Sicurezza delle architetture orientate ai servizi

Sicurezza Delle Architetture Orientate Ai Servizi (Università degli Studi di Milano)

Sicurezza delle architetture orientate ai servizi

Ilaria Salbe

Appunti per il CdL Magistrale:
Sicurezza Informatica

Università degli Studi di Milano
Anno Accademico 2019/2020

Indice

1 Tecnologie di base

1.1 Introduzione

Le architetture multilivello derivano dall'aggiunta di un livello aggiuntivo per consentire ai client di accedere alle applicazioni tramite un Web Server. Il Web Server è un'applicazione software che, in esecuzione su un server, è in grado di gestire le richieste di trasferimento di pagine web di un client, tipicamente un web browser. L'aggiunta del livello Web ha portato alla nozione di "application server", piattaforme middleware che supportano l'accesso via Web. In informatica un application server (a volte abbreviato con la sigla AS) è una tipologia di server che fornisce l'infrastruttura e le funzionalità logiche di supporto, sviluppo ed esecuzione di applicazioni nonché altri componenti server in un contesto distribuito. Si tratta di un complesso di servizi orientati alla realizzazione di applicazioni ad architettura multilivello ed enterprise, spesso orientate per il web (applicazioni web). In altre parole su di esso gira la cosiddetta logica di business in un'architettura hardware/software di tipo multi-tier e può dunque essere definito un middleware. La sua gestione è opera dei cosiddetti sistemisti applicativi dove oltre alle operazioni di installazione e configurazione un'operazione tipica è quella di deployment dell'applicazione web.

Con il termine middleware si intende un insieme di programmi informatici che fungono da intermediari tra diverse applicazioni e componenti software. Sono spesso utilizzati come supporto per sistemi distribuiti complessi con architetture multitier.

Al giorno d'oggi i protocolli e processi middleware sono impiegati maggiormente per la creazione di sistemi distribuiti sulla rete, col compito principale di garantire la trasparenza dell'eterogeneità dei sistemi coinvolti. Alcuni servizi di rete standard sfruttano protocolli applicativi specifici come HTTP, FTP, SMTP ecc. ; protocolli di middleware come CORBA sono invece impiegati per la creazione di applicazioni distribuite non-standard o comunque più complesse per le quali non è sufficiente la comunicazione attraverso i normali protocolli. Indipendentemente dalla collocazione protocollare nello standard OSI, su può pensare ad un middleware Layer come tutto ciò che si interpone tra il sistema operativo (quindi software di basso livello che controlla direttamente l'hardware) e i software applicativi di alto livello, garantendo principalmente l'indipendenza tra questi due livelli. Un protocollo di Middleware fornisce un ben preciso modello di programmazione per la generazione di applicazioni distribuite, sollevando il programmatore dalla necessità di occuparsi di tutti i dettagli di basso livello della comunicazione e lo scambio di messaggi tra processi distribuiti. Le principali proprietà che il middleware deve fornire sono:

- Marshalling e Unmarshalling: i dati prodotti dalle applicazioni devono essere formattati per garantire l'indipendenza hardware-software dell'host di provenienza: il Marshalling ("impastamento"), effettuato in fase di trasmissione, prevede la suddivisione dei dati in più messaggi, la codifica in caratteri universali ASCII, la conversione little-endian big-endian ecc.; l'Unmarshalling, effettuato in fase di ricezione, opera esattamente al contrario riportando i dati in uno stato comprensibile a quel particolare host.

- Invocazioni Remote: i moderni middleware, con la nascita della programmazione a oggetti distribuiti, devono permettere l'invocazione remota di processi e metodi per garantire un'interazione efficiente tra le applicazioni.

- Totale trasparenza di utilizzo per l'utente finale: l'utente dell'applicazione programmata col supporto middleware non deve "accorgersi" della presenza dello stesso e quindi non deve occuparsi di coordinarne il funzionamento, in quanto sarà del tutto integrato nel codice dell'applicazione.

L'integrazione dei processi e dei servizi, residenti su sistemi con tecnologie e architetture diverse (Invocazioni Remote), è una funzione delle applicazioni middleware. Esso oggi identifica una serie di strumenti come DBMS, Web server, Application server, sistemi di gestione dei contenuti ed altri strumenti basati sul concetto di sviluppo e pubblicazione di applicazioni e contenuti. Gli sviluppi attuali si dirigono verso XML, SOAP, servizi Web e architetture orientate al servizio.

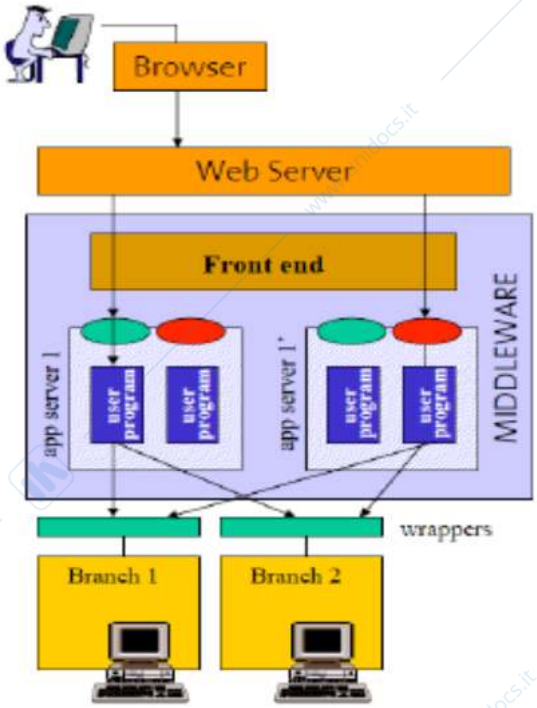

Figura 1: Multi-tiered structure of Web applications

L'introduzione di forme per permettere agli utenti di fornire informazioni ad un Web Server ha reso necessario modficare HTML (e HTTP), ma ha fornito un'interfaccia più avanzata rispetto al semplice recupero di file.

Proviamo a spiegare il significato del messaggio HTTP proposto in Figura 2.

Il messaggio di richiesta è composto di quattro parti:

- riga di richiesta (request line): la riga di richiesta è composta da metodo, URI e versione del protocollo. In questo caso è stato utilizzato il metodo POST, il quale è usato di norma per inviare informazioni al server (ad esempio i dati di un form). In questo caso l'URI indica che cosa si sta inviando e il body ne indica il contenuto.

- sezione header (informazioni aggiuntive):

  – Accept: pecifica l'ordine preferito dal browser relativamente ai MIME types supportati. La stringa "*/*" indica che il browser è in grado di gestire qualunque tipologia di MIME type.

  - User-Agent: indica il tipo di browser. Sia Internet Explorer che Netscape Navigator si identificano come "Mozilla", ma il primo include anche la stringa "MSIE".
  - Content-type: specifica il MIME type del documento di risposta.
  - Content-length: specifica la lunghezza del body della response, espresso in byte

- riga vuota (CRLF: i 2 caratteri carriage return e line feed);
- body (corpo del messaggio).

- POST /cgi-bin/post-query HTTP/1.0
- Accept: www/source
- Accept: text/html
- Accept: video/mpeg
- Accept: image/jpeg
- ...
- Accept: application/postscript
- User-Agent: Lynx/2.2 libwww/2.14
- From: grobe@www.cc.ukans.edu
- Content-type: application/x-www-form-urlencoded
- Content-length: 150
- * a blank line *
- &name = E
- &email= ernesto.damiani@unimi.it

Figura 2: HTTP parameter passing

1.1.1 CGI - Common Gateway Interface

Le prime implementazioni di applicazioni Web sono state costruite direttamente sui sistemi esistenti. In informatica Common Gateway Interface è una tecnologia standard usata dai web server per interfacciarsi con applicazioni esterne generando contenuti web dinamici. Lo script (o programma) CGI (Common Gateway Interface) ha funzionato come client in senso tradizionale (ad esempio utilizzando RPC - Remote Procedure Calling): L'utente ha fatto clic in un determinato URL e il server ha invocato lo script corrispondente. Lo script è stato eseguito, ha prodotto i risultati e li ha restituiti al server (di solito come l'indirizzo di una pagina Web). Il server ha recuperato la pagina e l'ha inviata al browser. CGI ha diversi problemi che non sono facili da risolvere:

- Gli script CGI sono processi separati, che richiedono ulteriori switch di contesto quando viene effettuata una chiamata (e quindi aumentando il ritardo complessivo)
- Fast-CGI consente di effettuare chiamate a un singolo processo in esecuzione, ma richiede comunque due cambi di contesto
- CGI è davvero un trucco rapido non progettato per prestazioni, sicurezza, scalabilità, ecc.

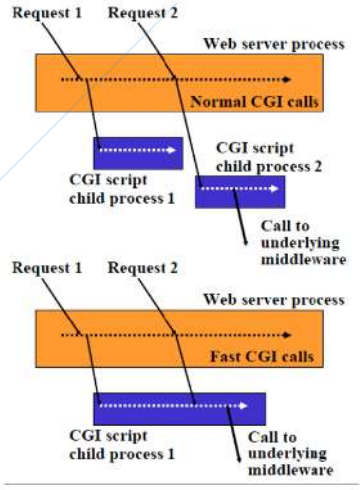

Figura 3: Why CGI don't scale

1.1.2 JVM - Java Virtual Machine

Un browser Web convenzionale non va oltre la visualizzazione dei dati: la potenza di elaborazione sul client non viene utilizzata. Aggiungendo una JVM (Java Virtual Machine) al browser, ora diventa possibile scaricare dinamicamente la funzionalità client (un'applet) ogni volta che è necessario. Il client diventa veramente indipendente dal sistema operativo ed è sempre sotto il controllo del server.

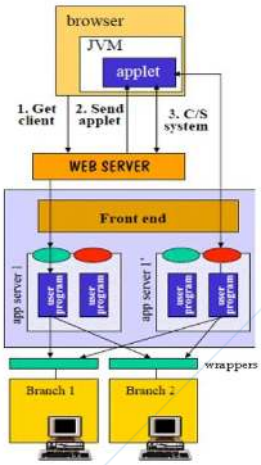

Figura 4: Introduzione di una Java Virtual Machine

1.1.3 Servlet

In informatica, nell'ambito della programmazione Web, i servlet sono oggetti scritti in linguaggio Java che operano all'interno di un server web (es. Tomcat, Jetty) oppure un server per applicazioni (es. WildFly, GlassFish) permettendo la creazione di applicazione web (elaborazione lato server).

Il nome deriva in contrapposizione alle Java applet, piccoli programmi scritti in linguaggio Java che si eseguono all'interno del browser dell'utente client (elaborazione lato client). L'uso più frequente delle servlet è la generazione di pagine web dinamiche a seconda dei parametri di richiesta inviati dal client browser dell'utente al server. Negli ultimi anni non viene eseguita la programmazione diretta delle servlet, ma si preferisce usare dei framework web che implementano la specifica servlet, oppure delle JavaServer Pages che vengono poi tradotte (compilate) in servlet a runtime. I servlet hanno lo stesso ruolo degli script CGI: forniscono un modo per invocare un programma in risposta a una richiesta HTTP, ma i servlet vengono eseguiti come thread del processo del server Java (non necessariamente il server Web) e non come processi OS separati.

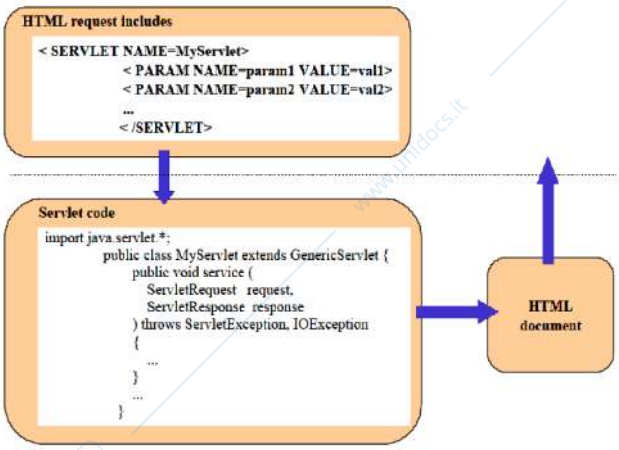

Figura 5: Lanciare una Servlet

1.1.4 Protocollo HTTP

Il protocollo HTTP è stato originariamente progettato come protocollo di scambio di documenti (richiedere un documento, ottenere il documento, renderlo / visualizzarlo). É quasi come l'e-mail (in realtà utilizza intestazioni e tipi MIME conformi a RFC 822). Il formato del documento (HTML) è stato progettato per far fronte ai problemi della GUI. L'interazione attraverso lo scambio di documenti può essere molto inefficiente quando le due parti dell'interazione sono programmi (i documenti devono essere creati, inviati, analizzati all'arrivo, ecc.). Il modello WWW iniziale era distorto verso il server: il client (il browser) non fa molto oltre alla visualizzazione del documento. Per applicazioni complesse, ciò significava molto più traffico tra client e server, con carichi elevati sul server all'aumentare del numero di utenti.

Business-to-Business o B2B indica le relazioni che un'impresa detiene con i propri fornitori per attività di approvvigionamento, di pianificazione e monitoraggio della produzione, o di sussidio nelle attività di sviluppo del prodotto, oppure le relazioni che l'impresa detiene con clienti professionali, cioè altre imprese, collocate in punti diversi della filiera produttiva. L'idea base dietro a B2B segue il modello client/server. Un servizio fornito da una compagnia può essere direttamente invocato da un client in esecuzione in un'altra società. Quali sono i problemi che derivano da questo modello?

- Lo sviluppo congiunto di client e server non è possibile

- É probabile che il server e il client siano nascosti dietro i firewall
- L'interazione avviene tra i sistemi esistenti, non è possibile ipotizzare che le piattaforme IT siano uniformi
- Internet è economico ma aperto a tutti (a differenza delle linee affittate che sono costose ma private). I sistemi / protocolli esistenti non sono realmente progettati per questo tipo di interazioni

1.1.5 Web Service

Le evoluzioni delle tecnologie Internet hanno due obiettivi principali:

- soddisfare i bisogni degli utenti
- integrare tra loro sistemi informativi tra loro eterogenei e sempre più complessi

I Web Service sono componenti software distribuiti ed accoppiati in modo lasco, che forniscono un servizio ben definito e sono accessibili da programmi mediante protocolli Internet standard. Servizio inteso non necessariamente come un servizio finale, ma come un componente indipendente che può essere usato per fornire un servizio finale.

Figura 6: Scenario di richiesta di un servizio

I predecessori utilizzati per sopperire a questi problemi sono le chiamate di procedura remota, ma sostituiti poi con i Web Service, un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su una medesima rete oppure in un contesto distribuito. Tale caratteristica si ottiene associando all'applicazione un'interfaccia software (descritta in un formato automaticamente elaborabile quale, ad es., il Web Services Description Language) che espone all'esterno il servizio/i associato/i e utilizzando la quale altri sistemi possono interagire con l'applicazione stessa attivando le operazioni descritte nell'interfaccia (servizi o richieste di procedure remote) tramite appositi "messaggi" di richiesta: tali messaggi di richiesta sono inclusi in una "busta" (la più famosa è SOAP), formattati secondo lo standard XML, incapsulati e trasportati tramite i protocolli del Web (solitamente HTTP), da cui appunto il nome Web service. Di fatto dunque il web service consiste in una chiamata ad un servizio molto simile a una funzione, subroutine o metodo scritta in maniera inusuale rispetto alla norma e con i suddetti metodi di chiamata, utili in termini di interoperabilità in un'architettura tipica complessa di tipo modulare.

Esistono 3 classi di servizi:

- I servizi di comunicazione e di trasporto (SOAP)
- I servizi di base: per l'indicizzazione e ricerca dei servizi (UDDI; WDSL)
- I servizi business: servizi specifici per certi settori d'attività (ebXML, Biztalk, ...)

L'architettura che sta alla base dei Web services è ben definita e sempre la stessa, indipendentemente dai Web services specifici dei vari fornitori utilizzati. Il modello dei Web Services, da un punto di vista orientato ai servizi, è basato su tre ruoli fondamentali e sulle necessarie interazioni tra ciascun ruolo. In figura 7 sono illustrati ruoli e interazioni. I tre ruoli sono:

- Service Provider – É una piattaforma che fornisce l'accesso al servizio. Per esporre un'applicazione come un servizio, il provider deve fornire un accesso basato su un protocollo standard e una descrizione standard del servizio (meta data). In parole povere, è l'entità che implementa il servizio e offre di eseguirlo per conto del richiedente (il server).
- Service requestor o user – In generale, è un'applicazione che invoca o avvia un'interazione con un servizio. Il service requestor e il service provider devono condividere le stesse modalità di accesso e interazione col servizio. Si tratta del potenziale utente di un servizio (il client).
- Service registry – É un registro presso il quale i service provider possono pubblicare il servizio e i service requestor possono trovare i servizi desiderati. Il service registry deve fornire una tassonomia consistente dei Web Services per facilitarne la scoperta, e una descrizione dettagliata dell'azienda che fornisce il servizio e del servizio stesso. Quindi, si tratta di un luogo in cui sono elencati i servizi disponibili e che consente ai fornitori di pubblicizzare i propri servizi e ai richiedenti di cercare e richiedere servizi

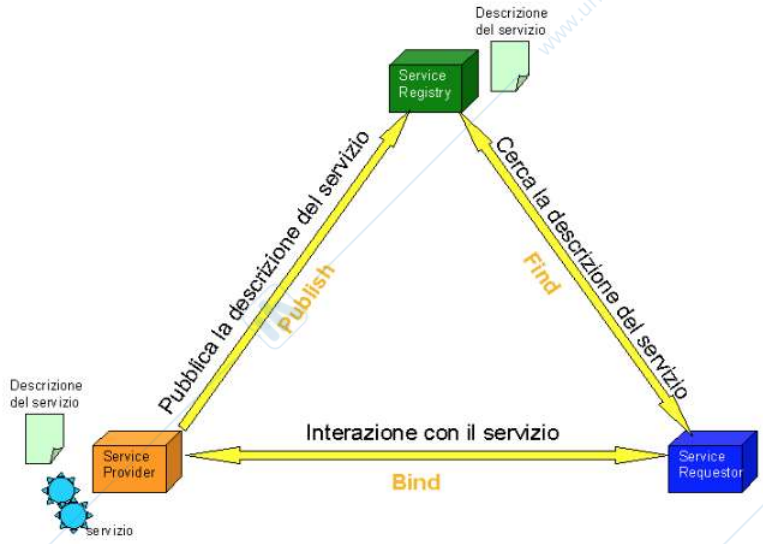

Figura 7: Ruoli e interazione nel modello concettuale dei Web Services

Il Web Service Provider sviluppa e definisce i Web Services e li pubblica poi all'interno dei Web Service Registries, oppure li rende direttamente disponibili ai Web Service Requester. Il Web Service Requester effettua un'operazione di ricerca per localizzare i servizi desiderati resi disponibili dai Web Service Provider e poi richiedono questi stessi servizi ai Web Service Registries oppure direttamente al luogo della loro pubblicazione. Una volta localizzato, il Web Service Requester si connette al suddetto servizio. I Web Service Registries si comportano come directory centralizzate, e sono depositari dei Web Services definiti e pubblicati dai Web Service Provider.

L'architettura del servizio Web proposta da IBM si basa su due concetti chiave:

- Architettura delle piattaforme middleware sincrone esistenti
- Specifiche attuali di SOAP (Simple Object Access Protocol), DDI (Universal Description and Discovery Protocol) e WSDL (Web Services Description Language).

L'idea fondamentale che sta alla base dei Web services è l'integrazione come servizio. Il concetto rappresenta un set definito di tecnologie basate sugli standard industriali, che lavorano insieme per facilitare l'interoperabilità tra sistemi eterogenei, che siano all'interno della stessa azienda, o risiedano da qualche parte su Internet. Con i Web services si possono abilitare applicazioni al Web, per farle comunicare con altre applicazioni che sono state a loro volta abilitate agli standard dei Web services. Nella sua essenza il concetto dei Web services è un diverso approccio alla distributed computing. I Web services effettivamente costituiscono un approccio che realizza completamente ciò a cui aspira la distributed computing: un modo flessibile ed economicamente vantaggioso di sfruttare tutte le capacità e le risorse della distributed computing, sia internamente sia esternamente e nella maniera più efficiente possibile – senza preoccupazioni per il tipo di applicazioni o sistemi operativi coinvolti.

Un'importante differenza con il middleware convenzionale è legata agli sforzi di standardizzazione del W3C (World Wibe Consortium) che garantiscono:

- Indipendenza dalla piattaforma (hardware, sistema operativo)
- Utilizzo dell'infrastruttura di rete esistente (HTTP)
- Neutralità del linguaggio di programmazione (.NET parla con Java)
- Portabilità attraverso strumenti middleware di diversi fornitori

I servizi Web sono componenti "debolmente accoppiati" che favoriscono il riutilizzo del software. Le tecnologie Web Service sono componibili e possono essere adottate in modo incrementale. Riassumendo i vantaggi relativi all'utilizzo di Web Service sono:

- permettono l'interoperabilità tra diverse applicazioni software su diverse piattaforme hardware;
- utilizzano standard e protocolli "open"; i protocolli ed il formato dei dati è, ove possibile, in formato testuale, cosa che li rende di più facile comprensione ed utilizzo da parte degli sviluppatori;
- mediante l'uso di HTTP per il trasporto dei messaggi, i Web service normalmente non necessitano di modifiche alle regole di sicurezza utilizzate come filtro sui firewall;

- possono essere facilmente utilizzati, in combinazione l'uno con l'altro (indipendentemente da chi li fornisce e da dove vengono resi disponibili) per formare servizi "integrati" e complessi;

- consentono il riutilizzo di infrastrutture ed applicazioni già sviluppate e sono (relativamente) indipendenti da eventuali modifiche delle stesse.

Transport	HTTP, IIOP, SMTP, IMS		
Messaging	XML, SOAP		WS-Addressing
Description	XML Schema, WSDL		WS-Policy, SSDL
Discovery	UDDI		WS-MetadataExchange
Choreography	WSCL	WSCI	WS-Coordination
Business Processes	WS-BPEL	BPML	WSCDL
Stateful Resources	WS-Resource Framework		
Transactions	WS-CAF	WS-Transactions WS-Business Activities	
Reliable Messaging	WS-Reliability		WS-ReliableMessaging
Security	WS-Security SAML, XACML		WS-Trust, WS-Privacy WS-SecureConversation
Event Notification	WS-Notification		WS-Eventing
Management	WSDM		WS-Management
Data Access	OGSA-DAI		SDO

Tabella 1: Standard per Web Service

Il fatto che l'industria si sta muovendo verso standard di programmazione (Java) e di scambio dei dati (XML) comuni, ha favorito l'ascesa dei Web Services. É assolutamente essenziale che gli standard nell'area dei Web Services siano ovunque gli stessi. Senza l'accordo dell'industria di tutto il mondo sugli standard della tecnologia sottostante ai Web Services, la promessa della neutralità delle piattaforme e della compatibilità universale di sistemi eterogenei non può semplicemente essere mantenuta.

Ulteriori standard per chiamate di procedure remote (SOAP) e chiamate di directory (UDDI) e di servizi di descrizione (WSDL) completano la gamma delle tecnologie dei Web Services.

EXtensible Markup Language, o XML, fornisce le fondamenta per i Web Services. É quella tecnologia dei Web services su sui si basano tutte le altre tecnologie. Similmente a HTML, XML usa tag di programmazione per definire la proprietà dei dati sulle pagine Web e altri documenti. Le tag XML, tuttavia, sono molto più versatili perchè possono definire non solo il modo in cui questi dati vengono visti, ma anche il contenuto stesso dei dati.

Simple Object Access Protocol, o semplicemente SOAP, gestisce tutte le comunicazioni tra i vari componenti. SOAP è una procedura remota per chiamare protocolli, che usa la sintassi XML per mandare i comandi che chiamano applicazioni da server remoti. Per il trasporto SOAP si basa su HTTP, di conseguenza i comandi SOAP attraversano i firewall delle aziende senza controllo, in quanto appaiono ai Web Server come pagine Web HTML standard.

Universal Discovery, Description and Integration (UDDI), mettono a disposizione directory services pubblicamente raggiungibili per Web services basati su XML. Quando

richiesto dalle applicazioni, i registri UDDI, XML-based, forniscono informazioni su servizi nuovi, sullo stato di servizi particolari e sulla disponibilità di servizi compatibili. UDDI conferisce ai diversi business una directory distribuita a livello mondiale, Web based, similmente alle pagine gialle, per pubblicare i propri servizi e per scoprire e integrare automaticamente tutti i servizi disponibili. I messaggi UDDI vengono trasmessi via SOAP. Come la rete attuale di server DNS (Domain Name System) ci aspettiamo di veder proliferare in futuro anche una rete di server UDDI.

UDDI usa il linguaggio basato su XML Web services Description Language, abbreviato WSDL, per descrivere le caratteristiche di un dato Web service, ed anche i protocolli e i formati che questo servizio utilizza.

