., Kelle A., Dan A., King R. P., Franck R.: *Web Service Level Agreement (WSLA) Specification*. 2003.

[7] Mitra N.: *SOAP Version* 1.2 Part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/

[8] Peltz C.: Orchestration and choreography. *IEEE Computer*, ottobre 2003, p. 46-52.

[9] Thatte S. (Editor): *Business Process Execution Language for Web Services*. Version 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[10] Tosic V., Patel K., Pagurek B.: WSOL - Web Service Offerings Language. In: Workshop "Web Services, e-Business, and the Semantic Web (WES)". In conjunction with CAISE '02. (2002).

[11] UDDI : http://www.uddi.org/

[12] UDDI Group, UDDI Version 2.0 XML Schema, http://www.uddi.org/schema/uddi_v2.xsd

[13] WfMC, Workflow Management Coalition, http://www.wfmc.org

[14] Weerawarana S., Curbera F.: *Business Processes: Understanding BPEL4WS*. http://www.ibm.com

Barbara Pernici è professore ordinario di Sistemi per l'Elaborazione dell'Informazione al Politecnico di Milano. È laureata in Ingegneria Elettronica al Politecnico di Milano e ha un Master of Science in Computer Science della Stanford University. È attiva nella ricerca sulla progettazione di sistemi informativi, sui sistemi informativi mobili, sulla gestione di processi in rete e sulla qualità dei dati. Attualmente è responsabile scientifico del progetto FIRB MAIS (Multichannel Adaptive Information Systems).
barbara.pernici@polimi.it.

Pierluigi Plebani è laureato in Ingegneria Informatica presso il Politecnico di Milano dove attualmente sta seguendo il ciclo di studi per il dottorato in Ingegneria dell'Informazione.
Si occupa di tecnologie basate su Web Service per applicazioni inter-aziendali e applicazioni mobili, e di problematiche legate alla gestione delle politiche di sicurezza.
plebani@elet.polimi.it.