Nell'ambito dell'informatica, con la locuzione inglese di Service-Oriented Architecture (SOA) si indica generalmente un'architettura software adatta a supportare l'uso di servizi Web per garantire l'interoperabilità tra diversi sistemi così da consentire l'utilizzo delle singole applicazioni come componenti del processo di business e soddisfare le richieste degli utenti in modo integrato e trasparente. SOA introduce alcune modifiche radicali al software:

- Indipendenza linguistica (ciò che conta è l'interfaccia)
- Interazione basata su eventi (non più modelli sincroni)
- Scambi basati su messaggi (no RPC)
- Composizione e orchestrazione

La SOA è possibile ma più difficile senza i servizi Web. Infatti, tra le tecnologie che permettono di creare delle architetture orientate ai servizi, ci sono:

- le reti di comunicazione, senza le quali sarebbe impossibile far comunicare applicativi che si trovano su server diversi;
- i Web service che permettono di definire le modalità di comunicazione dei vari applicativi;
- l'Enterprise Service Bus (ESB) che ha la funzionalità di coordinare, orchestrare i vari applicativi per svolgere le funzioni di business.

In informatica il binding è il processo tramite cui viene effettuato il collegamento fra una entità di un software ed il suo corrispettivo valore. Nei termini della programmazione a oggetti, la decisione circa l'attributo o il metodo da richiamare in un dato momento dell'esecuzione del programma viene effettuata grazie al binding. Tale decisione può essere stata stabilita in anticipo in maniera fissa, e in tal caso si parla di binding statico (o early binding); oppure può essere presa a tempo di esecuzione, in maniera dinamica, e in tal caso si parla di binding dinamico (o late binding).

Mediante WSDL può essere descritta l'interfaccia pubblica di un Web service (ovvero una descrizione basata su XML) che indica come interagire con un determinato servizio. Il Web Services Invocation Framework (WSIF) supporta una semplice API Java per richiamare i servizi Web, indipendentemente da come o dove vengono forniti i servizi. Il framework consente la massima flessibilità per l'invocazione di qualsiasi servizio descritto da WSDL (Web Services Description Language).

Riassumento, è possibile utilizzare WSIF per consentire al sistema di decidere cosa fare quando viene invocato un servizio:

- Se la chiamata è verso un bean locale (EJB), non fare nulla
- Se la chiamata è verso un bean remoto (EJB), utilizzare RMI (Remote Method Invocation), una tecnologia che consente a processi Java distribuiti di comunicare attraverso una rete.
- Se la chiamata è a una coda, utilizzare JMS (Java Message Service), l'insieme di API, appartenente a Java EE, che consente ad applicazioni Java presenti in una rete di scambiarsi messaggi tra loro.
- Se la chiamata è a un servizio Web remoto, utilizzare SOAP e XML

1.2 SOAP - Simple Object Access Protocol

I problemi base da risolvere con l'utilizzo di un Web Service sono:

- Come rendere l'invocazione del servizio parte della linguaggio in modo più o meno trasparente
- Come scambiare dati tra macchine che potrebbero usare rappresentazioni diverse per tipi di dati diversi. Ciò comporta due aspetti: formati dei tipi di dati (ad es. Ordini di byte in architetture diverse) e strutture di dati (devono essere appiattiti e quindi ricostruiti)
- Come trovare il servizio che si desidera effettivamente tra una raccolta potenzialmente ampia di servizi e server. Il client non deve necessariamente sapere dove risiede il server o anche quale server fornisce il servizio
- Come gestire gli errori nell'invocazione del servizio in un modo più o meno elegante, ad esempio:
  - server inattivo o occupato
  - comunicazione inattiva
  - richieste duplicate

CORBA (Common Object Request Broker Architecture) è uno standard sviluppato da OMG per permettere la comunicazione fra componenti indipendentemente dalla loro distribuzione sui diversi nodi della rete o dal linguaggio di programmazione con cui siano stati sviluppati. Esso facilita lo sviluppo di sistemi distribuiti fornendo:

- un'infrastruttura per permettere la comunicazione fra oggetti in un sistema distribuito,
- un insieme di servizi utili,
- un supporto che permette ad applicazioni, implementate usando vari linguaggi, di interoperare.

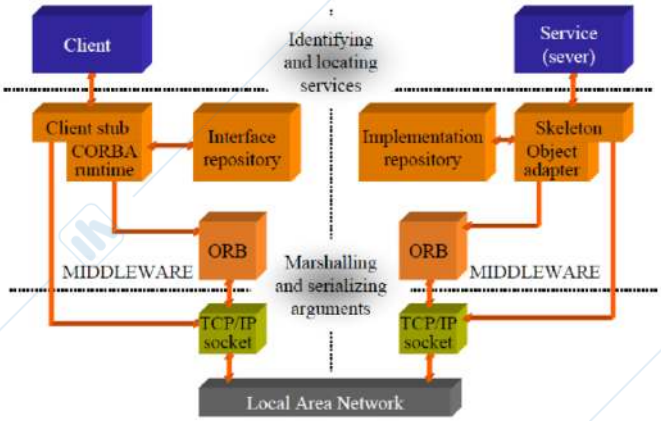

Figura 8: Invocazione CORBA

In informatica, il Component Object Model (COM) è un'interfaccia per componenti software, che permette la comunicazione tra processi e creazione dinamica di oggetti con qualsiasi linguaggio di programmazione che supporta questa tecnologia. Nei normali Sistemi Operativi, i processi che devono comunicare con altri processi non li possono chiamare direttamente ma devono usare alcune forme di comunicazione "inter-processo" del S.O. stesso. COM definisce l'interazione fra componenti e loro client fornendo una connessione senza componenti intermedi di sistema. Inoltre, fornisce una comunicazione trasparente tra processi diversi, intercettando la chiamata del client ed inoltrandola al componente del processo richiesto. COM è neutrale rispetto al linguaggio di programmazione, cioè si può usare qualunque linguaggio di programmazione per usare componenti COM a runtime. Questa neutralità è ottenuta specificando i formati dei dati e delle chiamate a funzioni con il linguaggio IDL, del tutto indipendente. Concettualmente quindi, è un passo in avanti ulteriore rispetto alle librerie a collegamento dinamiche (DLL). Ogni oggetto COM deve essere unico all'interno del sistema, e viene usato attraverso interfacce software anche loro globalmente uniche.

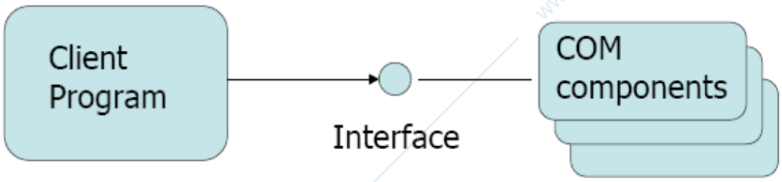

Figura 9: Modello COM

Il Distributed Component Object Model (DCOM) è una tecnologia informatica basata sul Component Object Model, e come questa, fa parte di quelle tecnologie che Microsoft stessa intende soppiantare con il framework Microsoft .NET. DCOM permette di effettuare chiamate di procedure remote attraverso una rete, occupandosi di tutte le mediazioni necessarie, in maniera indipendente dal linguaggio. La composizione delle classi e dei relativi metodi è esplicitata in un linguaggio di definizione d'interfaccia, IDL (Interface Description Language). In particolare, DCOM aggiunge queste importanti funzionalità al COM:

- Serializzazione (marshalling): codifica e decodifica in sequenze di byte dei parametri e dei valori di ritorno delle chiamate a metodo remote, per consentirne la trasmissione via rete.
- Garbage collection distribuita (distributed garbage collection): assicura il rilascio di riferimenti mantenuti dai client delle interfacce quando, per esempio, un processo client va in crash, o quando viene meno la connessione di rete.

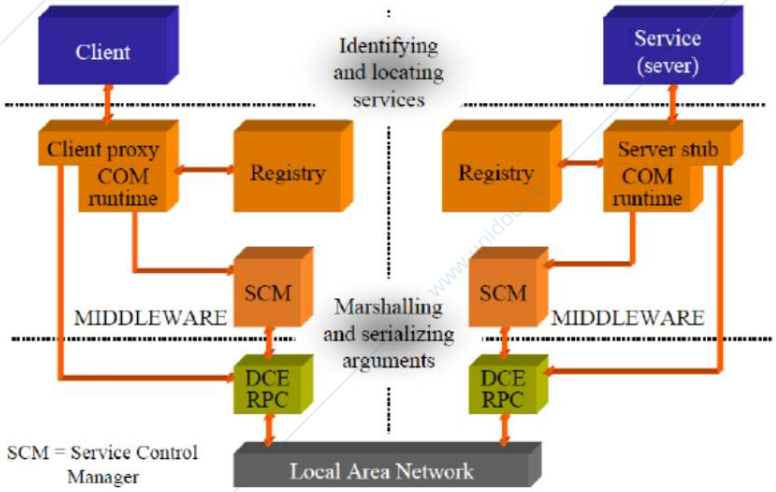

Figura 10: Invocazione DCOM

Il modello DCOM permette di realizzare applicazioni distribuite richiamando in modo trasparente i metodi di un oggetto COM che si trova su un computer diverso (senza sapere dove si trova l'oggetto). Per la comunicazione viene usato un meccanismo RPC, noto come ORPC (Object RPC), che può essere configurato per usare vari protocolli, come TCP e UDP. Gli oggetti DCOM supportano insiemi di funzioni collegate tra loro; ogni insieme di funzioni prende il nome di interfaccia; le interfacce sono implementate come DLL e quindi possono essere modificate senza dover ricompilare le applicazioni che le utilizzano. Quando una applicazione usa un oggetto DCOM riceve un puntatore alle funzioni di interfaccia che permette all'applicazione di ignorare dove si trova l'oggetto. Solo il sistema operativo ha bisogno di sapere dove si trova l'oggetto remoto per permettere agli oggetti locali di richiamarne i metodi. Il computer client deve essere configurato per DCOM in modo da registrare nel proprio registro di sistema la classe dell'oggetto remoto. Sul server si possono impostare i permessi necessari per eseguire l'applicazione DCOM in modo locale o remoto.

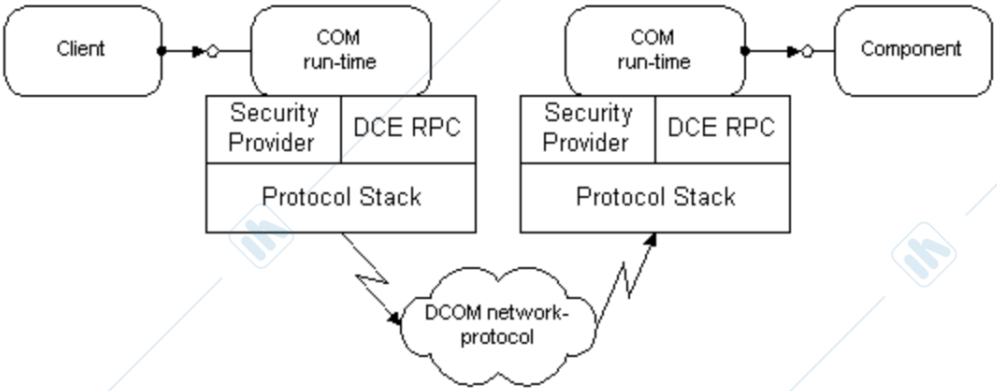

Figura 11: Modello DCOM su macchine differenti

Un fattore critico per un'applicazione distribuita è la sua capacità di crescere con il numero di utenti, la quantità di dati e la funzionalità richiesta. L'applicazione dovrebbe essere piccola e veloce quando le esigenze sono minime, ma dovrebbe essere in grado di gestire ulteriori richieste senza sacrificare le prestazioni o l'affidabilità. DCOM offre una serie di funzionalità che migliorano la scalabilità dell'applicazione:

- Nello stesso processo, le chiamate di funzioni sono veloci e dirette.

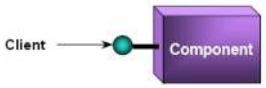

Figura 12: Scalabilità COM/DCOM

- Sulla stessa macchina, avvengono in maniera veloce e tramite IPC sicure. Con il termine ICP intendiamo la comunicazione tra processi.

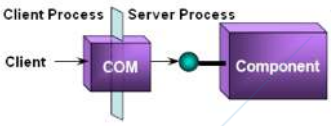

Figura 13: Scalabilità COM/DCOM

- Tra macchine diverse, le chiamate avvengono tramite DCE-RPC in maniera sicura, affidabile e flessibile. DCE-RPC è il sistema di chiamata di procedura remota che consente ai programmatori di scrivere software distribuito come se funzionassero tutti sullo stesso computer, senza doversi preoccupare del codice di rete sottostante.

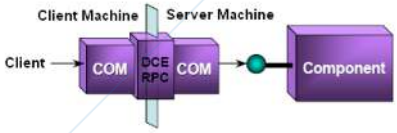

Figura 14: Scalabilità COM/DCOM

La Figura 15 rappresenta con quali protocolli vengono trasportate le informazioni nel modello DCOM.

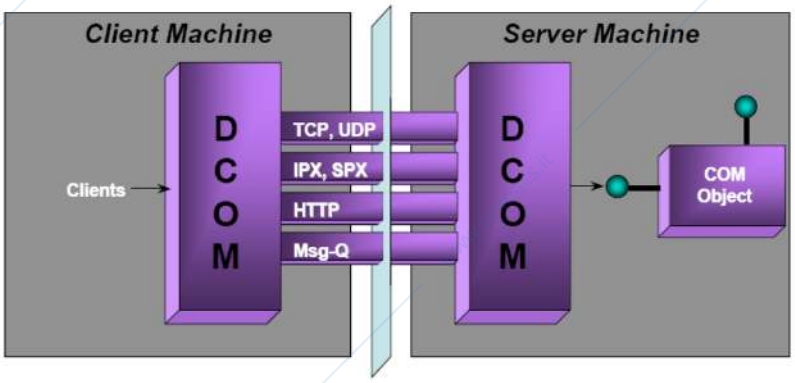

Figura 15: Trasporti COM/DCOM

La Figura 16 rappresenta le modalità tramite quali il modello DCOM garantisce la sicurezza.

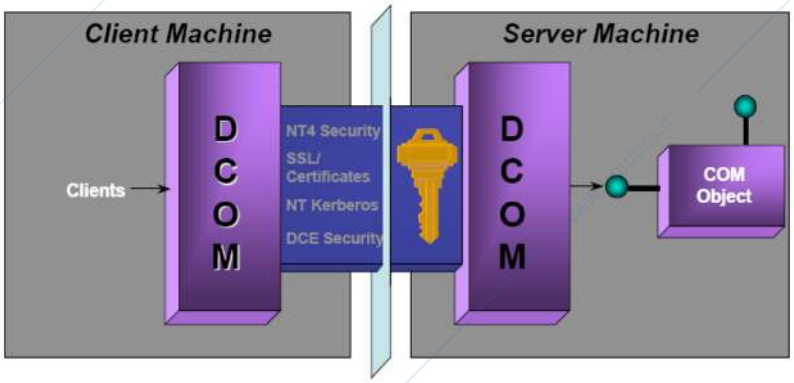

Figura 16: Sicurezza COM/DCOM

L'architettura DCOM è caratterizzata da:

- Multiplexing: Porta singola per protocollo, per processo del server, indipendentemente dal numero di oggetti
- Scalabile: protocolli senza connessione come UDP preferito
- Stabilite sessioni orientate alla connessione (TCP) riutilizzate dallo stesso client
- Larghezza di banda: la grandezza dei parametri passati in una chiamata di metodo influisce direttamente sul tempo impiegato a completare la chiamata. La soluzione

DCOM per minimizzare la larghezza di banda e la latenza è l'uso del protocollo di trasporto UDP (senza connessione).

I punti di forza dell'infrastruttura COM sono che l'attenzione si concentra sullo standard degli oggetti binari e sul riutilizzo di componenti scalabili/a grana fine. Si ha una concretezza e profondità della definizione, ad esempio sicurezza, gestione della vita, attivazione, installazione e implementazione. Inoltre si ha una estensibilità progettata. In CORBA/IIOP, invece, l'attenzione si concentra sul riutilizzo/integrazione tra nodi o rete: in pratica, è utile per soluzioni verticali, non per riutilizzo/integrazione orizzontale. Le specifiche sono incomplete: formato di marshalling di alcuni tipi di strutture di dati; e implicazioni della mancanza di servizi (ad es. Denominazione, eventi, gestione della vita). Non è prevista nessuna estensibilità progettata.

RPC, CORBA, DCOM, anche Java, utilizzano diversi meccanismi e protocolli per comunicare. Tutti mappano su TCP o UDP in un modo o nell'altro, ma usano una sintassi diversa per il marshalling, la serializzazione e il packaging dei messaggi. Il problema è che questi meccanismi sono un'eredità dal momento in cui le comunicazioni erano principalmente all'interno di LAN e all'interno di sistemi omogenei. La creazione di un ambiente B2B che combina i sistemi di diverse aziende diventa difficile perché i protocolli disponibili in RPC, CORBA o DCOM sono di livello troppo basso e non compatibili tra loro (sono necessari gateway, ecc.). Per risolvere questo problema, è stato utilizzato XML per definire SOAP. SOAP è la struttura operativa (framework) estensibile e decentralizzata che può operare sopra varie pile protocollari per reti di computer fornendo, tramite messaggi, richieste di procedure remote. I richiami di procedure remote possono essere infatti modellati come interazione di parecchi messaggi SOAP. SOAP dunque è uno dei protocolli che abilitano i servizi Web.

SOAP può operare su differenti protocolli di rete, ma HTTP è il più comunemente utilizzato e l'unico ad essere stato standardizzato dal W3C, su cui è incapsulato (embedded) il relativo messaggio. SOAP si basa sul metalinguaggio XML e la sua struttura segue la configurazione head-body, analogamente ad HTML. Il segmento opzionale "Header" contiene metadati come quelli che riguardano il routing, la sicurezza, le transazioni e parametri per l'orchestration. Il segmento obbligatorio "Body" trasporta il contenuto informativo e talora viene detto carico utile, o payload. Questo deve seguire uno schema definito dal linguaggio XML Schema.

Il protocollo definisce un set di regole che il client deve rispettare per richiedere la risposta al server che ospita ed espone il web service. Attraverso lo scambio di messaggi SOAP definiamo, quindi, delle RPC-Call (Remote Procedure Call) cioè delle chiamate di procedure remote: in pratica un componente software locale svolge un'operazione attraverso un'elaborazione compiuta, totalmente o in parte, attraverso l'ausilio di un sistema remoto (il web service). La trasmissione e la negoziazione di questi messaggi XML è regolata secondo i protocolli HTTP o SMTP, all'interno dei quali viene incapsulato il messaggio SOAP.

SOAP era originariamente concepito come la minima infrastruttura possibile necessaria per eseguire RPC attraverso Internet: uso dell'XML come rappresentazione intermedia tra i sistemi con struttura dei messaggi molto semplice e mappatura su HTTP per il tunneling attraverso i firewall e l'utilizzo dell'infrastruttura Web. L'idea era di evitare i problemi associati allo IIOP / GIOP di CORBA (che ricopriva un ruolo simile, ma che utilizzava una rappresentazione intermedia non standard e doveva comunque essere tunnelato tramite HTTP). L'obiettivo era di avere un'estensione che potesse essere facilmente collegata alle piattaforme middleware esistenti per consentire loro di interagire

attraverso Internet anziché attraverso una LAN come nel caso originale. Da qui l'enfasi su RPC sin dall'inizio (essenzialmente tutte le forme di middleware usano RPC a un livello o all'altro). Alla fine SOAP ha iniziato a essere presentato come un veicolo generico per gli scambi di messaggi guidati da computer attraverso Internet e quindi è stato aperto per supportare interazioni diverse da RPC e protocolli diversi da HTTP.

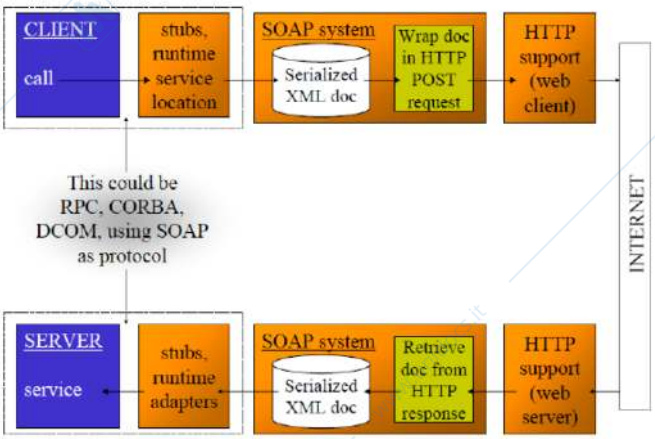

Figura 17: Invocazione SOAP

Il W3C ha iniziato a lavorare su SOAP nel 1999. Originariamente: Simple Object Access Protocol. SOAP copre le seguenti aree principali:

- Creazione del messaggio: un formato del messaggio per la comunicazione unidirezionale che descrive come un messaggio può essere compresso in un documento XML
- Modello di elaborazione: regole per l'elaborazione di un messaggio SOAP e una semplice classificazione delle entità coinvolte nell'elaborazione di un messaggio SOAP. Quali parti dei messaggi devono essere lette da chi e come reagire nel caso in cui il contenuto non sia compreso
- Modello di estensibilità: come il costrutto di base del messaggio può essere esteso con costrutti specifici dell'applicazione
- Framework di associazione del protocollo: consente il trasporto di messaggi SOAP mediante protocolli diversi (HTTP, SMTP, ...). Si tratta di convenzioni su come trasformare una chiamata RPC in un messaggio SOAP e come implementare lo stile di interazione RPC.

SOAP è "un protocollo leggero destinato allo scambio di informazioni strutturate [...]", "un paradigma di scambio di messaggi unidirezionale senza stato". SOAP efinisce il formato generale di un messaggio e come elaborarlo. RPC è implementato in cima alle specifiche di base in seguito alle convenzioni della "rappresentazione SOAP RPC".

Un messaggio SOAP può passare attraverso più hop sulla strada dal mittente iniziale al destinatario finale. Le entità coinvolte nel trasporto del messaggio sono chiamate nodi SOAP. Gli intermediari SOAP inoltrano il messaggio e possono manipolarlo. Ogni nodo SOAP assume un certo ruolo che influenza l'elaborazione dei messaggi sul nodo.

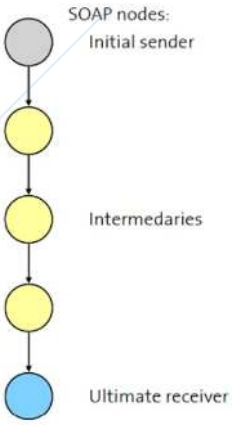

Figura 18: Percorso messaggio SOAP

1.2.1 Struttura messaggi SOAP

Riassumendo, SOAP (Simple Object Access Protocol) è un protocollo basato su XML per lo scambio d'informazioni in un ambiente distribuito e definisce un formato comune per trasmettere dati tra client e service. SOAP prevede l'imbustamento dei contenuti applicativi da scambiare all'interno di un formato di busta XML ed è indipendente dal protocollo di trasporto utilizzato per la consegna dei messaggi, che teoricamente potrebbe anche avvenire off-line. L'elemento base di un messaggio SOAP è il SOAP Envelope (Figura 19).

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
        ...

</soap:Envelope>
```

Figura 19: SOAP Envelope

L'Envelope ha due figli: l'Header, opzionale, e il Body, obbligatorio. Il linguaggio SOAP non definisce la semantica di questi due elementi, ma solo la struttura del messaggio. L'elemento opzionale SOAP Header (Figura 20) estende il messaggio e contiene metadati, informazioni utili al processamento del messaggio, come ad esempio l'identità dell'utente, informazioni riguardo la cifratura del documento, informazioni per il routing del messaggio, informazioni sulla sessione.

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope">

    <soap:Header>
        <myHeader soap:actor="..." soap:mustUnderstand="...">
            ...
        </myHeader>
    </soap:Header>

    <soap:Body ...>
        ...
    </soap:Body>
</soap:Envelope>
```

Figura 20: SOAP Header (optional)

Il SOAP Body (Figura 21), obbligatorio, infine contiene i dati veri e propri del messaggio, chiamato Payload.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
    <soap:Body xmlns="http://www.bank.org/ns">
        <pagamento>
            <da>cliente</da>
            <a>shop</a>
            <importo>100</importo>
        </pagamento>
    </soap:Body>
</soap:Envelope>
```

Figura 21: SOAP Body

Il livello di messaggio è indipendente dal livello di trasporto, quindi la busta SOAP può essere impacchettata per essere inviata via HTTP, SMTP, etc., ma la struttura e il contenuto della busta SOAP rimarrà immutato.

L'header è intesa come un segnaposto generico per informazioni che non dipendono necessariamente dall'applicazione (l'applicazione potrebbe anche non essere consapevole del fatto che al messaggio è stata allegata un'header). Gli usi tipici dell'header sono: informazioni di coordinamento, identificatori (ad es. Per transazioni), informazioni di sicurezza (ad es. Certificati). SOAP fornisce meccanismi per specificare chi dovrebbe occuparsi delle intestazioni e cosa farne. A tal fine include:

- MustUnderstand: Talvolta alcuni header devono essere processati affinchè l'applicazione possa procedere oltre. Si consideri ad esempio l'header contenente le informazioni di cifratura del messaggio. Se non viene processato (e quindi il messaggio rimane cifrato) il processamento non può andare avanti. Per indicare che un header DEVE essere analizzato o il processamento interrotto si usa l'attibuto MustUnderstand settandolo a "1" (di default è "0").

- Actor: Un messaggio SOAP può viaggiare dal mittente al destinatario attraversando differenti endpont lungo il percorso. Non tutti gli header del messaggio sono necessariamente indirizzati al destinatario finale, ma anche a nodi intermedi. L'attributo opzionale actor serve appunto a specificare l'endpoint al quale è indirizzato l'elemento.

- La versione 1.2 SOAP definisce un altro attributo opzionale per i blocchi di intestazione, relay di tipo booleano, che indica se un blocco di intestazione indirizzato a un intermediario SOAP deve essere inoltrato se non viene elaborato.

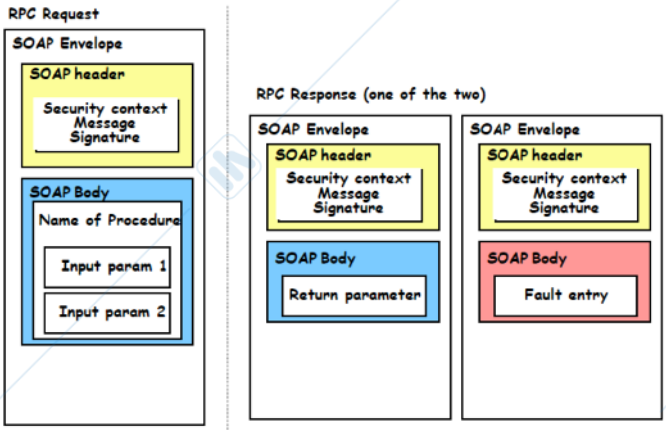

Figura 22: SOAP Security Headers

Il corpo SOAP è l'elemento obbligatorio nell'ambiente SOAP: Envelope, il che implica che è qui che devono essere trasportate le principali informazioni end-to-end trasmesse in un messaggio SOAP. Il body è destinato ai dati specifici dell'applicazione contenuti nel messaggio. Un elemento body equivale a un blocco di intestazione con attributi actor = ultimateReceiver e mustUnderstand = 1. A differenza dei blocchi di header, SOAP specifica il contenuto di alcuni elementi del corpo: ad esempio, fornisce una mappatura di RPC su un elemento del corpo SOAP (convenzioni RPC). La voce Errore (per la segnalazione di errori nell'elaborazione di un messaggio SOAP).

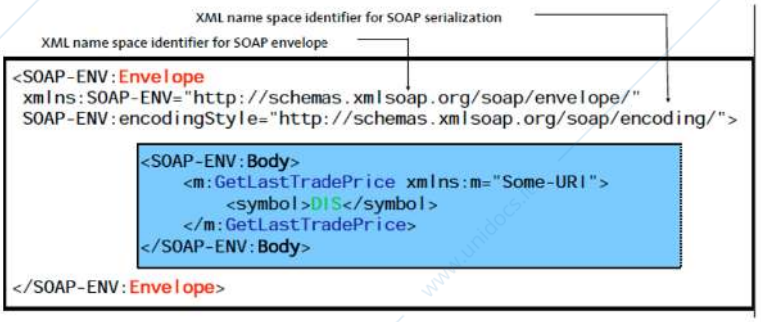

Figura 23: SOAP Body

Nella Figura 24 è rappresentato un esempio di messaggio SOAP completo.

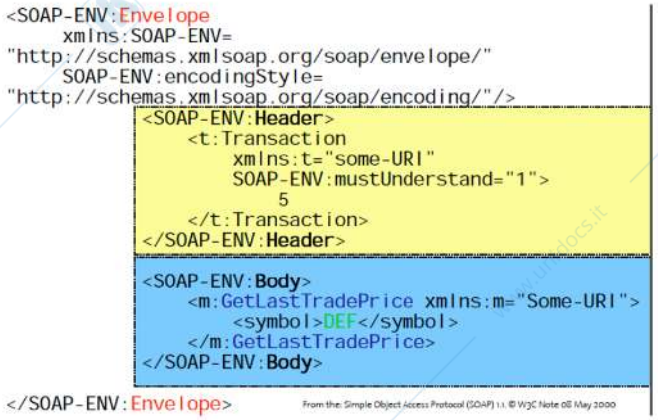

Figura 24: SOAP Message

Quando non è stato possibile elaborare un messaggio SOAP, viene restituito un errore SOAP. Un errore deve contenere le seguenti informazioni:

- Codice errore (Fault Code): indica la classe di errore e possibilmente un sottocodice (per informazioni specifiche sull'applicazione). I codici di errore includono:
  - Mancata corrispondenza della versione (Version Mismatch): spazio dei nomi non valido nell'envelope SOAP
  - Must Understand: un elemento di intestazione con "Must Understand" impostato su "true" non è stato compreso
  - Client: messaggio errato (formato o contenuto)

  - Server: problema con il server, impossibile elaborare il messaggio.

- Stringa di errore (Fault String): spiegazione dell'errore leggibile dall'uomo (non destinata all'elaborazione automatizzata)
- Fault Actor: chi ha causato l'errore
- Dettaglio (Detail): dati specifici dell'applicazione relativi all'errore

Gli errori nella comprensione di un blocco di intestazione obbligatorio vengono risolti, utilizzando un elemento di errore, ma includono anche un'header speciale che indica quale dei blocchi di intestazione originale non è stato compreso.

Per ogni messaggio ricevuto, ogni nodo SOAP sul percorso del messaggio deve elaborare il messaggio come segue:

1. Decidere in quali ruoli recitare (ruoli standard: next o ultimateReceiver o altri ruoli definiti dall'applicazione). Questi ruoli possono anche dipendere dal contenuto del messaggio.
2. Identificare i blocchi di intestazione obbligatori destinati al nodo (ruolo corrispondente, mustUnderstand = true). Se un nodo obbligatorio non è compreso dal nodo, deve essere generato un errore. Il messaggio non deve essere ulteriormente elaborato.
3. Elaborare i blocchi di intestazione obbligatori e, nel caso del destinatario finale, il corpo. Altri blocchi di intestazione destinati al nodo potrebbero essere elaborati. L'ordine di elaborazione non è significativo.
4. Gli intermediari SOAP inoltreranno, infine, il messaggio.

I blocchi di intestazione elaborati possono essere rimossi a seconda delle specifiche del blocco. I blocchi di intestazione indirizzati all'intermediario ma non elaborati vengono inoltrati solo se l'attributo di inoltro è impostato su true. Gli intermediari SOAP attivi possono anche modificare un messaggio in altri modi (ad es. Crittografare il messaggio).

SOAP specifica una rappresentazione uniforme per richieste e risposte RPC indipendente dalla piattaforma. Non definisce i mapping ai linguaggi di programmazione. SOAP RPC non supporta funzionalità RPC/RMI avanzate come riferimenti a oggetti o garbage collection distribuita. Questo può essere aggiunto da applicazioni o standard aggiuntivi (vedi WSRF) Formalmente, RPC non fa parte delle specifiche SOAP di base. Il suo utilizzo è facoltativo.

Per richiamare un RPC SOAP, sono necessarie le seguenti informazioni:

- L'indirizzo del nodo SOAP di destinazione.
- Il nome della procedura o del metodo.
- Identità e valori di tutti gli argomenti da passare alla procedura o al metodo insieme a eventuali parametri di output e valore restituito.
- Una chiara separazione degli argomenti utilizzati per identificare la risorsa Web che è la destinazione effettiva per l'RPC, in contrasto con quelli che trasmettono dati o controllano le informazioni utilizzate per elaborare la chiamata dalla risorsa di destinazione.

- Il modello di scambio di messaggi che verrà utilizzato per trasmettere l'RPC, insieme a un'identificazione del cosiddetto "Metodo Web" (su cui più avanti) verrà utilizzato.

- Facoltativamente, i dati che possono essere trasportati come parte dei blocchi di intestazione SOAP.

Tali informazioni possono essere espresse con una varietà di mezzi, tra cui IDL (Interface Definition Definition). Si noti che SOAP non fornisce alcun IDL, formale o informale. Si noti, inoltre, che le informazioni di cui sopra differiscono in modo sottile dalle informazioni generalmente necessarie per invocare altri RPC non SOAP.

Request:

```
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>DIS</symbol>
</m:GetLastTradePrice>
</SOAP-ENV:Body>
```

Response:

```
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
<Price>34.5</Price>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
```

Figura 25: RPC SOAP

I messaggi SOAP possono essere trasferiti utilizzando qualsiasi protocollo. Un'associazione (binding) di SOAP a un protocollo di trasporto è una descrizione di come deve essere inviato un messaggio SOAP usando quel protocollo di trasporto. Binding specifica come sono correlati i messaggi di risposta e richiesta. Il framework di associazione SOAP esprime le linee guida per specificare un'associazione a un protocollo specifico. I messaggi SOAP vengono in genere trasferiti tramite HTTP. L'associazione a HTTP è definita nella specifica SOAP. HTTP ha un modello di connessione noto e un modello di scambio di messaggi. Il client identifica il server tramite un URI, si connette ad esso utilizzando la rete TCP / IP sottostante, invia un messaggio di richiesta HTTP e riceve un messaggio di risposta HTTP sulla stessa connessione TCP. HTTP correla implicitamente il suo messaggio di richiesta con il suo messaggio di risposta; pertanto, un'applicazione che utilizza questa associazione può scegliere di inferire una correlazione tra un messaggio SOAP inviato nel corpo di un messaggio di richiesta HTTP e un messaggio SOAP restituito nella risposta HTTP. Allo stesso modo, HTTP identifica l'endpoint del server tramite un URI, l'URI di richiesta, che può anche fungere da identificazione di un nodo SOAP sul server. HTTP consente più intermediari tra il client iniziale e il server di origine identificato dall'URI di richiesta, nel qual caso il modello di richiesta / risposta è una serie di tali coppie. Si noti, tuttavia, che gli intermediari HTTP sono distinti dagli intermediari SOAP. Il binding HTTP utilizza la funzione Metodo web SOAP per consentire alle applicazioni di scegliere il cosiddetto metodo Web - limitandolo a uno di GET o POST - da utilizzare sullo scambio di messaggi HTTP. Inoltre, utilizza due modelli di scambio di messaggi che offrono alle applicazioni due modi per scambiare messaggi SOAP tramite HTTP: SOAP può utilizzare GET o POST. Con GET, la richiesta non è un messaggio SOAP ma la

risposta è un messaggio SOAP, con POST sia la richiesta che la risposta sono messaggi SOAP (nella versione 1.2, la versione 1.1 considera principalmente l'uso del POST).

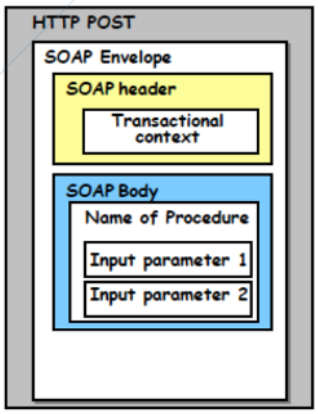

Figura 26: SOAP HTTP Binding

La Figura 27 presenta una vista globale sul modello di scambio di messaggi SOAP, mediante l'utilizzo del protocollo HTTP.

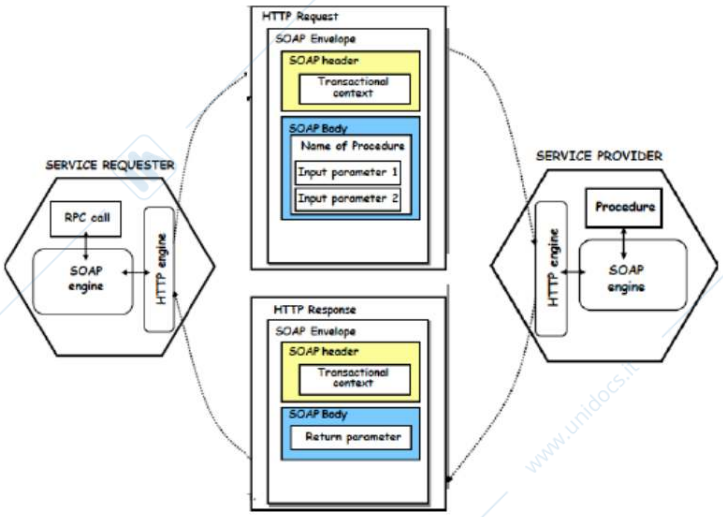

Figura 27: SOAP HTTP Binding

Il Web Services Invocation Framework (WSIF) supporta una semplice API Java per richiamare i servizi Web, indipendentemente da come o dove vengono forniti i servizi. Il framework consente la massima flessibilità per l'invocazione di qualsiasi servizio descritto da WSDL (Web Services Description Language). Utilizzando WSIF, WSDL può diventare il fulcro di un framework di integrazione per l'accesso a software in esecuzione su piattaforme diverse e l'utilizzo di protocolli molto diversi. L'unica condizione preliminare è che il software debba essere descritto usando WSDL e che abbia incluso nella sua descrizione un'associazione per la quale il framework WSIF del cliente ha un provider. WSIF definisce e viene fornito con i provider per i protocolli Java, Enterprise JavaBeans (EJB), Java Message Service (JMS) e Java EE Connector Architecture (JCA) locali, il che significa che un client può definire un EJB o un servizio di messaggi Java accessibile servizio direttamente come un'associazione WSDL e accedervi in modo trasparente

tramite WSIF, utilizzando la stessa API utilizzata per un servizio SOAP o una classe Java locale. Utilizzare WSIF per consentire al sistema di decidere cosa fare quando viene invocato il servizio:

- Se la chiamata è verso un EJB (bean) locale, non fare nulla;
- Se la chiamata è verso un bean remoto, utilizzare RMI (Remote Method Invocation);
- Se la chiamata è a una coda, utilizzare JMS;
- Se la chiamata è a un servizio Web remoto, utilizzare SOAP e XML.

Se c'è una descrizione dell'interfaccia singola, il sistema decide sull'associazione. Questo tipo di funzionalità è al centro della nozione di Service Oriented Architecture.

Quindi, WSIF è un API Java progettata per consentire la costruzione di client di servizi Web in un modo che non siano programmaticamente dipendente da un binding specifico. Un esempio è illustrato nella Figura ref fig: WSIF: un'applicazione Java che chiama un servizio con collegamenti SOAP WSDL è in grado di utilizzare la stessa interfaccia client di quando si chiama un servizio con collegamenti EJB.

```
<env:Body>
  <p:itinerary
   xmlns:p="http://.../reservation/travel">
   <p:departure>
    <p:departing>New York</p:departing>
    <p:arriving>Los Angeles</p:arriving>
    <p:depDate>2001-12-14</p:depDate>
    <p:depTime>late afternoon</p:depTime>
<p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
    <p:departing>Los Angeles</p:departing>
    <p:arriving>New York</p:arriving>
    <p:depDate>2001-12-20</p:depDate>
    <p:depTime>mid-morning</p:depTime>
    <p:seatPreference/>
   </p:return>
  </p:itinerary>
</env:Body>
```

Figura 28: WSIF

SOAP si basa su XML e si basa su XML per rappresentare i tipi di dati. L'idea originale in SOAP era di rendere espliciti tutti i dati scambiati sotto forma di un documento XML molto simile a ciò che accade con gli IDL nelle piattaforme middleware convenzionali.

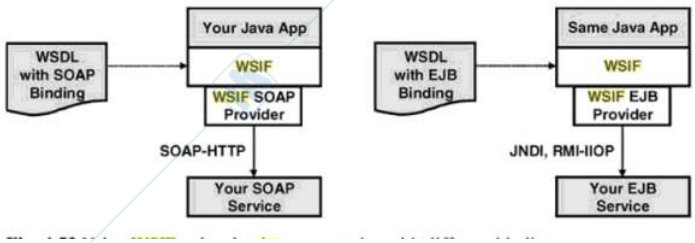

Figura 29: SOAP attachments

L'approccio in Figura 29 riflette l'assunto implicito che ciò che viene scambiato è simile ai parametri di input e output delle invocazioni del programma. É molto difficile utilizzare SOAP per lo scambio di tipi di dati complessi che non possono essere facilmente

tradotti in XML (e non c'è motivo di farlo): immagini, file binari, documenti, formati di rappresentazione proprietari, messaggi SOAP incorporati, ecc. C'è una soluzione: "messaggio SOAP con nota sugli allegati" proposta nel 2002 che ha affrontato questo problema. Utilizza i tipi MIME (come le e-mail) e si basa sull'inclusione del messaggio SOAP in un elemento MIME che contiene sia il messaggio SOAP che l'allegato. La soluzione è semplice e segue lo stesso approccio adottato nei messaggi di posta elettronica: include un riferimento e ha l'allegato effettivo alla fine del messaggio. Il documento MIME può essere incorporato in una richiesta HTTP allo stesso modo del messaggio SOAP. La Figura 30 mostra un esempio di messaggio SOAP con allegati.

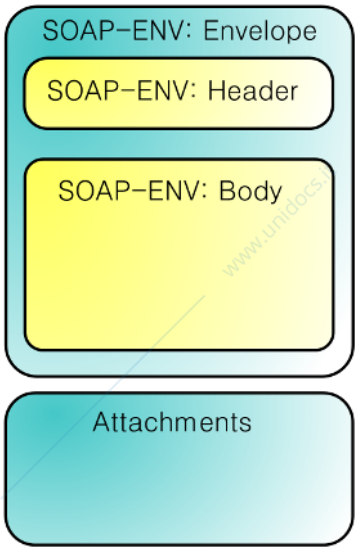

Figura 30: SOAP attachments

Il problema che nasce dall'utilizzo di questa tecnica è che la gestione del messaggio implica il trascinamento dell'allegato, che può avere conseguenze sulle prestazioni per i messaggi di grandi dimensioni. La scalabilità può essere seriamente compromessa quando l'allegato viene inviato in una volta sola (nessuno streaming). Inoltre, non tutte le implementazioni SOAP supportano gli allegati. I motori SOAP devono essere estesi per gestire i tipi MIME (non troppo complessi ma aggiungono costi generali). Le proposte alternative includono DIME di Microsoft (Direct Internet Message Encapsulation) e allegati WS.

Gli allegati sono relativamente facili da includere in un messaggio e tutte le proposte (basate su MIME o DIME) sono simili nello spirito. Le differenze sono nel modo in cui i dati vengono trasmessi dal mittente al destinatario e in che modo queste differenze influiscono sull'efficienza. MIME è ottimizzato per il mittente, ma il destinatario non ha idea di quanto sia grande il messaggio che riceve poiché MIME non include la lunghezza del messaggio per le parti che contiene. Ciò può creare problemi con i buffer e l'allocazione di memoria. Obbliga anche il destinatario a analizzare l'intero messaggio alla ricerca dei confini MIME tra le diverse parti (DIME specifica esplicitamente la lunghezza di ciascuna parte che può essere utilizzata per saltare ciò che non è rilevante).

Tutti questi problemi possono essere risolti con MIME in quanto fornisce meccanismi per l'aggiunta di lunghezze delle parti e potrebbe essere presumibilmente esteso per supportare alcune forme base di streaming. Tecnicamente, queste non sono questioni molto rilevanti e hanno più a che fare con il marketing e il controllo degli standard. Il vero

impatto degli allegati risiede nelle specifiche dell'interfaccia dei servizi Web che vedremo più avanti (come modellare gli allegati in WSDL?).

La stretta relazione tra SOAP, RPC e HTTP ha due ragioni principali: inanzitutto, SOAP è stato inizialmente progettato per il tipo di interazione client-server che viene generalmente implementato come RPC o sue varianti. Inoltre, RPC, SOAP e HTTP seguono modelli di interazione molto simili che possono essere facilmente mappati l'uno nell'altro (e questo è ciò che SOAP ha fatto).

I vantaggi di SOAP derivano dalla sua capacità di fornire un veicolo universale per trasmettere informazioni attraverso piattaforme e applicazioni di middleware eterogenee.A questo proposito, SOAP svolgerà un ruolo cruciale negli sforzi di integrazione delle applicazioni aziendali in futuro in quanto fornisce lo standard che è mancato in tutti questi anni. Alcuni dei primi sistemi che incorporano SOAP come metodo di accesso sono stati i database. Il processo è estremamente semplice: una procedura memorizzata è essenzialmente un'interfaccia RPC, dove:

- Servizio Web = procedura memorizzata
- IDL per stored procedure = tradotto in WSDL
- Chiamata per il servizio Web = utilizza il motore SOAP per mappare la chiamata a procedura memorizzata

Questo uso dimostra come SOAP si adatta alle architetture e alle interfacce middleware convenzionali. É solo un'estensione naturale per loro.

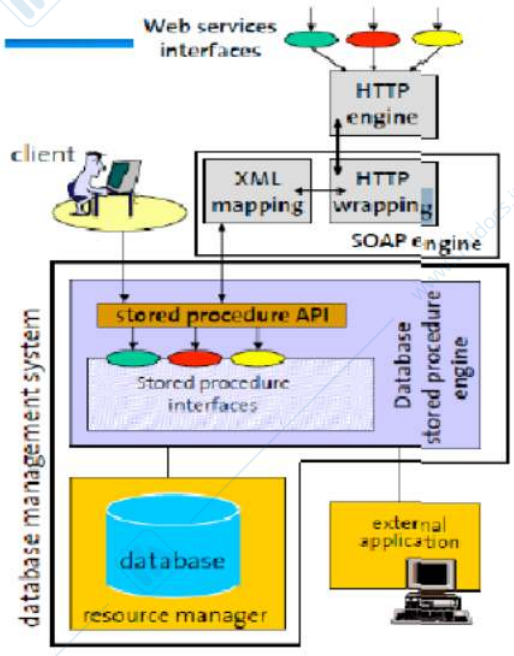

Figura 31: SOAP e Database

Inoltre, la caratteristica di neutralità di SOAP lo rende esplicitamente adatto all'uso con qualsiasi protocollo di trasporto. Le implementazioni spesso utilizzano HTTP come protocollo di trasporto, ma è possibile utilizzare altri protocolli di trasporto popolari. Ad esempio, SOAP può essere utilizzato anche su SMTP, JMS e code di messaggi. SOAP,

quando combinato con gli scambi post/get HTTP, esegue facilmente il tunneling attraverso i firewall e i proxy esistenti e, di conseguenza, non richiede la modifica delle diffuse infrastrutture di elaborazione e comunicazione esistenti per l'elaborazione degli scambi post/get HTTP. SOAP ha a disposizione tutte le funzionalità di XML, compresa una facile internazionalizzazione ed estensibilità con spazi dei nomi XML. Le limitazioni di SOAP derivano dalla sua aderenza al modello client-server: scambi di dati come parametri nelle invocazioni di metodi, cioè schemi di interazione rigidi che sono altamente sincroni. Infatti, quando si fa affidamento su HTTP come protocollo di trasporto e non si utilizza l'indirizzamento di servizi Web o un bus di servizio aziendale, i ruoli delle parti interagenti sono fissi. Solo una parte (il client) può utilizzare i servizi dell'altra parte. Un'altra limitazione riguarda la sua semplicità: infatti, SOAP non è sufficiente in un'applicazione reale, mancano molti aspetti. La verbosità del protocollo, la lenta velocità di analisi dell'XML e la mancanza di un modello di interazione standardizzato hanno portato al dominio sul campo da parte dei servizi che utilizzano il protocollo HTTP più direttamente, ad esempio, REST.

Riassumendo, SOAP, nella sua forma attuale, fornisce un meccanismo di base per incapsulare i messaggi in un documento XML:

- mappando il documento XML con il messaggio SOAP in una richiesta HTTP
- trasformando delle chiamate RPC in messaggi SOAP
- utilizzando semplici regole su come elaborare un messaggio SOAP (le regole sono diventate più precise e complete nella versione 1.2 della specifica).

SOAP è un protocollo molto semplice destinato al trasferimento di dati da una piattaforma middleware a un'altra. Nonostante le sue pretese di essere aperto (che sono vere), le specifiche e le implementazioni attuali sono molto legate a RPC e HTTP. SOAP sfrutta la standardizzazione dell'XML per risolvere i problemi di rappresentazione e serializzazione dei dati (utilizza lo schema XML per rappresentare i dati e le strutture dei dati e si affida anche all'XML per serializzare i dati per la trasmissione). Man mano che XML diventa più potente e compaiono ulteriori standard attorno a XML, SOAP può sfruttarli semplicemente indicando quale schema e codifica viene utilizzato come parte del messaggio SOAP. Lo schema e la codifica attuali sono generici, ma presto ci saranno standard verticali che implementano schemi e codifiche su misura per una particolare area di applicazione (ad esempio, gli sforzi intorno all'EDI).

1.3 WSDL

1.3.1 Introduzione

Un Interface Description Language o linguaggio di definizione dell'interfaccia (IDL), è un linguaggio di specifica utilizzata per descrivere un componente software application programming interface (API). IDL descrivono un'interfaccia in un modo indipendente dalla lingua, consentendo la comunicazione tra i componenti software che non condividono una lingua. Ad esempio, tra quelle scritte in C ++ e quelle scritte in Java. Gli IDL sono comunemente utilizzati in chiamata di procedura remota software. In questi casi le macchine alle due estremità del collegamento possono fare uso di diversi sistemi operativi e linguaggi di programmazione. IDL offre un ponte tra i due sistemi differenti.

Gli IDL sono in genere utilizzati in approcci di comunicazione impliciti, in cui le primitive di comunicazione sono nascoste dietro le chiamate di procedure o metodi e, quindi, comunicare implica conoscere l'interfaccia della procedura/metodo dall'altra parte. IDL è un linguaggio per dichiarare le interfacce (usato per pubblicare interfacce e dal compilatore e dal linker per decidere cosa fare quando si verifica una chiamata a una procedura/metodo remoti). É una mappatura tra rappresentazioni di dati nella lingua a una rappresentazione intermedia (per il marshalling). Rappresenta, inoltre, un modo per serializzare i dati per la trasmissione. Sono dei compilatori per trasformare le definizioni dell'interfaccia in moduli che possono essere collegati e compilati in altri moduli software. La maggior parte degli IDL sono basati sui caratteri (i dati sono rappresentati in ASCII).

Per la comunicazione implicita, la rappresentazione intermedia è fortemente legata al modo in cui le interfacce sono definite per i moduli software a livello di linguaggio di programmazione. L'IDL consente di definire ciascun servizio in termini di nomi e parametri di input e output (oltre forse ad altri aspetti rilevanti).

Tutti i sistemi RPC hanno un linguaggio che consente di descrivere i servizi in modo astratto (indipendentemente dal linguaggio di programmazione utilizzato). Questa lingua ha il nome generico di IDL (ad esempio, l'IDL di SUN RPC è chiamato XDR). Un compilatore di interfaccia viene quindi utilizzato per generare gli stub per client e server (rpcgen in SUN RPC). Lo stub o anche metodo stub, è una porzione di codice utilizzata in sostituzione di altre funzionalità software in quanto può simulare il comportamento di codice esistente (come una routine su un sistema remoto) o l'interfaccia COM, e temporaneo sostituto di codice ancora da sviluppare. Sono pertanto utili durante il porting di software, l'elaborazione distribuita e in generale durante lo sviluppo di software e il software testing. Rpcgen genera le intestazioni delle procedure che il programmatore può quindi utilizzare per compilare i dettagli dell'implementazione.

Data una specifica IDL, il compilatore di interfaccia esegue una varietà di attività:

- Genera la procedura stub client per ciascuna firma della procedura nell'interfaccia. Lo stub verrà quindi compilato e collegato al codice client
- Genera uno stub del server. Può anche creare un server principale, con lo stub e il dispatcher compilati e collegati al suo interno. Questo codice può quindi essere esteso dal progettista scrivendo l'implementazione delle procedure. Nei server WEB il dispatcher è il thread che legge dalla rete le richieste da elaborare in arrivo. Dopo averle esaminate, sceglie un thread lavoratore inattivo (ad esempio uno che è bloccato) e gli consegna la richiesta. A questo punto, il dispatcher sveglia il lavoratore inattivo spostandolo dallo stato bloccato (blocked) allo stato pronto (ready).
- Potrebbe generare un file *.h per importare l'interfaccia e tutte le costanti e i tipi necessari.

L'obiettivo di XML è fornire un modo standardizzato per specificare le strutture di dati in modo tale che quando si scambiano dati, è possibile capire cosa è stato inviato. Il DTD (Document Type Definition) specifica come viene descritta la struttura dei dati: istruzioni di elaborazione, dichiarazioni, commenti ed elementi. Utilizzando il DTD, il documento XML può essere interpretato correttamente da un programma semplicemente analizzando il documento usando la grammatica fornita dal DTD. L'idea è simile a IDL tranne per il fatto che invece di definire i parametri come combinazioni di tipi standard, un DTD descrive i documenti arbitrari come dati semi-strutturati. Utilizzando XML è possibile scambiare dati tramite server HTTP e Web ed elaborarli automaticamente.

L'uso dell'XML riduce l'universalità del browser poiché ora un browser necessita di programmi aggiuntivi per gestire specifici linguaggi di markup sviluppati utilizzando XML (in qualche modo simile ai plug-in ma più comprensivo in termini di funzionalità). Tuttavia, questo non è un grosso problema dal momento che il browser è per l'uomo mentre XML è per l'elaborazione automatizzata. XML può essere utilizzato come linguaggio intermedio per il marshalling/serializzazione degli argomenti quando si invocano servizi su Internet.

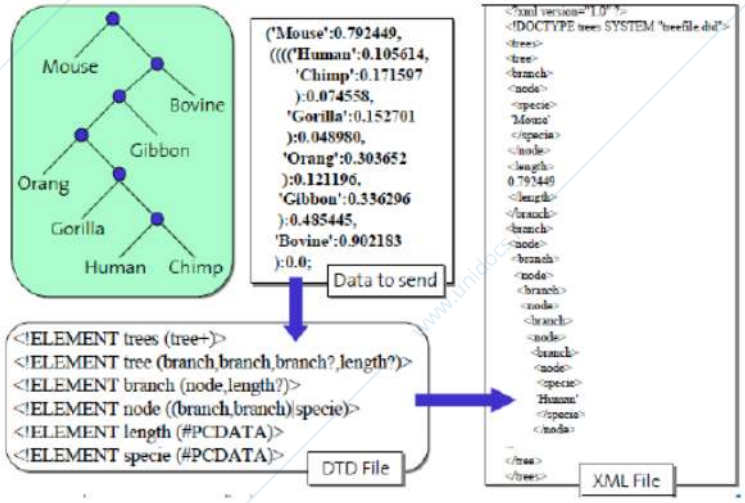

Figura 32: XML e DTD

Enterprise Application Integration (EAI) (integrazione d'applicazioni di impresa) si riferisce al processo d'integrazione tra diversi tipi di sistemi informatici attraverso l'utilizzo di software e soluzioni architetturali. Un altro problema relativo all'accesso ai sistemi EAI attraverso un'interfaccia web è la rappresentazione dei dati relazionali. Se si utilizza HTML, i dati vengono formattati per la presentazione, non per l'elaborazione. Se si utilizzano XML e DTD, la struttura è più adatta per l'elaborazione ma l'elaborazione è ad-hoc (è possibile definire qualsiasi DTD desiderato). Quindi, è stato proposto XML Schema per consentire l'elaborazione di database come query su documenti XML. XML Schema è un linguaggio di definizione dei dati per documenti XML che consente di trattarli come dati relazionali in modo standardizzato. Lo XML Schema o Schema XML è un linguaggio di descrizione del contenuto di un file XML, l'unico che finora abbia raggiunto la validazione ufficiale del W3C (la 1.1). Come tutti i linguaggi di descrizione del contenuto XML, il suo scopo è delineare quali elementi sono permessi, quali tipi di dati sono ad essi associati e quale relazione gerarchica hanno fra loro gli elementi contenuti in un file XML. Ciò permette principalmente la convalida del file XML, ovvero la verifica che i suoi elementi siano in accordo con la descrizione in linguaggio XML Schema. Lo XML Schema permette inoltre l'estrazione da un file XML, o meglio una visione da un file XML, di un insieme di oggetti con determinati attributi e una struttura.

Che cosa c'è di diverso tra XML Schema e DTD? XML Schema:

- utilizza la stessa sintassi di XML (i DTD hanno una sintassi diversa);
- fornisce una serie più ampia di tipi (simili a quelli in SQL);
- consente di definire tipi complessi dai tipi di base;

- supporta vincoli di integrità chiave e referenziale;
- può essere utilizzato dai linguaggi di query (XQuery, ad esempio) per analizzare i documenti XML e trattarli come dati relazionali;
- può essere utilizzato per specificare il modello di dati utilizzato da un'interfaccia del servizio Web.

Il Web Services Description Language (WSDL) è un XML-based linguaggio di descrizione dell'interfaccia (IDL) che viene utilizzato per descrivere le funzionalità offerte da un servizio web. WSDL (Web Services Description Language) può essere meglio compreso quando lo affrontiamo come una versione XML di un IDL che copre anche gli aspetti relativi all'integrazione tramite Internet e la maggiore complessità dei servizi Web. Un IDL convenzionale non include informazioni quali:

- posizione del servizio (implicita nella piattaforma e rilevata tramite associazione statica o dinamica)
- diversi binding (in genere un IDL è associato a un protocollo di trasporto)
- serie di operazioni (poiché un'interfaccia definisce un singolo punto di accesso e non esiste una sequenza di operazioni coinvolta nello stesso servizio)

Un IDL nelle piattaforme di integrazione di middleware e applicazioni aziendali convenzionali ha diversi scopi:

- descrizione delle interfacce dei servizi forniti (ad es. RPC).
- fungere da rappresentazione intermedia per colmare l'eterogeneità fornendo una mappatura dei tipi di dati nativi alla rappresentazione intermedia associata all'IDL in questione.
- servire come base per lo sviluppo attraverso un compilatore IDL che produce stub e librerie che possono essere utilizzate per sviluppare l'applicazione.

1.3.2 WSDL

La specifica WSDL è nella versione v2.0 (giugno 2007) ed illustra come descrivere le diverse parti che comprendono un'interfaccia di servizio Web (Figura 33):

- il sistema di tipi utilizzato per descrivere il modello di dati di servizio (schema XML)
- i messaggi coinvolti nell'interazione con il servizio
- le singole operazioni composte da 4 possibili schemi di scambio di messaggi
- le serie di operazioni che costituiscono un servizio
- la mappatura su un protocollo di trasporto per i messaggi
- la posizione in cui risiede il fornitore di servizi
- gruppi di posizioni che possono essere utilizzati per accedere allo stesso servizio

Include anche una specifica che indica come associare WSDL ai protocolli SOAP, HTTP (POST / GET) e MIME.

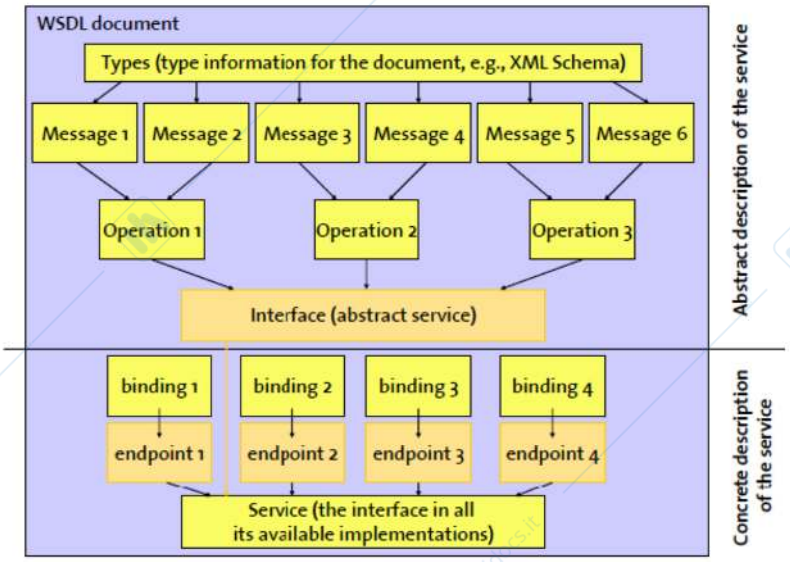

Figura 33: Elementi di WSDL 2.0

La Figura 34 descrive i livelli che partono dalla discovery fino ad arrivare al trasporto effettivo dei dati.

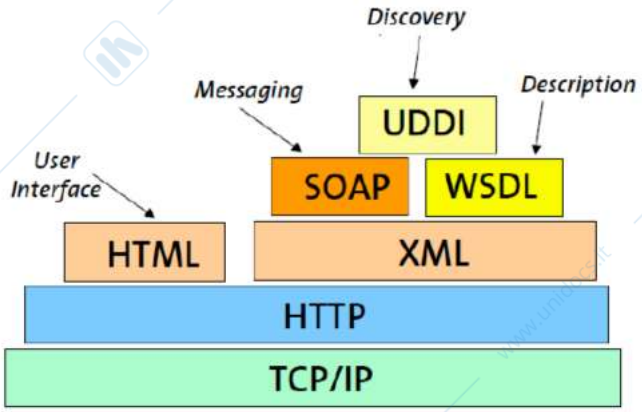

Figura 34: Layering

Quando è possibile interagire con qualsiasi fornitore di servizi utilizzando il protocollo SOAP standard, è ancora necessario:

- descrivere i servizi (WSDL)
- scoprire i servizi (UDDI, Universal Description, Discovery and Integration)

Il processo di scoperta e descrizione dei servizi è rappresentato dalla Figura 35.

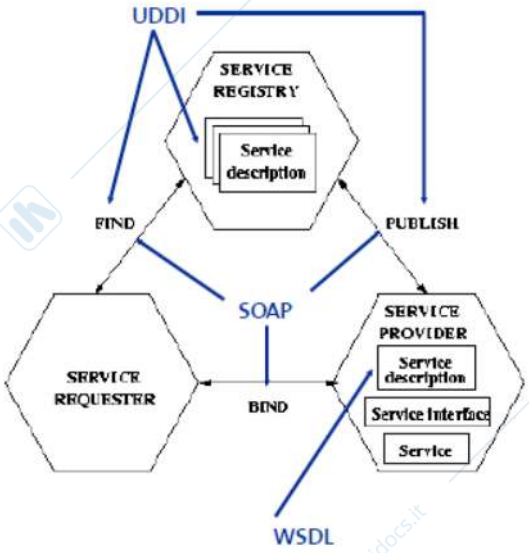

Figura 35: Il ruolo di WSDL/UDDI

I tipi in WSDL vengono utilizzati per specificare il contenuto dei messaggi (messaggi normali e messaggi di errore) che verranno scambiati come parte delle interazioni con i servizi Web. Il sistema dei tipi si basa in genere su XML Schema (strutture e tipi di dati). Il supporto è obbligatorio per tutti i processori WSDL. Un elemento di estensibilità può essere utilizzato per definire uno schema diverso da XML schema.

PURCHASE ORDER TYPE

```
<element name="PO" type="tns:POType"/>
<complexType name="POType">
  <all>
    <element name="id" type="string"/>
    <element name="name" type="string"/>
    <element name="items">
      <complexType>
        <all>
          <element name="item" type="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
        </all>
      </complexType>
    </element>
  </all>
</complexType>
```

ITEM TYPE

```
<complexType name="Item">
  <all>
    <element name="quantity" type="int"/>
    <element name="product" type="string"/>
  </all>
</complexType>
```

INVOICE TYPE

```
<element name="Invoice" type="tns:InvoiceType"/>
<complexType name="InvoiceType">
  <all>
    <element name="id" type="string"/>
  </all>
</complexType>
```

Figura 36: Data Types

I messaggi hanno un nome che li identifica in tutto il documento XML. Sono divisi in parti, ognuna delle quali è una struttura di dati rappresentata in XML. Ogni parte deve avere un tipo (tipi di base o complessi, precedentemente dichiarati nel documento WSDL). Un elemento del messaggio WSDL corrisponde al contenuto del corpo di un messaggio SOAP. Osservando i tipi e osservando il messaggio, è possibile creare un messaggio SOAP

che corrisponda alla descrizione WSDL (e ciò può essere fatto automaticamente poiché la descrizione è basata su XML e anche i tipi supportati da SOAP). Un messaggio non definisce alcuna forma di interazione, è solo un messaggio. In WSDL 1.0, la struttura di un "messaggio" è stata definita in modo esplicito, elencando tutte le sue parti. In WSDL 2.0, un "componente di riferimento del messaggio" è definito come parte di un'operazione e contiene tre elementi:

- etichetta del messaggio (message label): indica il modello di messaggio utilizzato per il messaggio.
- direzione (direction): indica se si tratta di un messaggio in entrata o in uscita.
- elemento del messaggio (message element): rappresenta il contenuto effettivo del messaggio espresso in termini dei tipi precedentemente definiti.

I guasti sono un tipo speciale di messaggio utilizzato per segnalare errori. La Figura 37 rappresenta un esempio di messaggio di errore.

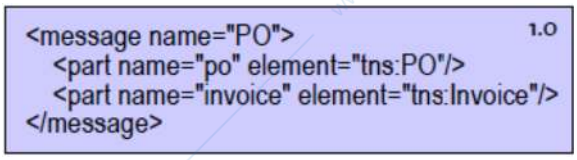

Figura 37: Message Faults

In WSDL 2.0, un'operazione è un insieme di messaggi e guasti. Il sequenziamento e il numero di messaggi nell'operazione sono determinati dal modello di scambio di messaggi. Lo stile di un'operazione distingue tra comportamento simile a RPC, scambio di messaggi orientato al documento o attributi set-and get-of. Le operazioni possono essere annotate con caratteristiche e proprietà (ad es. Affidabilità, sicurezza, routing).

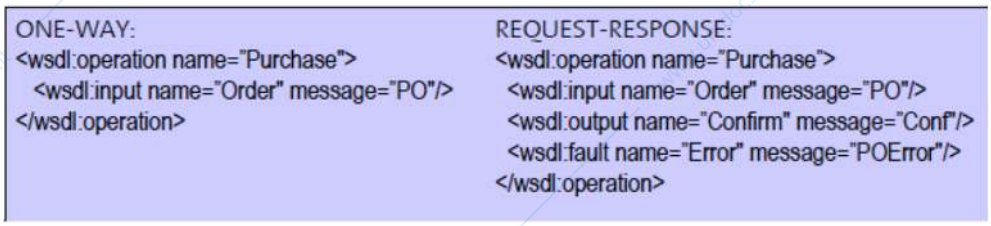

Figura 38: Operations

Un'interfaccia corrisponde alla definizione astratta di un servizio Web (astratta perché non specifica alcuna informazione su dove risiede il servizio o quali protocolli vengono utilizzati per invocare il servizio Web). L'interfaccia è semplicemente un elenco di operazioni che possono essere utilizzate in quel servizio Web. Le operazioni non sono definite da sole, ma solo come parte di un'interfaccia.

```
<message name="m1">
 <part name="body" element="tns:GetCompanyInfo"/>
</message>

<message name="m2">
 <part name="body" element="tns:GetCompanyInfoResult"/>
 <part name="docs" type="xsd:string"/>
 <part name="logo" type="tns:ArrayOfBinary"/>
</message>

<portType name="pt1">
 <operation name="GetCompanyInfo">
     <input message="m1"/>
     <output message="m2"/>
 </operation>
</portType>
```

Figura 39: Interfaces

Un'associazione (binding) definisce i formati dei messaggi e i dettagli del protocollo per le operazioni e i messaggi di un determinato Port Type (punto finale nella nuova specifica). Un'associazione corrisponde a un singolo end point (ovvio poiché deve fare riferimento alle operazioni e ai messaggi dell'endpoint). Un end point può avere diversi binding, fornendo così diversi canali di accesso allo stesso servizio astratto. L'associazione è estensibile con elementi che consentono di specificare i mapping dei messaggi e delle operazioni su qualsiasi formato o protocollo di trasporto. In questo modo, WSDL non è specifico del protocollo.

Un end point specifica l'indirizzo di un'associazione, ovvero come accedere al servizio, utilizzando un protocollo e un formato particolari. Gli endpoint possono specificare solo un indirizzo e non devono contenere alcuna informazione vincolante. L'endpoint viene spesso specificato come parte di un servizio piuttosto che da solo.

I servizi raggruppano una raccolta di porte e diventano quindi la definizione completa del servizio vista dall'esterno: un servizio supporta diversi protocolli (ha diversi binding). L'accesso al servizio, in base a un determinato protocollo, avviene tramite un indirizzo specifico (specificato nelle porte di ciascun binding), mentre le operazioni e i messaggi da scambiare sono definiti nell'End Point. Le porte che fanno parte dello stesso servizio potrebbero non comunicare tra loro. Le porte che fanno parte dello stesso servizio sono considerate alternative tutte con lo stesso comportamento (determinato dall'End Point), ma raggiungibili attraverso protocolli diversi.

Lo stile di un messaggio SOAP controlla il formato dell'elemento <soap: Body>: la Figura 40 mostra un confronto tra lo stile RPC e quello REST. Inoltre, come possiamo vedere nella Figura 41 lo stile del messaggio SOAP è specificato nel documento WSDL nella sezione binding.

• **RPC** style, an extra child element of the Body is added to identify the method to be called. Parameters are listed inside this one.	• **Document** style, the Body contains an arbitrary XML document.
<soap:Body> <tns:ConfirmOrder xmlns:tns="http://my.package/"> <number xsi:type="xsd:integer">1234</number> <confirm xsi:type="xsd:boolean">true</confirm> </tns:Method> </soap:Body>	<soap:Body> <tns:order number="00001234"> Purchase Order Confirmation <tns:status>Confirmed</tns:status> ... </tns:order> </soap:Body>
ConfirmOrder(number,confirm),	*Send(OrderDocument),*

Figura 40: RPC vs REST

```
<wsdl:binding name="…" type="tns:port
type name"> <soap:binding
style="rpc|document"> … <soap:binding>
</wsdl:binding>
```

Figura 41: Controllo dello stile

Lo stile si riflette anche nell'elemento <wsdl: message>, che può avere:

- Per lo stile REST, al massimo 1 <wsdl: partelement = "...">
- Per lo stile RPC, qualsiasi numero di <wsdl: parttype = "..."> elementi. Con lo stile RPC è necessario solo un elemento <wsdl: types> per definire i tipi complessi utilizzati dai parametri.

Lo stile REST è più generale in quanto può implementare lo stile RPC utilizzando un XML Schema appropriato.

Nell'informatica, la serializzazione è il processo di traduzione delle strutture dati o dello stato dell'oggetto in un formato che può essere archiviato (ad esempio, in un file o buffer di memoria) o trasmesso (ad esempio, attraverso un collegamento di rete) e ricostruito più tardi (possibilmente in un diverso ambiente informatico). Quando la serie di bit risultante viene riletta secondo il formato di serializzazione, può essere utilizzata per creare un clone semanticamente identico dell'oggetto originale. Per molti oggetti complessi, come quelli che fanno ampio uso di riferimenti, questo processo non è semplice. La serializzazione di oggetti orientati agli oggetti non include nessuno dei metodi associati a cui erano precedentemente collegati. Questo processo di serializzazione di un oggetto è anche chiamato marshalling di un oggetto in alcune situazioni. L'operazione opposta, che estrae una struttura di dati da una serie di byte, è la deserializzazione (chiamata anche unmarshalling). Il contenuto dei dati serializzati di un messaggio SOAP può anche essere codificato in diversi modi:

- Letterale (seguire la definizione del XML Schema del WSDL)
- Codifica SOAP (seguire la Sezione 5 delle specifiche SOAP 1.1)

La codifica è specificata nella sezione di binding del documento WSDL, per ciascun messaggio scambiato come parte di ciascuna operazione, come mostrato nella Figura 42.

```
<wsdl:input>
<soap:body use="literal"/>
<soap:body use="encoded"
  encodingStyle="http://schemas.xmlsoap.org/soap/
  encod ing"/>
</wsdl:input>
```

Figura 42: Controllo della codifica

Lo stile e la codifica sono generalmente associati (REST/Literal e RPC/Encoded). Tuttavia, sono ortogonali e possono essere selezionati indipendentemente quando viene distribuito il servizio. Questi sono anche ortogonali rispetto al MEP (Message Exchange Pattern) utilizzato dall'operazione. Un modello di messaggistica è un modello architettonico orientato alla rete che descrive come due diverse parti di un sistema di trasmissione

di messaggi si collegano e comunicano tra loro. Un modello di scambio di messaggi (MEP) descrive il modello di messaggi richiesto da un protocollo di comunicazione per stabilire o utilizzare un canale di comunicazione. Esistono due principali schemi di scambio di messaggi: un modello di richiesta-risposta e un modello a senso unico. Ad esempio, HTTP è un protocollo modello richiesta-risposta e UDP è un modello unidirezionale.

Come prima approssimazione, una conversazione modella le sequenze di operazioni che un client può invocare come parte dell'interazione con un servizio Web. In generale, una conversazione definisce un'interazione complessa tra più servizi Web che comporta lo scambio di più messaggi e l'invocazione di diverse operazioni in un ordine ben definito. In questo contesto, un protocollo di coordinamento specifica l'insieme di conversazioni corrette tra i vari servizi. La descrizione dell'interfaccia di servizio (WSDL) elenca solo le operazioni disponibili, ma non specifica quale sia l'ordine corretto per invocarle.

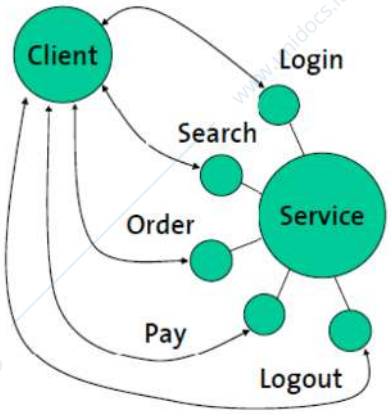

Figura 43: Conversazioni

WSDL definisce l'interfaccia di un servizio Web in termini di quali sono i messaggi scambiati (ricevuti e prodotti dal servizio). Un documento WSDL struttura anche i messaggi in coppie (che corrispondono alle operazioni fornite da un servizio). Tuttavia, WSDL non contiene ulteriori informazioni che specificano quale sia l'ordine corretto di invocazione delle varie operazioni. Se un'operazione non deve (ancora) essere invocata, viene restituito un messaggio di errore. Dal punto di vista del cliente, ciò rende difficile garantire automaticamente la correttezza dell'interazione. Per quanto riguarda il servizio, potrebbe essere necessaria un'interazione tra più operazioni per mantenere le informazioni sulla sessione (Statefulinteraction). Queste informazioni vengono anche utilizzate per imporre la correttezza dell'interazione. Qualunque meccanismo venga impiegato, questi vincoli non emergono nella descrizione dell'interfaccia WSDL. L'obiettivo è rendere lo sviluppo il più automatico possibile. Esistono molti modi diversi di modellare le conversazioni. Estendendo la descrizione di base dell'interfaccia di un servizio, il requisito più importante è descrivere quali sono tutte le possibili conversazioni valide con esso. Pertanto, la nostra descrizione deve definire tutte le sequenze accettabili di invocazioni operative. A livello di interfaccia, l'obiettivo è sensibilizzare i clienti su quale sia il protocollo di coordinamento supportato dal servizio in modo che interagiscano correttamente con esso. Internamente, lo scopo del modello di conversazione è supportare e facilitare lo sviluppo dell'attuazione del servizio basandosi sull'infrastruttura di coordinamento per

condurre la conversazione. I peer da coordinare sono generalmente etichettati con il loro ruolo nella conversazione. Il protocollo di coordinamento specifica:

- quali sono i possibili messaggi (o operazioni) che compongono la conversazione
- per ciascun messaggio, quale ruolo è l'originatore e chi è il destinatario previsto
- quali sono i vincoli di ordinamento nella sequenza di scambi di messaggi
- a quali condizioni la conversazione è considerata completata

Queste informazioni completano il tipo di porta WSDL (o interfaccia) che deve essere fornito da ciascun ruolo, che elenca solo i messaggi accettati.

Un modo semplice per descrivere un protocollo di coordinamento è usare una macchina a stati finiti (Figura 44):

- gli stati modellano le tappe seguite dalla conversazione
- le transizioni corrispondono alle operazioni invocate dal cliente sul servizio (o in generale, ai messaggi che vengono ricevuti o inviati dal servizio)
- l'insieme di operazioni valide (che possono essere legalmente invocate dal cliente) cambiano con lo stato della conversazione. Ogni volta che viene scambiato un messaggio, lo stato della conversazione deve essere aggiornato
- le operazioni valide sono quelle associate alle transizioni in uscita dallo stato corrente

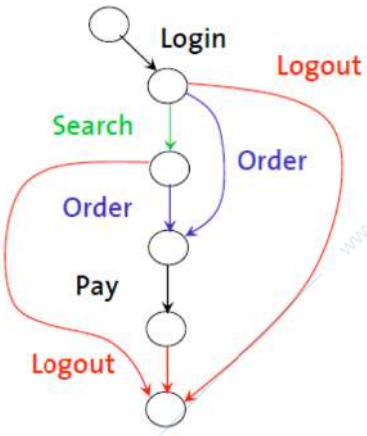

Figura 44: Finite State Machines

Quando sono coinvolti più peer, le macchine a stati possono ancora essere utilizzate con l'aggiunta di ruoli, identificando il mittente e il destinatario di ciascun messaggio. Sono possibili anche rappresentazioni alternative (con la stessa semantica): i diagrammi di sequenza UML consentono di visualizzare l'evoluzione della conversazione nel tempo, mostrando ogni messaggio mentre viene scambiato tra due ruoli. I diagrammi di sequenza mancano del supporto di ramificazione, per cui diagrammi alternativi devono essere utilizzati per mostrare percorsi alternativi nella conversazione. Inoltre, quando sono coinvolti molti ruoli, la leggibilità del diagramma potrebbe risentirne.

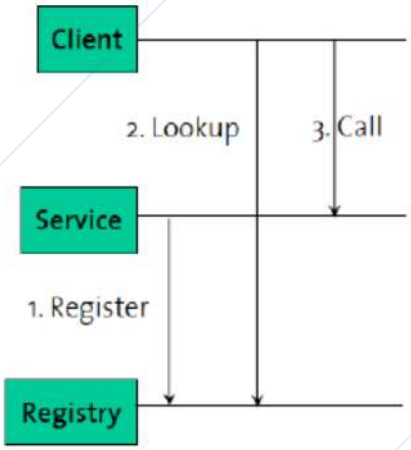

Figura 45: UML Sequence Diagram

I diagrammi di attività UML supportano anche più ruoli attraverso gli swimlane[1]. A differenza dei diagrammi di sequenza, supportano anche i rami (e il parallelismo) nel flusso della conversazione. Le attività modellano l'invocazione sincrona o asincrona di un'operazione. L'attività è posizionata nella struttura del ruolo del cliente ed è etichettata con il ruolo target. Gli stati finali vengono utilizzati per identificare le conversazioni completate.

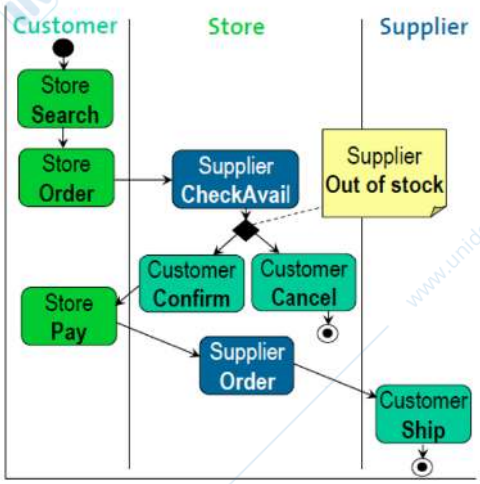

Figura 46: UML Activity Diagram

Uno dei vantaggi dell'utilizzo di XML auto-descrittivo[2] per la codifica dei messaggi SOAP è che diventa davvero facile sviluppare i corrispondenti parser (per leggere i messaggi) e gli emettitori (per scrivere i messaggi). Vi sono tuttavia alcuni svantaggi, non

[1]Lo swim lane (o swimlane) è un elemento grafico utilizzato nei diagrammi di flusso (flowcharts) che suddivide visualmente le responsabilità dei sub-processi di una macroattività. Gli swimlane possono essere strutturati sia orizzontalmente che verticalmente.

[2]i dati XML sono autodescrittivi: anche i partner saranno in grado di comprenderli ed elaborarli. Inoltre, essi possono essere gestiti anche in futuro quando le applicazioni che li hanno generati saranno diventate obsolete.

solo legati al sovraccarico delle prestazioni (l'analisi e la convalida XML sono costose), ma anche alle limitazioni dell'XML come formato di scambio di dati (allegati SOAP per lo scambio di dati binari). Un altro problema è che la parseability non garantisce l'interoperabilità. Il fatto che tutte le parti interessate possano analizzare i messaggi SOAP, risolve solo il problema di interoperabilità a livello di sintassi. Sebbene i progressi siano già stati compiuti standardizzando la sintassi, c'è ancora molto da fare per concordare la semantica dei messaggi. A livello di SOAP, potrebbe essere necessario applicare trasformazioni ai messaggi scambiati (gli strumenti di mappatura dei dati per EAI non sono scomparsi, sono appena diventati basati su XML/XSLT). A livello di WSDL, dovrebbe essere possibile descrivere la semantica, oltre alla sintassi delle interfacce di servizio. WSDL definisce un'interfaccia di servizio (o tipo di porta) come un insieme di operazioni, raggruppando coppie di messaggi, che sono definiti in termini di parti (con nome e tipo di dati, definiti in uno schema XML). Da una descrizione WSDL è possibile inferire (e validare) automaticamente la struttura dei corrispondenti messaggi SOAP.

Figura 47: Trasformazione dei dati

Nell'esempio in Figura 47, sia il client che il server utilizzano WSDL per descrivere la loro interfaccia e SOAP per scambiare un messaggio. Anche se ipotizziamo che le due parti siano in qualche modo compatibili, questa standardizzazione non garantisce l'interoperabilità, a meno che entrambi i servizi non utilizzino lo stesso schema XML e (estratto dalla descrizione dell'interfaccia), concordino sulla semantica del messaggio. Se è impossibile risolvere questa mancata corrispondenza, il messaggio non può essere inviato direttamente, ma deve essere trasformato tra i due schemi preservandone la semantica. Questa trasformazione può avvenire sul lato client (il client sa come adattarsi a un determinato server), sul lato server (il server supporta diversi modelli di dati) o - in uno scenario di vera integrazione - nel mezzo (usando un mediatore o ESB). Un enterprise service bus (ESB) è un'infrastruttura software che fornisce servizi di supporto a service-oriented architecture complesse. Si basa su sistemi disparati, interconnessi con tecnologie eterogenee, e fornisce in maniera consistente servizi di coordinamento, sicurezza, messaggistica, instradamento intelligente e trasformazioni, agendo come una dorsale attraverso la quale viaggiano servizi software e componenti applicativi.

In generale, date N parti da integrare, devono essere definite (e mantenute) trasformazioni fino a $N(N-1)$. Ciò presuppone che dovrebbe essere possibile trasformarsi direttamente tra gli schemi utilizzati da ciascuna delle parti. Se permettiamo di comporre più trasformazioni e introdurre una rappresentazione intermedia, il numero di trasformazioni viene ridotto a $2N$. Se i vari schemi sono completamente sovrapposti (ovvero, le trasformazioni non perdono informazioni quando vengono applicate ai messaggi), è an-

che possibile evitare una singola rappresentazione intermedia e applicare una sequenza di trasformazioni al fine di ottenere il formato desiderato. In questo caso, abbiamo solo bisogno di N trasformazioni.

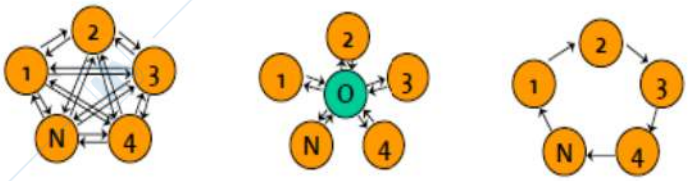

Figura 48: Ridimensionamento delle informazioni

Ogni elemento sintattico di un'interfaccia di servizio (messaggio, struttura di dati o operazione) ha un significato semantico preciso associato ad esso. Questo significato dovrebbe essere preso in considerazione dai clienti che invocano il servizio, in modo che possano comprendere quale funzionalità viene offerta dal servizio. La semantica può essere modellata utilizzando:

- documenti standard
- vincoli (ad esempio, nel caso di domini con elementi enumerabili)
- ontologie (che definiscono formalmente un vocabolario di termini e relazioni). In informatica, un'ontologia è una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse. Più nel dettaglio, si tratta di una teoria assiomatica del primo ordine esprimibile in una logica descrittiva. Il termine ontologia formale è entrato in uso nel campo dell'intelligenza artificiale e della rappresentazione della conoscenza, per descrivere il modo in cui diversi schemi vengono combinati in una struttura dati contenente tutte le entità rilevanti e le loro relazioni in un dominio. I programmi informatici possono poi usare l'ontologia per una varietà di scopi, tra cui il ragionamento induttivo, la classificazione, e svariate tecniche per la risoluzione di problemi.
- contratti (pre-condizioni e post-condizioni)

In uno scenario di integrazione, l'infrastruttura del middleware dovrebbe preservare la semantica delle applicazioni da integrare e fornire supporto per la mediazione (la trasformazione dei messaggi tra diverse rappresentazioni mediante la mappatura di concetti condivisi tra tutte le applicazioni). Se i servizi sono descritti con WSDL, la semantica è molto ridotta. Pertanto, ci sono molte estensioni a WSDL che possono essere usate per modellare la semantica, ad es. usando l'RDF (Resource Description Framework) e OWL (Web Lingua Ontologica).

Il Resource Description Framework (RDF) è lo strumento base proposto da W3C per la codifica, lo scambio e il riutilizzo di metadati strutturati e consente l'interoperabilità semantica tra applicazioni che condividono le informazioni sul Web. È costituito da due componenti:

- RDF Model and Syntax: espone la struttura del modello RDF, e descrive una possibile sintassi.

- RDF Schema: espone la sintassi per definire schemi e vocabolari per i metadati.

L'RDF Data Model si basa su tre principi chiave:

1. Qualunque cosa può essere identificata da un Uniform Resource Identifier (URI).
2. The least power: utilizzare il linguaggio meno espressivo per definire qualunque cosa.
3. Qualunque cosa può dire qualunque cosa su qualunque cosa.

Il Web Ontology Language (OWL) è un linguaggio di markup per rappresentare esplicitamente significato e semantica di termini con vocabolari e relazioni tra gli stessi. Esistono varie versioni del linguaggio, che differiscono molto tra di loro. Lo scopo di OWL è descrivere delle basi di conoscenze, effettuare delle deduzioni su di esse e integrarle con i contenuti delle pagine Web. OWL intende rendere possibile, ad esempio: ricerche nel Web che superino i problemi di omonimia e ambiguità presenti nelle normali ricerche testuali; applicazioni che effettuino delle deduzioni sui dati. La rappresentazione dei termini e delle relative relazioni è chiamata ontologia. Insieme a RDF, di cui è un'estensione, OWL fa parte del progetto del web semantico.

1.4 UDDI

Un servizio è fornito da un server situato in un determinato indirizzo IP e in ascolto di una determinata porta. Il binding è il processo di mappatura del nome di un servizio su un indirizzo e una porta che possono essere utilizzati a scopi di comunicazione. É possibile eseguire il binding:

- localmente: il client deve conoscere il nome (indirizzo) dell'host del server
- distribuito: esiste un servizio separato (posizione del servizio, nome e servizi di directory, ecc.) incaricato di mappare nomi e indirizzi. Questo servizio deve essere raggiungibile per tutti i partecipanti. Con un binding distribuito, sono possibili diverse operazioni generali:
  - REGISTER (esportazione di un'interfaccia): un server può registrare i nomi dei servizi e la porta corrispondente
  - RECESSO (WITHDRAW): un server può ritirare un servizio
  - LookUP (importazione di un'interfaccia): un client può chiedere al binder per l'indirizzo e la porta di un determinato servizio.

  Deve esserci anche un modo per individuare il binder (posizione predefinita, variabili di ambiente, trasmissione a tutti i nodi che cercano il binder).

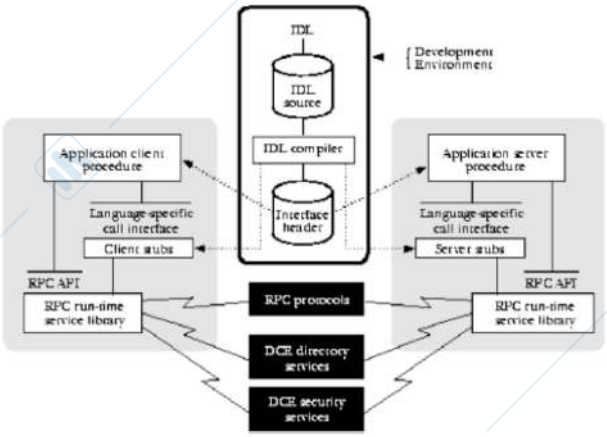

Figura 49: RPC binding

Come abbiamo visto nella Figura 34 e nella Tabella 1, UDDI serve per scoprire servizi ed è il protocollo/linguaggio più vicino all'utente. Infatti, quando è possibile interagire con qualsiasi fornitore di servizi utilizzando il protocollo SOAP standard, è ancora necessario:

- descrivere i servizi (WSDL)
- scoprire i servizi (UDDI)

Universal Description, Discovery and Integration (UDDI) è il nome di un protocollo e di un sistema di registry web che mettono a disposizione informazioni sui web services, la loro localizzazione e le loro interfacce. I registry sono distribuiti su siti di pubblico accesso detti Operator Sites, che si sincronizzano tra loro mediante replicazione. Lo scopo dei registry UDDI è fornire una piattaforma pubblica per il service discovery. I servizi web, infatti, mostrano la loro massima utilità quando è possibile utilizzarli dinamicamente, localizzandoli ed interfacciandovisi in maniera semplice, rapida e (semi)automatica. La specifica UDDI è probabilmente quella che si è evoluta maggiormente da tutte le specifiche che abbiamo visto finora. L'ultima versione è la versione 3 (OASIS Standard febbraio 2005):

- La versione 1 ha definito la base per un registro dei servizi alle imprese.
- La versione 2 ha adattato il funzionamento del registro a SOAP e WSDL
- La versione 3 ridefinisce il ruolo e lo scopo dei registri UDDI, sottolinea il ruolo delle implementazioni private e affronta il problema dell'interazione tra i registri UDDI privati e pubblici.

Inizialmente, UDDI è stato concepito come un "Registro delle imprese universale" simile ai motori di ricerca (ad esempio Google) che doveva essere utilizzato come meccanismo principale per trovare servizi elettronici forniti da aziende di tutto il mondo. Ciò ha innescato una notevole quantità di attività attorno a scenari molto avanzati e complessi (Web semantico, associazione dinamica ai partner, runtime/selezione automatica dei

partner, ecc.). Oggi UDDI è molto più pragmatico e riconosce la realtà delle interazioni B2B: si presenta come "infrastruttura per i servizi Web", il che significa lo stesso ruolo di un servizio di nomi e directory (ad esempio, binder in RPC), ma applicato a servizi Web e utilizzato principalmente in ambienti con restrizioni (internamente all'interno di un'azienda o tra un insieme predefinito di partner commerciali). I servizi offerti tramite Internet ad altre società richiedono molte più informazioni rispetto a un tipico servizio middleware. Questa documentazione presenta tre aspetti:

- informazioni di base;
- categorizzazione;
- dati tecnici.

L'UDDI è uno degli standard alla base del funzionamento dei Web service: è stato progettato per essere interrogato da messaggi in SOAP e per fornire il collegamento ai documenti WSDL che descrivono i vincoli protocollari e i formati dei messaggi necessari per l'interazione con i Web service elencati nella propria directory. Le informazioni che costituiscono una registrazione UDDI (Figura 50) sono costituite da cinque parti, tutte definite come strutture XML:

- businessService: un elenco di tutti i servizi Web offerti dall'entità aziendale.
- businessEntity: contiene dati relativi ai contatti e agli indirizzi di un fornitore di servizi.
- bindingTemplate: gli aspetti tecnici del servizio offerto
- tModel ("modello tecnico"): un elemento generico che può essere utilizzato per memorizzare informazioni aggiuntive sul servizio, in genere informazioni tecniche aggiuntive su come utilizzare il servizio, condizioni d'uso, garanzie, ecc.

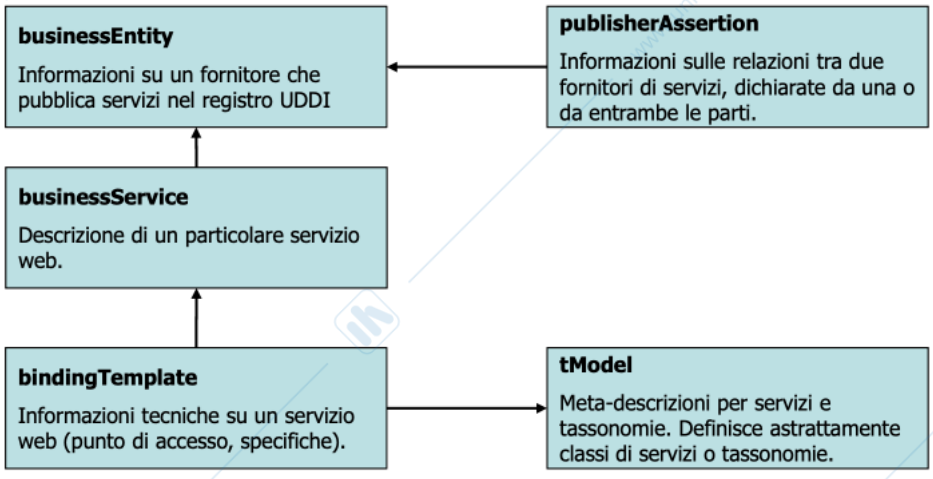

Figura 50: Registrazione UDDI

Insieme, questi elementi vengono utilizzati per fornire le tre componenti che formano una registrazione UDDI:

- Pagine bianche (White Pages): indirizzo, contatti (dell'azienda che offre uno o più servizi) e identificativi;
- Pagine gialle (Yellow Pages): categorizzazione dei servizi basata su tassonomie standardizzate (quale tipo di servizi sono offerti e un elenco dei diversi servizi offerti);
- Pagine verdi (Green Pages): informazioni (tecniche) dei servizi forniti dall'azienda.

I fornitori di servizi web possono inserire le proprie business info nel registry, definire relazioni con altri fornitori, pubblicare e classificare i propri servizi. Gli utenti interessati ad un servizio web possono ricercare un servizio compatibile con le loro esigenze secondo vari criteri, ricercare informazioni sul fornitore di servizi e/o sui suoi partner, oppure avere accesso immediato alle informazioni necessarie ad interfacciarsi con il servizio (WSDL).

1.4.1 BusinessEntity

Le informazioni generiche sulle pagine bianche e gialle su un fornitore di servizi sono archiviate in businessEntity, che contiene i seguenti dati:

- discoveryURLs: un elenco di URL che puntano a meccanismi di rilevazione dei servizi alternativi basati su file.
- nome (informazioni testuali)
- descrizione dell'attività (informazioni testuali)
- contatti (informazioni testuali)
- businessServices: un elenco di servizi forniti da businessEntity
- identifierBag: un elenco di identificatori esterni
- categoryBag: un elenco di categorie di attività (ad es. Industria, categoria di prodotto, regione geografica)

Ogni businessEntity ha un attributo businessKey, che fornisce un ID unico per il fornitore.

BusinessEntity non deve necessariamente essere la società: è pensato per rappresentare qualsiasi entità che fornisce servizi. Può essere un dipartimento, un gruppo di persone, un server, un insieme di server, ecc.

1.4.2 BusinessService

I servizi forniti da un businessEntity sono descritti in termini commerciali usando gli elementi businessService. Un businessEntity può avere diversi businessServices, ma un businessService appartiene a un businessEntity. Il businessService può effettivamente essere fornito da un businessEntity diverso da quello in cui si trova l'elemento. Questo si chiama proiezione e consente di includere servizi forniti da altre organizzazioni come parte dei propri servizi. BusinessService contiene:

- serviceKey, un ID che identifica in modo univoco il servizio.
- name, il nome del servizio.

- description, una descrizione del servizio.
- categorybag, che contiene una classificazione del servizio.
- bindingTemplates, che contiene riferimenti ai record contenenti di dati tecnici per l'interfacciamento col servizio.

1.4.3 BindingTemplate

Le strutture bindingTemplate forniscono un supporto alla determinazione dell'entry point che consente di raggiungere il Web Service e alla descrizione di caratteristiche tecniche uniche di una data implementazione. Quindi, l'elemento bindingTemplate fornisce le informazioni tecniche utilizzabili per interfacciarsi con un determinato servizio. Contiene le seguenti informazioni:

- bindingKey: ID unico per il binding template
- serviceKey
- description: descrizione del bing template
- accessPoint/hostingRedirector: l'indirizzo di rete del servizio fornito. Il primo viene utilizzato per definire, mediante una stringa, come accedere al servizio. Le modalità d'accesso sono sette: via posta elettronica (mailto:), via HTTP (http:) o HTTPS (https:), via FTP (ftp:), via fax (fax:) o telefono (phone:), oppure attraverso un altro mezzo (other:) che dovrà essere specificato mediante un modello. L'elemento hostingRedirector possiede un attributo bindingKey che viene utilizzato per referenziare un altro bindingTemplate. Questo elemento è particolarmente utile quando più servizi fanno riferimento allo stesso punto d'accesso.
- tModels: un elenco di voci corrispondenti a tModels associate a questa specifica associazione. L'elenco include riferimenti ai modelli t, documenti che descrivono questi modelli, brevi descrizioni, ecc.
- categoryBag: informazioni aggiuntive sul servizio e sulla sua associazione (ad es. Se si tratta di un'associazione di prova, è in produzione, ecc.)

Un businessService può avere diversi modelli di bind, ma un bindingTemplate ha un solo businessService. Il binding template può essere visto come una cartella in cui sono inserite tutte le informazioni tecniche di un servizio.

1.4.4 tModel

L'azienda utilizza le strutture dati viste in precedenza per definire cosa offre e come. Se però, si vuole integrare i servizi con i propri partner è essenziale che non ci siano equivoci, e che ogni campo informativo sia estremamente chiaro. A tale proposito, il registro UDDI utilizza un approccio simile ai metadati, ovvero informazioni sui dati. Il registro UDDI utilizza la struttura tModel per modellare informazioni descrittive del comportamento del servizio, tipi di specifiche utilizzate o standard a cui un servizio è conforme, etc. Queste informazioni non sono caratteristiche di una particolare azienda o entità che pubblica e tanto meno di un singolo servizio, ma generalmente interessano un insieme di servizi, indipendentemente dal tipo di azienda che li fornisce. Per questo

motivo la struttura dati tModel, diversamente dalle altre strutture business, non è inclusa nell'elemento businessEntity. Un tModel è un contenitore generico di informazioni in cui i progettisti possono scrivere qualsiasi informazione tecnica associata all'uso di un servizio Web:

- l'interfaccia e il protocollo effettivi utilizzati, incluso un puntatore alla descrizione WSDL
- descrizione del protocollo aziendale e conversazioni supportate dal servizio
- "qualsiasi concetto che non sia meglio rappresentato da una delle altre strutture di dati UDDI"

Un tModel è un documento con una breve descrizione delle informazioni tecniche e un puntatore alle informazioni effettive. Contiene:

- tModelKey
- Nome e descrizione
- overviewDoc: (con un URL di panoramica e useType che indicano dove trovare le informazioni e il suo formato, ad esempio "testo" o "wsdldescription")
- identifierBag & categoryBag

Un tModel può puntare ad altri tModel e alla fine saranno standardizzate diverse forme di tModel (tModel per servizi WSDL, tModel per servizi basati su EDI, ecc.).

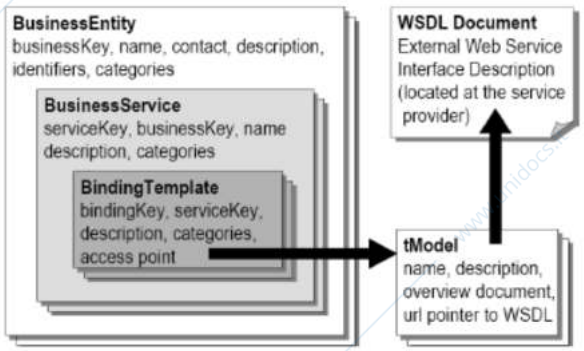

Figura 51: UDDI data model

tModel segue solo tre convenzioni:

- Struttura che fornisce informazioni necessarie per determinare compatibilità tra servizi (riferita mediante bindingTemplate).
- Struttura per la definizione di tassonomie che consentono di classificare aziende e servizi (riferita mediante categoryBag). Tassonomia significa classificare e associare ogni nozione ad un mondo, quindi descrivere.
- Struttura per contestualizzare un'informazione, mediante l'uso di namespace (rifeta mediante identifierBag)

Nel primo caso il tModel viene usato per specificare il tipo di protocollo di rete, formato di scambio dei messaggi, regole che definiscono la sequenza di scambio, etc., da usare per interagire con un particolare servizio, o insieme di servizi. Ad esempio supponiamo che due prodotti software comunicano tra loro attraverso un mezzo di comunicazione che aderisce a una certa specifica convenuta tra le due parti in gioco, ebbene il progettista di questa specifica può stabile un'unica identità tecnica e registrarla in un tModel presso un UDDI registry. Tutti quei servizi che si registrano successivamente e che ritengono di essere conformi a quella specifica possono includere nel loro bindingTemplate il riferimento al particolare modello rappresentativo dell'identità della specifica. In altre parole, il tModel registrato in questo modo diventa un "fingerprinter" (impronta digitale) di una data specifica.

Nel secondo caso, particolari strutture tModel di categoria possono essere "linkate" tra di loro per classificare aziende e servizi, formando strutture ad albero. Ad esempio, si può creare un modello di servizi di tipo banking e un modello di servizi per l'elaborazione del credito, che referenzia il primo. A sua volta, il secondo modello può essere puntato da un campo della sottostruttura categoryBag associata a un particolare servizio. Questo sarà automaticamente classificato come un servizio di banca per l'elaborazione del credito.

Nel terzo caso, invece, il tModel può essere usato per definire un particolare spazio dei nomi, in cui alcune coppie nome-valore assumono un certo significato. Ad esempio, si potrebbe creare una lista di coppie nome-valore per definire le varie componenti di un indirizzo, come il destinatario, il CAP, la località e la provincia, in maniera tale che un sistema automatico possa interpretarlo ed elaborarlo correttamente. In particolare, il campo destinatario è molto variabile a dispetto degli altri che in generale possono essere definiti mediante una maschera. In generale, la varietà di tipologie di indirizzi porta a definire schemi complessi attraverso coppie nome-valore raggruppate sotto lo stesso modello, diciamo modello "indirizzo". Per fare questo si deve utilizzare l'elemento identifierBag, che è una lista di coppie nome-valore definite mediante l'elemento keyedReference.

Una volta definita la struttura, l'elemento identifierBag può essere aggiunto al modello "indirizzo". Successivamente, questo può essere referenziato dalla sottostruttura address per interpretare in modo opportuno le linee d'indirizzo.

1.4.5 API UDDI

Le API UDDI sono gli strumenti primari per la pubblicazione e la scoperta delle informazioni sulle aziende, che si sono registrate presso un registro UDDI, e i relativi servizi forniti. La specifica 2.0 definisce un modello di programmazione per inviare e per ricevere approssimativamente 40 messaggi SOAP. Questi consentono di invocare le funzioni per l'accesso al registro e la manipolazione delle informazioni ivi contenute. La specifica UDDI fornisce una serie di API (Application Program Interfaces) che forniscono l'accesso a un sistema UDDI. Esse si distinguono in:

- API per la pubblicazione (publisher API): per pubblicare e modificare le informazioni in un registro UDDI. Tutte le operazioni in questa API sono atomiche in senso transazionale. Forniscono un modello di programmazione per memorizzare e modificare le strutture dati. Per l'utilizzo delle API di pubblicazione è richiesto un accesso autorizzato al registro ed è responsabilità dell'operatore realizzare un protocollo di autenticazione efficiente e compatibile con le tali API.

- API per l'interrogazione (Inquiry API): per individuare e trovare dettagli sulle voci in un registro UDDI. Supporta una serie di pattern (navigazione, drill-down, in-

vocazione). consentono di consultare le informazioni contenute nel registro. Per le API di consultazione, invece, non è richiesta alcuna forma di autenticazione in quanto si limitano solo alla lettura di dati pubblici, a tutto vantaggio dei fornitori. Quando subentrano errori nelle Inquiry API, essi sono riportati come errori SOAP. Le funzioni di ricerca effettuano ricerche nel registro in base a parole chiave e restituiscono elenchi riepilogativi con informazioni generali (chiave, nome e descrizione) su attività commerciali o servizi corrispondenti. I find qualifiers (qualificatori) vengono utilizzati per ordinare i risultati e controllare la corrispondenza delle parole chiave: alternanza tra AND/OR, case sensitive/insensitive, uso di caratteri jolly e categorie. Per ridurre al minimo il numero di richieste, è possibile nidificare le query di ricerca.

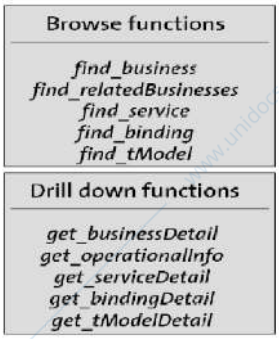

Figura 52: Inquiry API

- UDDI Security: per il controllo dell'accesso al registro UDDI (basato su token). L'API di pubblicazione richiede l'autenticazione dell'utente mediante un token di sessione e in genere utilizza SOAP su HTTPS. Il registro esegue il controllo dell'accesso per tutte le funzioni di pubblicazione: le informazioni sulle voci possono essere modificate solo dal proprietario. Le informazioni sulla categoria e i riferimenti con chiave associati alle voci vengono convalidati prima di accettare nuove informazioni nel registro. Le funzioni di eliminazione vengono utilizzate per rimuovere dal registro le voci identificate dalla loro chiave. La rimozione di un'azienda rimuoverà tutti i servizi ad essa associati.

Figura 53: Security API

- UDDI Subscription: consente ai clienti di abbonarsi alle modifiche alle informazioni nel registro UDDI (le modifiche possono essere incluse nella richiesta di abbonamento).
- UDDI Replication: come eseguire la replica delle informazioni su nodi in un registro UDDI
- Trasferimento di custodia e proprietà UDDI (UDDI Custody and Ownership transfer): per modificare il proprietario (editore) delle informazioni e la custodia da un nodo all'altro all'interno di un registro UDDI

UDDI fornisce anche un set di API per i client di un sistema UDDI:

- Listener di abbonamenti UDDI (UDDI Subscription Listener): il lato client dell'API di abbonamento
- Set di valori UDDI: utilizzato per convalidare le informazioni fornite a un registro UDDI

Le API UDDI forniscono ai programmatori un modo semplice per interagire con i dati archiviati in un registro UDDI tramite il protocollo SOAP.

La specifica UDDI è piuttosto completa e comprende molti aspetti di un registro UDDI dal suo uso alla sua distribuzione su più nodi e la coerenza dei dati in un registro distribuito. La maggior parte dei registri UDDI sono privati e in genere fungono da fonte di documentazione per gli sforzi di integrazione basati sui servizi Web. I registri UDDI non sono necessariamente intesi come repository finale delle informazioni relative ai servizi Web. Anche nella versione "universale" del repository, l'idea è quella di standardizzare le funzioni di base e quindi creare strumenti proprietari che sfruttino il repository di base. In questo modo è possibile personalizzare il design e mantenere la compatibilità necessaria tra i repository. Pur essendo la parte più visibile degli sforzi sui Web Service, l'UDDI è forse il meno critico a causa della complessità delle interazioni B2B (stabilire fiducia, contratti, vincoli e procedure legali, ecc.). L'obiettivo finale è, ovviamente, la completa automazione, ma fino a quando ciò non accade è necessario risolvere un lungo elenco di problemi ed è necessaria molta più standardizzazione.

C'erano alcuni registri universali UDDI in funzione (gestiti da IBM, Microsoft, SAP, ecc.). Questi registri erano molto visibili e spesso la prima cosa che si vedeva dei servizi Web. La maggior parte delle voci in esse non funzionava o non corrispondeva a nessun servizio reale. Questa è stata una fonte di critica ai servizi Web in generale. La critica non è stata del tutto immeritata, ma è spesso fuorviata: ciò che c'era da criticare non era l'UDDI stesso, ma l'uso che ne era stato fatto. L'UDDI consiste in un'infrastruttura di supporto per servizi Web in ambienti ben definiti e vincolati (ovvero, senza accesso pubblico e in cui esiste un contesto che fornisce le informazioni mancanti). La maggior parte dei registri UDDI in atto oggi sono registri privati che operano all'interno di società (ricordiamo che l'uso più ampio dei servizi Web oggi è per EAI convenzionale) o gestiti da un insieme di società in modo privato. L'UDDI è ora diventato il modo accettato per documentare i servizi Web e fornire le informazioni mancanti nelle descrizioni WSDL.

Quindi, un registro UDDI fornisce un meccanismo, indipendente dalle piattaforme hardware e software, per descrivere e trovare informazioni su aziende e servizi. Le strutture dati UDDI forniscono un framework per la descrizione delle informazioni principali sulle aziende e relativi servizi, mentre le API UDDI offrono un modello di programmazione per effettuare operazioni di pubblicazione e di consultazione, indipendentemente dal linguaggio usato per implementarle.

1.5 BPEL - Business Process Execution Language

1.5.1 Business Process

Un business process è una raccolta di attività collegate che trovano fine alla consegna di un servizio o prodotto a un cliente. Il processo aziendale (o business process) è un insieme di attività interrelate, svolte all'interno dell'azienda nell'ambito della gestione operativa delle sue funzioni aziendali, che creano valore trasformando delle risorse (input del processo) in un prodotto finale (output del processo) a valore aggiunto, destinato ad un soggetto interno o esterno all'azienda (cliente). Il processo è teso al raggiungimento di un obiettivo aziendale, determinato in sede di pianificazione, se questa è presente. Tanto le risorse quanto il prodotto possono essere beni, servizi o informazioni oppure una combinazione di questi elementi. La trasformazione dell'input in output può essere eseguita con l'impiego di lavoro umano, di macchine o di entrambi.

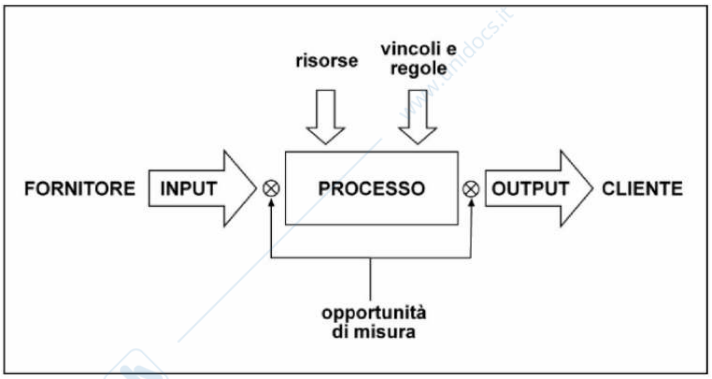

Figura 54: Business Processes

Un business process descrive le procedure chiave all'interno di un'organizzazione e queste procedure comportano:

- Passaggi multipli
- Numerose persone
- Grandi quantità di risorse

Nelle grandi aziende ci sono molti fattori che aumentano la complessità dei processi aziendali:

- Più persone non hanno informazioni
- Conformità alle regole non garantite
- I processi non sono ben documentati
- La società non dispone di strumenti di monitoraggio
- Step, persone e risorse non sono adeguatamente coordinati

I sistemi di gestione del flusso di lavoro (Workflow Management Systems) cercano di affrontare questi problemi automatizzando gli aspetti di coordinamento di un processo aziendale, ovvero chi deve fare cosa, quando e con quali strumenti. Un sistema di

Workflow Management è definito come un sistema che definisce, crea e gestisce l'esecuzione di workflows attraverso l'uso di software, che è eseguito all'interno di uno o più motori di workflow e che è capace di interpretare la definizione dei processi, interagire con i partecipanti al workflow e, qualora richiesto, di invocare l'uso di strumenti IT e applicazioni.

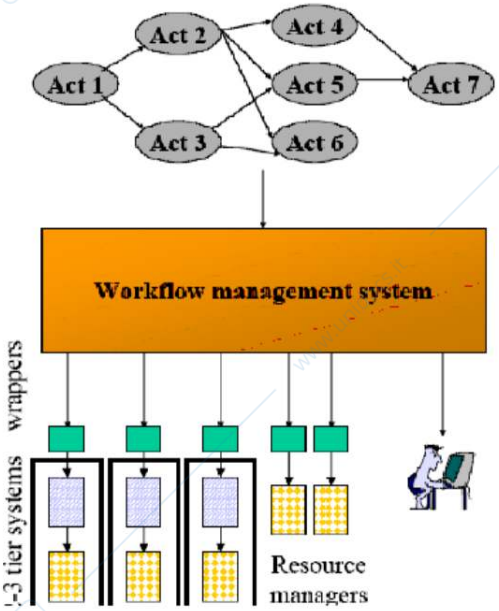

Figura 55: Business Processes

La complessità di molti processi aziendali li rende un elemento chiave nella capacità di un'azienda di evolversi. La reingegnerizzazione dei processi aziendali è un modo per definire, comprendere meglio e ottimizzare questi processi:

- Potenziamento delle capacità di elaborazione (1,5 milioni / anno fatture)
- Riduzione delle spese generali
- Riduzione dei tempi di lavorazione (da 7 giorni a 4 ore) e riduzione dei costi di lavorazione.

I processi che sono già implementati con strumenti di middleware tradizionali sono molto difficili da capire, per non parlare della modifica. Un processo può essere visto come un programma: deve essere ben documentato, in un formato adatto e avere un ciclo di vita coerente. Per i processi che non sono implementati nel software, lo sforzo di reingegnerizzazione spesso porta a tentativi di implementazione. É qui che la tecnologia del flusso di lavoro svolge un ruolo importante. Oggi, molte delle singole attività all'interno di un processo aziendale vengono eseguite direttamente o indirettamente con i computer. Poiché facciamo sempre più affidamento sui computer, ogni PC e workstation in cui si svolgono le singole attività diventano un repository di informazioni isolato in cui risiede parte del processo aziendale. Ognuno di quei computer è un'isola di informazioni. Ma quei computer non possono facilmente parlare tra loro e non è semplice raccogliere tutte le informazioni memorizzate in ogni isola. In queste circostanze, è molto difficile

avere una visione globale di ciò che sta accadendo: monitoraggio, audit, raccolta dei dati, ecc. Tutte le isole di informazioni devono essere collegate insieme per costruire un unico sistema in cui i processi aziendali esistono come concetti fisici e non come entità astratte.

Gli obiettivi del B2B o dell'e-commerce sono simili a quelli del flusso di lavoro:

- Ridurre il costo delle transazioni
- Fornire una visione di alto livello delle operazioni IT
- Associare l'IT ai processi aziendali
- Ridurre i costi generali e i tempi di esecuzione

Il maggiore ostacolo al flusso di lavoro era l'eterogeneità delle applicazioni. Questo ostacolo si allontana in larga misura attraverso l'uso dei servizi web. I flussi di lavoro o i processi sembrano essere la lingua migliore per specificare ad alto livello come una raccolta di servizi dovrebbe interagire. I flussi di lavoro sono spesso visti come il linguaggio delle architetture orientate al servizio.

La portabilità e l'interoperabilità dei modelli di processi aziendali definiscono composizioni eseguibili e protocolli astratti di coordinamento, per questo motico è fondamentale stardizzare la composizione di business process.

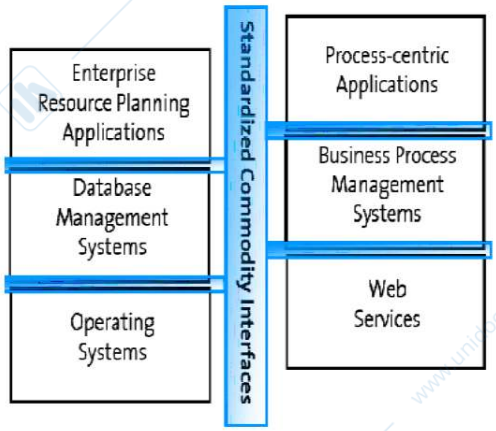

Figura 56: Standardization of Business Processes

Una volta che i servizi di base iniziano a essere pubblicati sul Web, la cosa naturale da fare è trovarli e combinarli in servizi più complessi e a valore aggiunto (Value added services). Supponendo che siano disponibili i servizi di base necessari, è possibile definire la logica di business di un'applicazione fuori dalla sua composizione. É possibile creare diverse applicazioni componendo gli stessi servizi in modi diversi. I servizi compositi sono ancora servizi, ovvero servizi che possono essere composti: la composizione è ricorsiva. La composizione di servizi Web fornisce astrazioni per la modellazione di alto livello delle composizioni e l'infrastruttura per eseguirle in modo efficiente. Alcuni esempi di composizione sono:

- Cercare il prezzo più economico per un libro/film/ brano su tutti gli e-store Internet e acquistalo, assicurandoti che venga spedito prima del tuo compleanno.

- Ricercare da diversi motori di ricerca Web, unisci tutti i risultati, rimuovendo i duplicati
- Pianificare una vacanza: ordina biglietti aerei, prenota hotel e auto per una serie di destinazioni
- Convertire i prezzi delle quotazioni di borsa in una valuta diversa
- Ricevere informazioni meteo e rapporti neve e valanghe per le stazioni sciistiche più vicine
- Raccogliere le offerte dai fornitori, selezionare il vincitore dopo una scadenza e informare tutti i partecipanti del risultato

Composizione e coordinamento possono essere visti come due facce dello stesso problema. La composizione modella la struttura interna e l'implementazione di un servizio. I protocolli di coordinamento si concentrano sulle interazioni esterne di un insieme di servizi. La composizione deve seguire il protocollo di coordinamento di tutti i suoi servizi componenti. Il protocollo di coordinamento per il servizio composito può essere dedotto osservando la sua struttura interna.

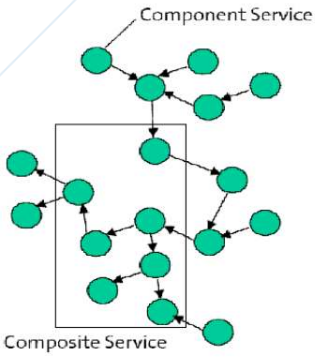

Figura 57: Composition and coordination of Business Processes

Il semplice esempio, presentato in Figura 58, definisce un protocollo per scambiare un ordine di acquisto tra due servizi Web di due società diverse. Viene inviata una richiesta di ordine di acquisto, seguita da un riconoscimento immediato e una risposta di conferma in seguito.

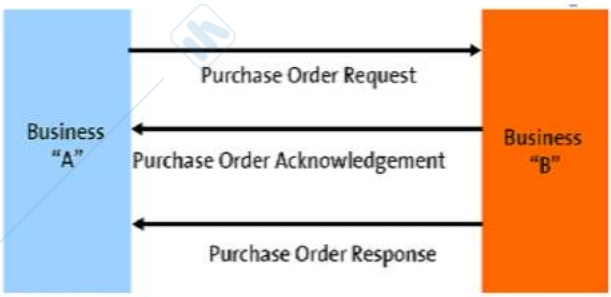

Figura 58: Example of Business Processes

La visione di coordinamento (Figura 59) su questa interazione definisce lo scambio osservabile di messaggi tra i servizi Web. Questo è descritto dal protocollo aziendale (o processo pubblico) che collega le due parti.

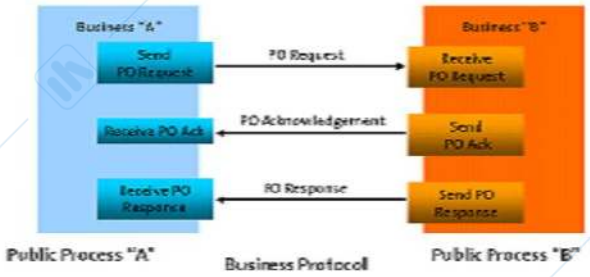

Figura 59: Coordination View

La vista della composizione (Figura 60) su questa interazione definisce il comportamento interno di ciascuno dei servizi Web coinvolti. Questo è descritto dall'orchestrazione (o processo privato) di ciascuna delle parti coinvolte.

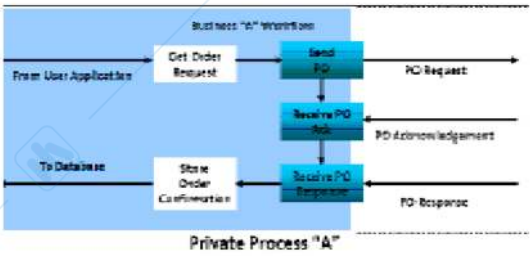

Figura 60: Composition View

I processi pubblici e privati di ciascuno dei servizi Web devono essere coerenti affinché l'interazione funzioni. É necessario concordare il processo pubblico (protocollo aziendale) che coordina il modo in cui i servizi Web interagiscono. Il processo privato che orchestra lo scambio di messaggi con i sistemi interni è generalmente mantenuto riservato.

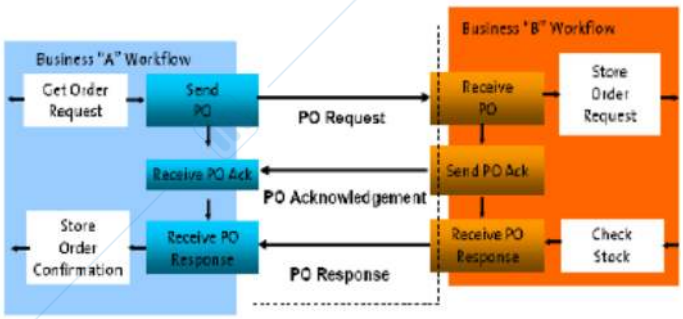

Figura 61: Complete View

Gli aspetti della composizione che andremo a vedere sono:

- Servizi Web del modello di componente (sincrono e asincrono) - Component Model Web Services. Il modello di componenti dei servizi Web (WSCM) è un modello di componenti incentrato sui servizi Web e XML per applicazioni Web interattive. Un modello di componente definisce i presupposti sui componenti che dovrebbero essere composti. In generale, la composizione di componenti omogenei è più semplice della composizione di componenti eterogenei poiché l'infrastruttura corrispondente è meno complessa. In un caso, tutti i componenti potrebbero essere servizi Web (accessibili con i protocolli SOAP/HTTP e descritti da un documento WSDL). Dall'altro estremo, i componenti sono solo "servizi" a cui è possibile accedere utilizzando una varietà di meccanismi di invocazione.

- Modello di composizione formale: statechart, reti di Petri, calcolo π. I modelli di composizione (o orchestrazione) definiscono come costruire un sistema coerente da una raccolta di servizi; definiscono:

  - Qual è l'ordine in cui vengono invocati i servizi e quali sono le condizioni alle quali un determinato servizio può o non può essere invocato (flusso di controllo)
  - Come i servizi scambiano dati tra loro (flusso di dati) e come interagiscono
  - Una qualche forma di gestione delle eccezioni, ovvero cosa succede se un servizio non è disponibile

  Esistono molti linguaggi diversi per la composizione di modelli:

  - XML-Oriented: gerarchie di attività
  - Basato sul flusso di lavoro: WS-BPEL (XML), JOpera (Visual)
  - Altri: basati su regole, diagrammi di attività UML, dati guidati

  La composizione può essere definita lungo la dimensione spaziale o temporale. Gli ADL (Architectural Description Language) descrivono l'architettura spaziale di un sistema in termini di componenti e connettori. I linguaggi di definizione del flusso di lavoro usano la nozione di processo per definire il modo in cui i componenti interagiscono nel tempo. I linguaggi di programmazione tradizionali soddisfano anche i requisiti per i linguaggi di composizione (Figura 62).

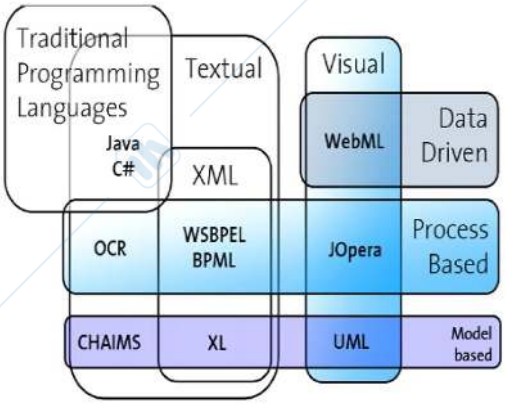

Figura 62: Linguaggi per la composizione

- Modello di selezione del servizio (Service Selection Model). Oltre a specificare quali sono i messaggi da scambiare, un modello di composizione dovrebbe anche definire come selezionare i servizi che dovrebbero ricevere o inviare tali messaggi. Al fine di migliorare la riusabilità di una composizione, tali servizi di solito non sono codificati nella composizione, ma associati in essa in momenti diversi:
  - Tempo di composizione
  - Tempo di compilazione e distribuzione
  - Tempo di avvio
  - Tempo di invocazione

  Il modello di selezione del servizio definisce il modo in cui i servizi sono associati in una composizione:
  - Associazione statica - Static Binding: l'URL dell'endpoint del servizio è hard-coded
  - Associazione dinamica per riferimento - Dynamic binding by reference: l'URL del servizio viene calcolato e memorizzato in una variabile
  - Binding dinamico per ricerca - Dynamic binding by lookup: prima di ogni richiamo del servizio viene inviata una query a un registro per individuare un'implementazione adeguata
  - Selezione dinamica dell'operazione - Dynamic Operation Selection: non vengono fatte ipotesi sulla firma del servizio arbitrario da invocare
- Modello di trasferimento dati (Data Transfer Model): I servizi in genere interagiscono scambiando alcuni dati. Non tutti i linguaggi di composizione includono un modello esplicito del flusso di dati, I dati non sono sempre trattati allo stesso modo: i dati a livello di composizione sono modellati a un livello più fine in quanto vengono utilizzati per controllare la composizione (controllare i rami del flusso) e hanno tipi di dati associati. I dati specifici dell'applicazione sono visti come un puntatore opaco (URL) che è previsto.

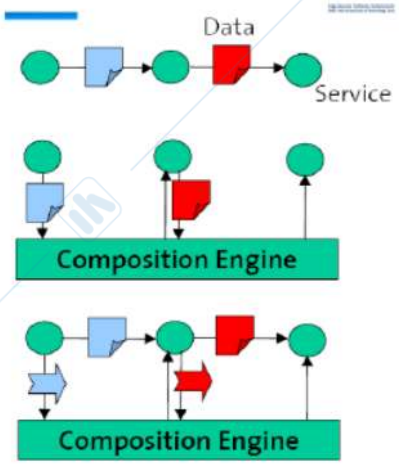

Figura 63: Data Model

I trasferimenti di dati (Data Transfer) definiscono come (e quando) i servizi dovrebbero scambiare dati:

- Whiteboard: le chiamate di servizio leggono i loro input e depositano i loro risultati in una raccolta di variabili. Ogni richiamo del servizio definisce una mappatura da e verso la lavagna. I dati possono anche essere copiati esplicitamente tra le variabili.
- Grafico del flusso di dati (Data Flow Graph):Attraverso i Data Flow Diagram si definiscono soprattutto come fluiscono (e vengono elaborate) le informazioni all'interno del sistema, quindi l'oggetto principale è il flusso delle informazioni o, per meglio dire, dei dati. Motivo per il quale diventa fondamentale capire dove sono immagazzinati i dati, da che fonte provengono, su quale fonte arrivano, quali componenti del sistema li elaborano. Le componenti di questo tipo di diagramma sono:
  * Funzioni, rappresentate da bolle;
  * Flussi di dati, rappresentati da frecce;
  * Archivi di dati, rappresentati da scatole aperte;
  * Agenti esterni o Input/Output di dati, rappresentati da scatole chiuse.

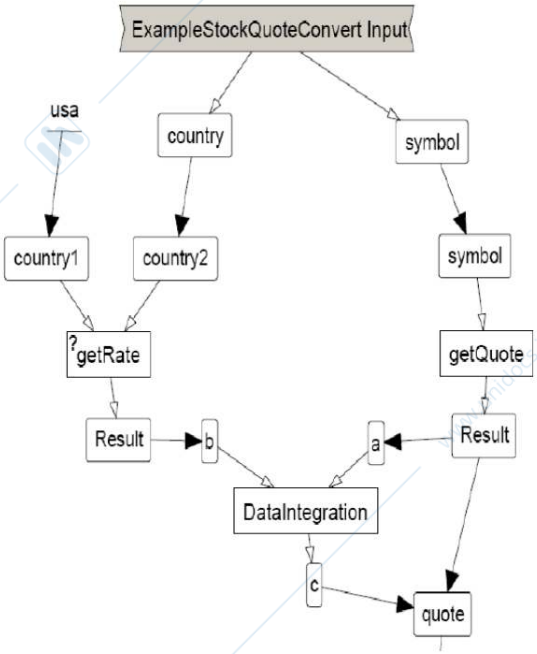

Figura 64: Data Flow Graph

- Gestione delle eccezioni: importante in quanto le composizioni di servizi dovrebbero esplicitamente modellare cosa fare se un servizio non è disponibile (timeout) o se la sua chiamata non riesce. La gestione delle eccezioni, basata sul flusso, utilizza i normali costrutti del flusso di controllo per diramare dopo che una chiamata al servizio non è riuscita. I blocchi Try / Catch vengono utilizzati in modo simile per associare un gestore di eccezioni a una serie di chiamate di servizio. Le regole possono anche essere associate a una composizione al fine di rilevare condizioni eccezionali globali.

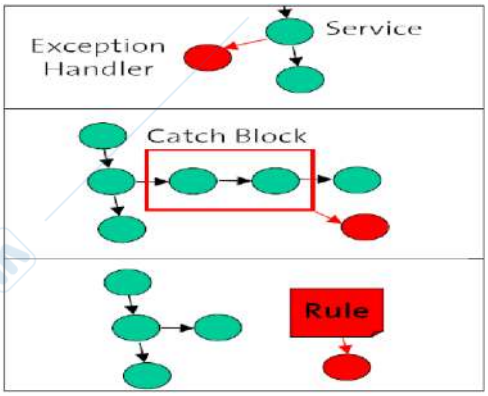

Figura 65: Exception Handling

- Transazioni: Il comportamento transazionale è modellato raggruppando le invocazioni del servizio e dichiarando le proprietà di atomicità e isolamento per il gruppo. Per supportare transazioni a lungo termine, ogni operazione di servizio può anche essere associata a un gestore di compensazione, che viene invocato solo se l'operazione deve essere annullata. Quando non si verificano guasti, il motore di composizione esegue un protocollo 2PC con tutti i servizi della regione atomica. Se si verifica un errore, il motore richiama i gestori di compensazione dei servizi che potrebbero essere richiamati correttamente. In parole povere, un processo aziendale spesso

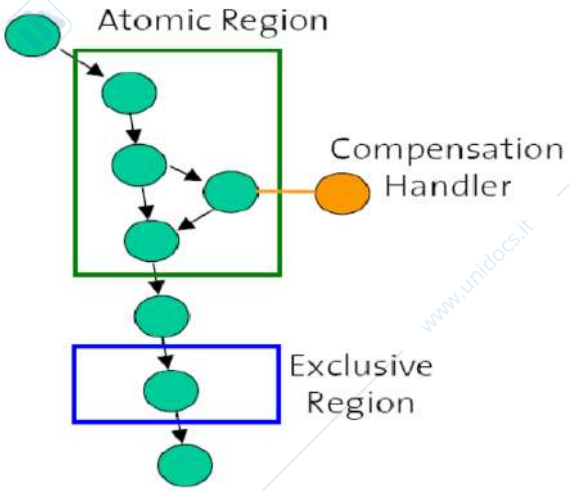

Figura 66: Transaction

contiene diverse transazioni nidificate. L'intera transazione commerciale può fallire o può essere annullata dopo che molte transazioni allegate sono già state elaborate. Quindi è necessario invertire l'effetto ottenuto durante l'esecuzione del processo. Ad esempio, un processo di pianificazione del viaggio può includere diverse transazioni nidificate per prenotare un biglietto, prenotare un hotel e un'auto. Se il viaggio viene annullato, le transazioni di prenotazione devono essere compensate da transazioni di annullamento nell'ordine appropriato. Un gestore della compensazione è un contenitore per le attività che eseguono azioni di compensazione.

1.5.2 WS-BPEL

BPEL (sigla di Business Process Execution Language) è un linguaggio basato sull'XML costruito per descrivere formalmente i processi commerciali e industriali in modo da permettere una suddivisione dei compiti tra attori diversi. Un'applicazione BPEL viene invocata come Web service e interagisce con il mondo esterno esclusivamente invocando altri Web service. In questo senso, essa stessa rappresenta una forma di coordinazione di servizi Web, permettendo altresì di comporre questi ultimi in maniera ricorsiva. BPEL costituisce il linguaggio standard per la Process Orchestration e rappresenta sicuramente uno dei componenti fondamentali per realizzare delle Service Oriented Architecture: esso, infatti, permette l'integrazione e la cooperazione di diverse componenti, generando così dei servizi web dal valore aggiunto che mantengono le caratteristiche di modularità e scalabilità. L'ambiente runtime all'interno del quale viene eseguito il generico processo è detto motore BPEL. Lo standard che definisce l'uso di BPEL nelle interazioni tra Web service è chiamato BPEL4WS o WS-BPEL. Nell'aprile del 2003 BPEL è stato sottoposto ad OASIS che lo ha standardizzato.

Quindi, WS-BPEL è uno standard per specificare il comportamento dei processi aziendali basato esclusivamente su servizi Web. WS-BPEL è un linguaggio basato sulla sintassi XML, che non si occupa direttamente dell'implementazione della lingua, ma solo della semantica delle primitive che fornisce. L'ultima versione delle specifiche è 2.0.

WS-BPEL è nato come fusione di XLANG (Microsoft) e WSFL (IBM), di conseguenza, ci sono problemi formali con il linguaggio. Il linguaggio viene utilizzato per definire:

- processi eseguibili, con le interazioni effettive tra i diversi servizi. Rappresentano i modelli di comportamento reale di un partecipante in un'interazione business. Questi processi possono implementare la logica aziendale interna di un servizio Web (Composizione). I processi eseguibili descrivono la composizione (o l'orchestrazione) di diverse interfacce del servizio Web (tipi di porta WSDL). Il risultato di una composizione che utilizza BPEL è destinato a essere pubblicato in modo ricorsivo come servizio Web.

- processi astratti, che modellano i messaggi scambiati dalle parti coinvolte in un protocollo aziendale senza rivelare dettagli sull'attuazione interna (coordinamento). I processi astratti definiscono i vincoli sull'interfaccia WSDL di un servizio Web in modo che possa essere utilizzato correttamente. Un esempio di applicazione sono i RosettaNetPIP (Partner Interface Processes), i quali standardizzano interfacce e protocolli lungo una catena di fornitura e-commerce. RosettaNet è costituita da un consorzio di importanti computer, elettronica di consumo, produttori di semiconduttori, società di telecomunicazione e logistica che lavorano insieme per creare e implementare standard di processo e-business aperti a livello industriale. Questi standard formano un linguaggio e-business comune, allineando i processi tra i partner della catena di approvvigionamento su base globale. Il processo astratto (pubblico) per un servizio Web può essere generalizzato dall'implementazione concreta (privata) eseguibile, se questo è costruito usando BPEL. La definizione astratta del processo può anche guidare l'implementazione del processo eseguibile privato dietro un servizio Web.

L'obiettivo è definire il coordinamento e la composizione dei servizi Web in modo portatile.

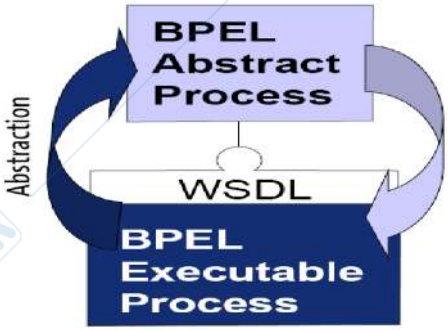

Figura 67: Processi Astratti

L'affermarsi dei Web Service rende più agevole e funzionale lo sviluppo di processi di business, in cui ogni servizio esposto corrisponde a un'attività che può essere messa a disposizione per essere invocata da agenti esterni. In questo modo, è possibile gestire dei workflow operativi, generati da attività atomiche e sapientemente invocati da un unico gestore logico esterno. Si è resa quindi evidente la necessità di mettere a disposizione uno strumento standardizzato che avesse la possibilità di interagire con diversi servizi web, esposti da diversi fornitori, componendoli al fine di creare nuovi servizi dal valore aggiunto. Il coordinamento dei vari servizi può essere realizzato principalmente in due modi:

- Orchestrazione: presuppone che il controllo del workflow venga mantenuto da un solo gestore logico, che interagisce, anche per processi di lunga durata, con altri servizi, interni od esterni.
- Coreografia: consiste in un approccio più "collaborativo", ossia ogni partecipante esegue il suo lavoro, e il sistema di coreografia si occupa esclusivamente di tenere traccia delle interazioni avvenute.

Queste due tipologie di architetture garantiscono un accoppiamento debole tra chi richiede e chi eroga il servizio; inoltre in ambedue le tipologie si evidenzia l'impossibilità di tracciare lo stato del servizio: si parla quindi di processi stateless. Per chiarire meglio il concetto di orchestrazione può essere effettuato un paragone con il concetto di workflow. La definizione di un workflow, infatti, si traduce nella definizione del flusso di lavoro che effettua operazioni passando da un task all'altro, gestendo stati e risultati; d'altro canto, la definizione di un processo orchestrato significa introdurre un elemento centrale nel processo, che possiede il controllo del flusso che circola tra i servizi, che quindi possono essere assimilati ai task dello stesso workflow.

Il linguaggio BPEL è stato riconosciuto come standard per l'orchestrazione di servizi web, quindi per la definizione dei processi di business: esso, infatti, mette a disposizione diverse funzioni per l'elaborazione dei dati ricevuti dai web service partner del processo, e, tramite funzioni X-Path, esegue operazioni anche complesse su di essi.

L'importanza del linguaggio BPEL come strumento per l'orchestrazione dei web service si evidenzia principalmente effettuando un parallelo con la specifica da cui deriva, ossia lo standard WSDL. Infatti, la specifica WSDL permette di definire l'interfaccia pubblica dei servizi web, quindi la modalità con cui effettuare operazioni con il servizio stesso; ma, nella definizione di questa interfaccia vi è la completa assenza del concetto di stato, che

porta quindi a definire l'interazione con i servizi web come una pura interazione Client-Server, che rende impossibile la creazione di processi di business complessi. Il linguaggio BPEL, invece, sopperisce a questa mancanza fornendo la possibilità di definire variabili che persistono durante tutta l'esecuzione del processo: questa caratteristica è realizzata rendendo anche il processo BPEL un web service, che interagisce con l'esterno tramite un'interfaccia WSDL, che presenta dei tag particolari per la concretizzazione del servizio. Si evidenzia quindi la natura ricorsiva di BPEL: BPEL consente, cioè di modellare le collaborazioni tra servizi esterni o interni, ma il processo da lui gestito viene anch'esso visto dall'esterno come un servizio Web. In questo modo si hanno notevoli vantaggi, prima fra tutte l'estrema modularità con cui possono essere costruiti diversi processi a partire dalle medesime attività atomiche.Un'altra caratteristica fondamentale è che BPEL è sostanzialmente un'applicazione XML-based. Questa peculiarità gli permette di integrarsi e collaborare con altre applicazioni basate su XML; inoltre, risulta necessaria per l'adempimento della principale funzione di questo linguaggio, ossia l'orchestrazione dei web service. Questi ultimi, infatti, come già detto, basano la loro comunicazione sullo standard WSDL, che è anch'esso XML-based, per cui la comunicazione risulta più agevole se fondata su standard comuni ad entrambi i partecipanti. É importante sottolineare inoltre come BPEL si distanzi ancora dallo standard WSDL nella gestione di interazioni in modalità sincrona e asincrona, cosa che invece i servizi web semplici non rendono possibile. In particolare, la modalità asincrona prevede che il chiamante effettui la chiamata senza allocare risorse condivise (e per questo viene anche chiamata shared nothing), e successivamente continui a eseguire il proprio lavoro senza attendere la risposta del chiamato. Una volta che quest'ultimo abbia terminato il proprio processo, esso può restituire il risultato delle proprie operazioni a BPEL.

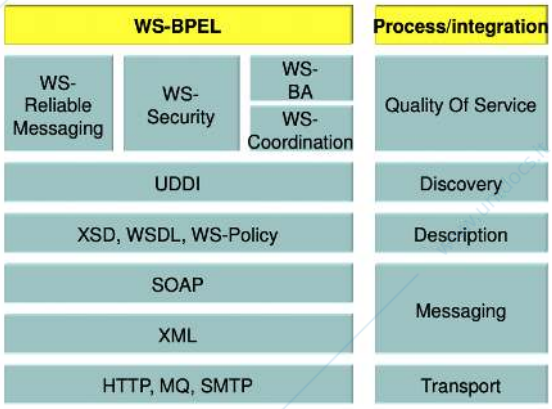

Figura 68: Stack of Web Service

Gli elementi fondamentali del linguaggio WS-BPEL sono equivalenti alle dichiarazioni in un normale linguaggio di programmazione. Definiscono il modo in cui i servizi devono essere chiamati, quali dati devono essere utilizzati e quali dati devono essere trattati in maniera stateful. Questi elementi stabiliscono cosa fa il processo, come reagisce in circostanze diverse (errori, messaggi in arrivo, eventi, ecc.) e come i dati passano da un passaggio all'altro. Gli elementi fondamentali sono:

- Partners. Il concetto di partner viene utilizzato per definire i servizi Web che devono essere richiamati come parte del processo. Ogni servizio invocato dal processo è dichiarato mediante un partnerLink all'interno di BPEL. Si basa su tre elementi:

- Tipo di interazione partner (Partner Link Type): contiene due PortTypes (WSDL), uno per ciascuno dei ruoli nella voce del partner (ovvero, un PortType appartiene al processo stesso, l'altro è il tipo di porta del servizio richiamato). I tipi di collegamento dei partner non vengono archiviati in un processo, ma di solito estendono un documento WSDL.
- Partner Link: il link effettivo al servizio. É qui che viene effettuata l'assegnazione effettiva a un binding (al di fuori dell'ambito di BPEL). Più collegamenti di partner possono condividere lo stesso tipo di collegamento di partner.
- Partners: un gruppo di Partner Link (questo è un elemento opzionale). Un collegamento partner può apparire solo in un partner.

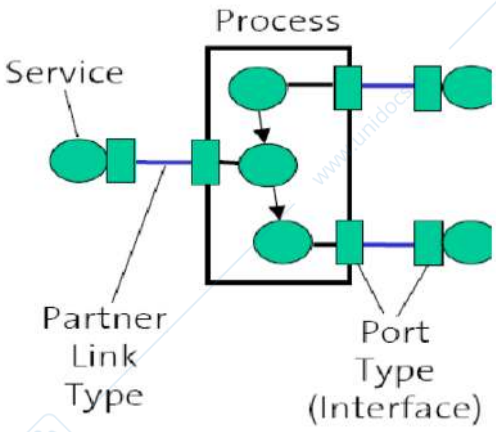

Figura 69: Partner Link Types

- Properties. Le proprietà forniscono una definizione globale e astratta di elementi di dati che devono essere utilizzati per correlare i messaggi con le istanze del processo. (ad es. numeri d'ordine). Gli alias delle proprietà associano tali proprietà a tipi di messaggio specifici. Ciò garantisce che la stessa proprietà possa essere riutilizzata tra messaggi diversi. Gli insiemi di correlazione sono un gruppo denominato di proprietà utilizzato per identificare in modo univoco una conversazione con stato gestita da un processo. Definiscono una mappatura tra dati, messaggi e proprietà che aiutano un'istanza di processo a identificare i messaggi che devono essere gestiti da soli.

- Correlation sets. La correlazione viene utilizzata quando si ricevono messaggi asincroni. I set di correlazione sono composti da attività che comportano lo scambio di un messaggio con partner esterni. Il contenuto di ciascun messaggio viene verificato rispetto all'insieme di correlazione per stabilire il collegamento tra il messaggio e l'istanza di processo corrispondente. Una set di correlazioni viene inizializzato dopo lo scambio del primo messaggio della conversazione e non può essere modificato durante il resto dell'interazione. Le proprietà di correlazione devono essere impostate su valori univoci tra tutte le istanze del processo.

- Variables Scopes. Il linguaggio BPEL permette di dichiarare variabili (elemento variables) da utilizzare all'interno del processo per scambiare o generare informazioni. Le variabili possono essere utilizzate per memorizzare:

  - il contenuto dei messaggi SOAP che vengono scambiati con i partner (con i tipi di messaggio definiti come parte della descrizione dell'interfaccia WSDL)
  - dati intermedi e temporanei utilizzati nella logica aziendale del processo per generare messaggi
  - dati privati che contengono lo stato interno del processo (ad es. Contatori)

  Le variabili sono referenziate da attività che scambiano messaggi per definire da dove leggere il contenuto di un messaggio e dove il messaggio ricevuto dovrebbe essere memorizzato. Le variabili sono comunemente coinvolte in espressioni o query. BPEL permette di specificare qualsiasi linguaggio per questo scopo, ma il suo binding di default è con XPath 1.0, che è quindi il principale formalismo utilizzato per eseguire calcoli e manipolazioni di dati all'interno delle specifiche BPEL correnti. L'intero contenuto di variabili (o solo parti) può anche essere copiato da una variabile diversa con un'attività <assign>. L'attività <assign> può anche essere utilizzata per assegnare valori costanti e applicare XPathqueries ai messaggi al fine di estrarre informazioni pertinenti.

- Fault Handlers: una forma di gestione delle eccezioni basata su blocchi che utilizza i blocchi <catch> per gestire la ricezione di messaggi di errore o eccezioni esplicite (<throw>). Una volta rilevata un'eccezione, la logica necessaria per correggere gli errori può essere programmata nel processo aziendale. I gestori di errori sono un componente di base per la tolleranza agli errori e svolgono lo stesso ruolo dei gestori di eccezioni nei normali linguaggi di programmazione. Ogni processo deve avere questo tipo di gestori.

- Compensation Handlers: inoltre, se si verifica un errore, le attività già completate (e correttamente completate) possono essere annullate utilizzando le funzionalità avanzate di rollback di BPEL. I Compensation Handler assumono che il processo contenga l'intera logica di ciò che deve essere fatto, cosa che raramente è vera in processi complessi. Quindi, essi definiscono le attività di "rollback" o "undo" per lo scope in cui sono inseriti. Solitamente, questi vengono chiamati da un fault handler o direttamente da un'invocazione di servizio, nel caso questa fallisca. Ad esempio, un processo di gestione degli ordini addebita al cliente l'utilizzo di un servizio di carta di credito e rimuove l'articolo ordinato dal servizio di inventario. Se uno dei due servizi fallisce, l'altro deve essere compensato (Figura 70

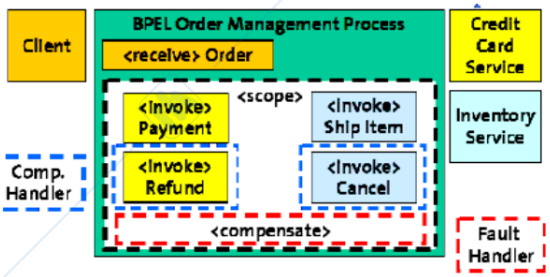

Figura 70: Compensation Handling

- Event Handlers

- Activities:
  - Simple activities: Le operazioni effettive completate dal processo:
    * <receive> si blocca fino alla ricezione di un messaggio
    * <reply> invia un messaggio in risposta a un messaggio ricevuto
    * <invoke> invia un messaggio per richiamare un'operazione remota
    * <assign> aggiorna il valore delle variabili
    * <wait> sospende l'esecuzione per un determinato periodo di tempo
    * <empty> no-op utilizzato a fini di sincronizzazione
    * <terminate> termina il processo
    * <throw>, <rethrow> genera un errore per la cattura di un gestore errori
    * <catch>, <catchAll> rileva un errore di un determinato tipo
    * <compensate> annulla gli effetti delle attività completate
  - Structured activities: definiscono le dipendenze del flusso di controllo in modo gerarchico annidando i seguenti elementi:
    * <sequence> esegue una serie di attività una dopo l'altra
    * <switch> sceglie tra una serie di attività (v1.1)
    * <while> si ripete in base a determinate condizioni
    * <flow> esegue una serie di attività parallele (con dipendenze del flusso di controllo arbitrarie)
    * <pick> attende messaggi alternativi o un allarme e si ramifica in base a quello che è arrivato per primo
    * <scope> definisce un blocco di attività

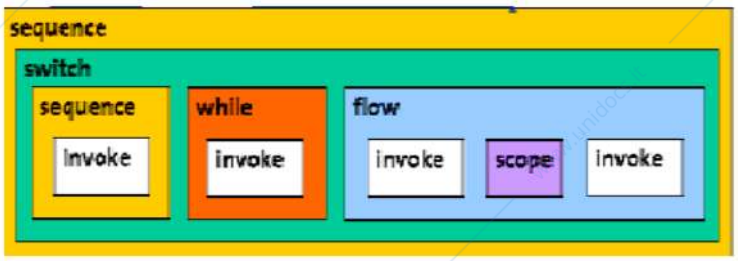

Figura 71: Control Flow

Diverse estensioni sono state proposte per affrontare diversi aspetti che non sono supportati dalla specifica attuale (2.0):

- BPELJ, le cui novità introdotte sono:
  - Gli snippet di codice Java possono essere incorporati nella definizione del processo BPEL
  - Vengono utilizzati per esprimere condizioni di ramificazione (e loop) complesse, inizializzazioni variabili e trasformazione dei messaggi
  - Questa estensione definisce anche come un processo BPEL può chiamare direttamente i componenti J2EE

- BPEL4Sub-Processes, il quale modularizza e riutilizza le definizioni dei processi. Inoltre, definisce come chiamare un processo da un altro.
- BPEL4PEOPLE:
  - Supporto ad attività umane e supporto al flusso di lavoro umano
  - Le attività <invoke> sono taggate con l'elemento <staff> per modellare l'invocazione dei servizi forniti da una risorsa umana
  - Simile ai tradizionali sistemi di gestione del flusso di lavoro, l'invocazione è qualificata descrivendo il ruolo organizzativo dell'utente che interagisce con il processo.

2 Web Service Security

2.1 Auxiliary Protocols

La Figura 72 rappresenta una prima descrizione dei principali meccanismi di sicurezza per Web Services, differenziati in base al livello sul quale operano. Gli standard per la sicurezza dei web service sono suddivisibili in 3 livelli:

- Alto Livello
- A livello di messaggio
- A livello di trasporto

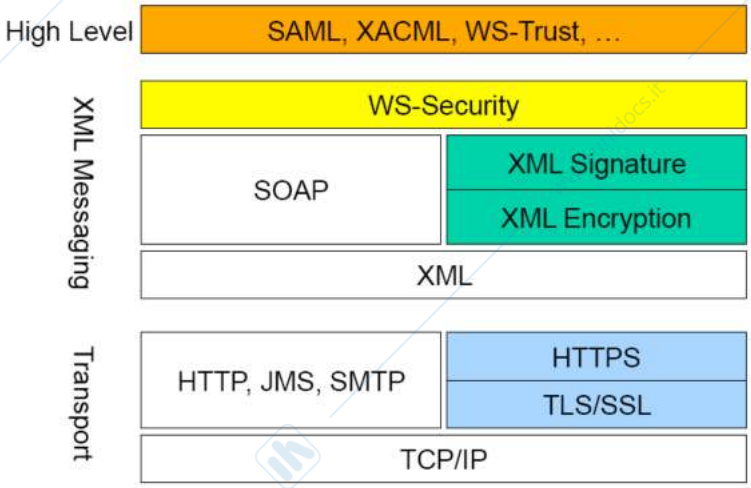

Figura 72: Security Standards

La soluzione più immediata e matura per affrontare i problemi che abbiamo descritto consiste nell'instaurare un meccanismo di sicurezza a livello trasporto. A livello trasporto possiamo contare su protocolli come SSL (Secure Socket Layer) e TLS (Transport Layer Security), in grado di fornire autenticazione, protezione dei dati, supporto a token di crittografia, garantendo integrità e confidenzialità. Come descritto, tramite SSL è possibile

instaurare sicurezza a livello trasporto ma non a livello applicazione. Inoltre, nel caso in cui per i web services ci sia richiesta di supporto per diversi protocolli (FTP, TCP, HTTP..) la sicurezza a livello messaggio non andrà ad impattare sul protocollo, mentre risulta complicato fornire un'adeguata sicurezza a livello trasporto. Quindi, meccanismi di sicurezza a livello trasporto possono non essere sufficienti qualora si operasse in un ambiente popolato da intermediari o pari livello (classici ambienti di un Enterprise Service Bus) che ricevano gli stessi messaggi. Può sorgere inoltre il requisito di agire su una gamma più o meno ampia, ad esempio necessitando di criptare campi diversi con chiavi diverse. In questi casi meccanismi di sicurezza punto-punto non sono più sufficienti e diviene essenziale ricorrere a meccanismi a livello messaggio o applicazione (end-to-end). Da tutte le riflessioni scaturite negli ultimi anni è emerso uno standard emanato dall'OASIS (Organization for the Advancement of Structured Information Standards), il WS-Security, specifica che punta a un arricchimento del messaging SOAP in modo da garantire integrità e confidenzialità ai messaggi inviati, nell'ottica di supportare una grande varietà di modelli di sicurezza (tra i quali PKI, Kerberos, SSL) e tecnologie di firma e criptazione. Il nucleo della specifica è basata sul token, un gettone o simbolo, e la specifica fornisce un meccanismo generico per associare il token al contenuto del messaggio, mantenendosi sufficientemente generica da supportare diversi tipi di formati di tokens. In più, la specifica descrive come criptare tokens e includere chiavi opache (ricevute incapsulate o criptate). In generale quindi lo sforzo dell'OASIS è stato volto a fornire un'estensione al protocollo SOAP che fosse in grado di supportare l'evoluzione dei sistemi di sicurezza. Assieme alla possibilità di inviare tokens di sicurezza come parte del messaggio, gli altri due meccanismi principali sono costituiti dall'integrità del messaggio e dalla confidenzialità. Nessuno dei tre meccanismi preso da solo è in grado di garantire una sicurezza completa per un Web service, ma possono essere utilizzati tanto in modo indipendente quanto accoppiati reciprocamente, ad esempio firmando e criptando parte di un messaggio e fornendo un security token associato alla chiave usata per la firma e la criptazione. Da notare che stiamo parlando solo di un'estensione del messaggio e non del WSDL.

Quindi, la sicurezza dei servizi Web (WSS o WS-Security) descrive i miglioramenti della messaggistica SOAP al fine di fornire qualità di protezione attraverso l'integrità del messaggio e l'autenticazione a singolo messaggio.

Questi meccanismi possono essere utilizzati per soddisfare un'ampia varietà di modelli di sicurezza e tecnologie di crittografia. Alcune delle principali specifiche di sicurezza sono:

- XML Signature (XMLDSIG), la quale garantisce integrità e l'identificazione dei partecipanti alla comunicazione
- XML Encryption (XMLENC), che garantisce la confidenzialità del messaggio
- WS-Security (WSS), che garantisce la protezione del messaggio SOAP
- SAML: Scambio di metadati di sicurezza interoperabile, ossia è un framework per lo scambio di informazioni di autenticazione e autorizzazione.
- XACML: permette il controllo degli accessi.
- WS-Trust e WS-Federation: associazione di più domini di sicurezza
- WS-SecureConversation: protezione di più scambi di messaggi

- WS-SecurityPolicy: descrivere quali funzioni di sicurezza sono supportate o necessarie da un servizio Web
- XrML: gestione dei diritti digitali
- XKMS: gestione e distribuzione delle chiavi

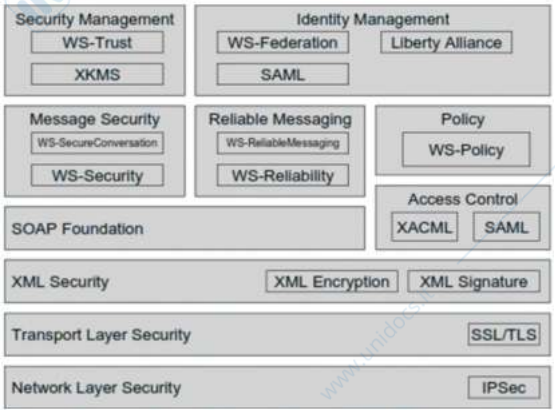

Figura 73: Security Standards Stack

2.1.1 XML Signature

Per autenticare un documento si ricorre alla tecnica della Digital Signature (firma digitale), tramite la quale si ha la possibilità di firmare un documento utilizzando un algoritmo di crittografia a chiave pubblica. In particolare, la firma digitale viene realizzata utilizzando la chiave privata di colui che vuole inviare il documento. In questo modo si è sicuri dell'identità di chi ha firmato il documento perchè la firma digitale può essere decifrata solamente utilizzando la sua chiave pubblica. Per creare la firma digitale non viene cifrato il documento ma il suo digest (sintesi), detto anche hash. Questo viene prodotto da un particolare algoritmo che, ricevendo in input il documento, è in grado di produrne una sintesi, una sorta di "impronta" che lo identifica in maniera univoca. Questa funzione è molto sensibile ad ogni minimo cambiamento del documento e quindi differenti versioni del documento produrranno digest diversi. Il digest è quindi utilizzato per verificare l'integrità del documento. Infatti, una volta decodificata la firma digitale, abbiamo a disposizione il digest del documento che è stato spedito; confrontandolo con il digest calcolato sul documento ricevuto, possiamo stabilire se il documento ha subito delle alterazioni o no.

XML Signature è una tecnologia che permette di "firmare" documenti XML o parti di essi, garantendone l'autenticità e l'integrità. Offre i servizi di Integrità/autenticazione del messaggio (message authentication) e Autenticazione di chi firma (signer authentication). Inoltre, assicura la condizione di non-ripudio. La firma XML prescrive come calcolare, archiviare e verificare la firma digitale di:

- interi documenti XML
- parti di documenti XML
- "tutto ciò a cui è possibile fare riferimento a partire da un URL", inclusi oggetti non XML, come ad esempio Immagini.

XML Signature è uno standard complesso e flessibile: è possibile applicare più firme sullo stesso contenuto XML e supporta una varietà di codici e protocolli di autenticazione. É stato inserito come standard da agosto 2001. Esistono due tipologie di firme:

- Enveloped Signature: la firma è contenuta all'interno dell'elemento firmato.

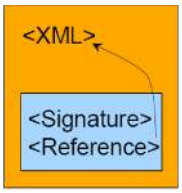

Figura 74: Enveloped Signature

- Enveloping Signature: la firma avvolge l'elemento firmato.

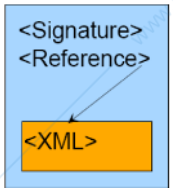

Figura 75: Enveloping Signature

- Detached Signature: la firma si riferisce a un elemento separato (all'interno o all'esterno del documento).

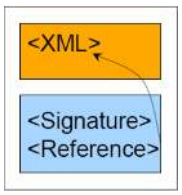

Figura 76: Detached Signature

Vediamo ora, il processo di applicazione ad un data object O:

1. Calcola H = hash(O)
2. Calcola H' = hash([H, t]) , con t dati ulteriori (algoritmo usato, timestamp, URI...)
3. Calcola S = signature(H')
4. Genera l'elemento XML <Signature>...</Signature>

Un esempio di elemento <Signature> è riportato nelle Figure 77 e 78.

Figura 77: XML Signature Structure

```
<Signature>
  <SignedInfo>
   <Reference URI="http://www.google.com"/>
  </SignedInfo>
  <SignatureValue>Base-64 encoded </SignatureValue>
  <KeyInfo>…</KeyInfo>
</Signature>
```

Figura 78: XML Signature Simplified Example

La firma XML è contenuta all'interno dell'elemento <Signature> costituito da tre elementi principali:

- Il primo elemento (<SignedInfo>) racchiude le informazioni su come è stata generata la firma: l'algoritmo utilizzato per ottenere la forma canonica XML, l'algoritmo utilizzato per produrre la firma e le informazioni sulla risorsa firmata. Queste ultime sono specificate all'interno dell'elemento <Reference> e riguardano l'URI del documento XML, l'algoritmo utilizzato per calcolarne il digest e il valore del digest stesso.

- L'elemento <SignatureValue> contiene il valore della firma digitale ottenuta secondo l'algoritmo indicato nell'elemento precedente.

- L'ultimo elemento (<KeyInfo>) contiene le informazioni sulla chiave pubblica di chi ha firmato il documento XML e viene utilizzata per decifrare la firma digitale XML nella fase di verifica dell'autenticità e dell'integrità del documento. In questo caso al suo interno è presente l'elemento <DSAKeyValue> che contiene il valore della chiave pubblica secondo l'algoritmo DSA.

Ora vediamo nel dettaglio il processo per la generazione del campo reference:

1. Dereference <Reference URL> per accedere al contenuto XML che deve essere firmato
2. Applicare le trasformazioni
3. Calcola <DigestValue> applicando <DigestMethod> al contenuto trasformato
4. Memorizzare il risultato nell'elemento <Reference>

Infatti, il documento firmato potrebbe contenere al suo interno dei dati cifrati e questo potrebbe causare delle difficoltà in fase di verifica della firma digitale. Per ovviare a questo problema è possibile utilizzare l'elemento <transforms> all'interno di <Reference>. In questo elemento è possibile specificare alcune trasformazioni che il verificatore della firma digitale deve compiere prima dell'operazione di verifica.

Il processo per la generazione della firma vera e propria:

1. Creare l'elemento <SignedInfo>.
2. Trasformarlo in forma canonica.
3. Calcolare <SignatureValue> applicando <SignatureMethod>.
4. Raggruppare tutto insieme agli elementi <KeyInfo> e <Object>.

Quello che è effettivamente firmato è il <Reference>, che contiene un digest (hash) del contenuto originale, che è solo indirettamente firmato. Quindi, riassumendo, per creare la firma digitale di un documento XML viene utilizzato il suo digest. Abbiamo detto che la funzione utilizzata per calcolarne il valore è molto sensibile alle variazioni del documento e quindi al fine di garantire il corretto funzionamento del meccanismo della firma digitale, il digest deve essere generato utilizzando la forma canonica del documento XML.

Per verificare il reference dobbiamo applicare la seguente procedura:

1. Dereference <Reference URL> per accedere al contenuto XML che deve essere convalidato rispetto al digest
2. Applicare le stesse trasformazioni
3. Calcola un hash usando lo stesso <DigestMethod>
4. Confrontare <DigestValue> con il risultato.

Mentre, per verificare la Signature, dobbiamo:

1. Canonicalizzare l'elemento <SignedInfo>
2. Ottenere la chiave seguendo l'elemento <KeyInfo>
3. Calcolare l'hash con <SignatureMethod>
4. Confrontarlo con <SignatureValue>

L'elemento <Reference> punta alla risorsa che viene firmata digitalmente (attributo URI). Deve esserci almeno un elemento <Reference> (ma è possibile averne diversi nella stessa firma). Alcuni esempi sono:

- Ospita un elemento dello stesso documento URI = "#CustomerInformation"
- La radice dell'URI del documento contenitore = ""
- Un documento XML esterno URI = "http://www.swisscom.ch/order.xml"
- Un frammento di un documento esterno URI = "http://www.swisscom.ch/order.xml#Total"
- Un URI di risorse non XML esterno = "http://www.swisscom.ch/order.pdf"

Un elemento <Reference> contiene un insieme di elementi <transforms>, che vengono applicati in modo pipeline al contenuto della risorsa di riferimento. Le stesse trasformazioni (nello stesso ordine) devono essere utilizzate durante la generazione e la convalida di un digest. Alcuni esempi di operazione di trasformazione sono:

- Trasformazioni standard:
  - Canonizzazione: W3C Recommendation (15 March 2001) specifica un algoritmo (XML canonicalization method (C14N)) che, dato un documento XML in input, genera un secondo documento XML equivalente al primo, ma con una particolare rappresentazione fisica detta "forma canonica". XML definisce un insieme di regole per strutturare l'informazione in formato testuale. Queste regole sono molto libere ed offrono un notevole grado di flessibilità nel senso che la stessa struttura dati o le stesse informazioni possono essere rappresentate in maniera differente. Possono, quindi, sussistere due documenti che sono logicamente equivalenti, visto che rispettano la stessa struttura (definita nel loro Dtd o XML Schema) e contengono gli stessi dati, ma non uguali. Nasce quindi il problema di riuscire a capire in maniera automatica quando due file sono equivalenti pur essendo fisicamente differenti. Per risolvere questo problema è stata ideata la specifica di Canonical XML, che definisce un insieme di regole per scrivere in forma canonica (detta anche simplified -semplificata-) un documento XML. La forma canonica di un documento XML è una sua rappresentazione fisica che risulta essere invariante rispetto alle possibili differenze sintattiche del documento stesso (ad esempio l'ordine degli attributi, la presenza di spazi bianchi, etc.). L'operazione che produce un documento in forma canonica è detta Canonicalizzazione (XML Canonicalization) e può essere realizzata in automatico utilizzando un ambiente di sviluppo XML conforme alle specifiche di Canonical XML. Riassumendo, le firme sono sensibili alle modifiche a bit singolo. I dati XML possono avere serializzazioni multiple (ed equivalenti). La soluzione più essere quella di fornire una procedura precisa (e standard) per la produzione di "stringhe" XML da infoset XML. Questa procedura è chiamata canonicalizzazione sensibile alle modifiche a bit singolo.

    Di seguito, elenchiamo alcune regole per la Canonicalizzazione XML:
    - BusUTF-8encoding
    - Le interruzioni di riga sono normalizzate su LF (ASCII #xA)
    - I riferimenti a caratteri ed entità vengono sostituiti
    - Le sezioni CDATA vengono sostituite con il loro contenuto
    - La dichiarazione XML e la definizione DTD vengono rimosse
    - Gli elementi <Empty/> sono convertiti in <Empty> </Empty>

* I delimitatori del valore di attributo sono impostati tra virgolette doppie
* Le dichiarazioni superflue vengono rimosse
* Gli attributi predefiniti sono esplicitamente aggiunti agli elementi
* Le dichiarazioni dello spazio dei nomi sono ordinate prima degli attributi (anch'essi ordinati)

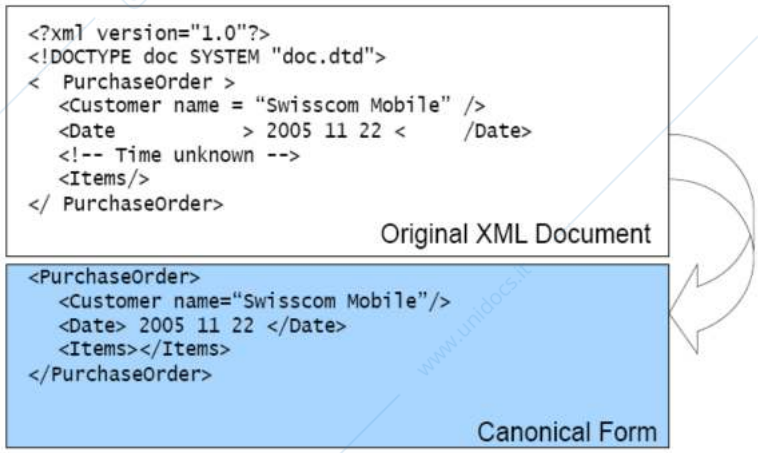

Figura 79: Canonicalization Example

- Enveloped Signature Transform: questa firma è necessaria per firmare un elemento che è il genitore della <Signature> (altrimenti, la firma dovrebbe essere usata come input per calcolare se stessa, il che rende impossibile il calcolo). Questa trasformazione rimuove semplicemente l'elemento <Signature> dal documento.

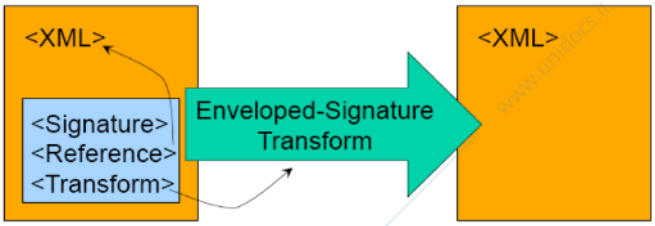

Figura 80: Enveloped Signature Transform Example

- Decrypt Transofrm

- Trasformazioni opzionali:
  - Base-64
  - XPath Filtering
  - XSLT Transform

Questi elementi descrivono come è stata calcolata una firma e ne memorizzano il valore in formato codificato:

- <DigestValue> contiene il valore codificato Base-64 del digest
- <SignatureValue> contiene il valore codificato Base-64 risultante dalla crittografia del digest dell'elemento <SignatureInfo> con la chiave descritta in <KeyInfo>
- <DigestMethod> descrive l'algoritmo utilizzato per calcolare <DigestValue> (ad es. SHA1)
- <SignatureMethod> descrive come è stato calcolato <SignatureValue> (ad es. RSA-SHA1) usando la chiave

<KeyInfo> fornisce informazioni sulla chiave utilizzata per convalidare <SignatureValue>. É abbastanza flessibile:

- L'elemento può essere omesso (le parti che scambiano il messaggio concordano sulla chiave usando un meccanismo fuori banda)
- La chiave è incorporata nel messaggio
- La chiave è referenziata dal messaggio
- Supporta diversi tipi di chiavi utilizzate con diversi standard crittografici:
  - TheDSA / RSA
  - Certificati X.509
  - PGP

Lo stesso elemento viene utilizzato nella crittografia XML.
La firma XML si rivolge a questi aspetti di sicurezza:

- Integrità del contenuto del messaggio / risorsa esterna: Convalida di Reference
- Integrità della firma: Convalida della firma
- Identità della fonte del documento: Convalida della firma. Questo aspetto è soddisfatto solo se si utilizza un <SignatureMethod> basato su chiave pubblica/privata.

Quello che vedi è ciò che firmi: le trasformazioni modificano e filtrano i dati prima che vengano firmati, quindi devono essere utilizzati con attenzione.

2.1.2 XML Encryption

Grazie alla sua caratteristiche di flessibilità ed espandibilità, XML è molto utilizzato come formato di scambi dati tra applicazioni. In molti ambiti (commerciale, militare, etc.) risulta molto importante avere a disposizione uno strumento in grado di assicurare la protezione dei dati che vengono scambiati tra le applicazioni. Attualmente lo strumento maggiormente utilizzato per garantire la sicurezza in ambito è SSL (Secure Socket Layer) che permette di creare un canale protetto per lo scambio di dati tra due applicazioni. Si applica a qualsiasi tipo di dati e compresi interi documenti XML (trattati come octet-stream). In XML il problema della sicurezza è affrontato dalla specifica di XML Encryption, una tecnologia in grado di proteggere un documento XML o parti di esso. XML Encription permette di cifrare gli elementi di un documento XML utilizzando i più diffusi algoritmi di crittografia, sia a chiave simmetrica che a chiave pubblica. Quindi

l'obiettivo di XML Encryption è assicurare la confidenzialità dei messaggi XML. Questo è possbile, offuscando parti di un documento XML seppur mantenendo una corretta sintassi XML. La tecnologia Encryption è esposta nel W3C Recommendation di dicembre 2002. Le caratteristiche dell'XML Encryption sono:

- End to End (scenario multi-hop)
- Crittografia completa o parziale
- Flessibilità: parti diverse di un messaggio possono essere lette da parti diverse utilizzando chiavi diverse

Alcune sfide e problemi, derivanti dall'utilizzo di XML Encryption, sono:

- Un documento XML crittografato è ancora XML?
- Come convalidare un documento XML crittografato rispetto al suo schema XML?

La tecnologia XML Encryption è complementare a XML Signature. Hanno funzioni differenti: XML signature assicura integrità e identità delle parti coinvolte; mentre Encryption garantisce la confidenzialità. Come dicevamo prima, condividono alcune specifiche come <KeyInfo>. Un XML crittografato è sostituito dall'elemento <EncryptedData>, mentre l'XML firmato fa riferimento dall'elemento <Signature>. I dati crittografati non firmati possono ancora essere manomessi. Quindi, XML Encryption garantisce la riservatezza a livello di messaggio SOAP (le parti selezionate possono accedere a diverse parti del messaggio).

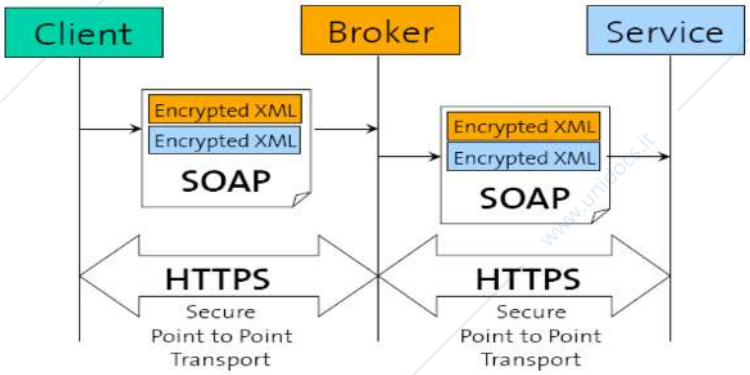

Figura 81: XML Encryption Scenario

La Figura 82 rappresenta un esempio di crittografia XML.
Il risultato è l'elemento <EncryptedData>...</EncryptedData> che contiene:

- Un riferimento ai dati criptati oppure i dati criptati stessi (elemento CipherData con contenuto testuale dei dati in formato base64)
- Le informazioni sull'algoritmo di cifratura utilizzato ed eventualmente, sulla chiave

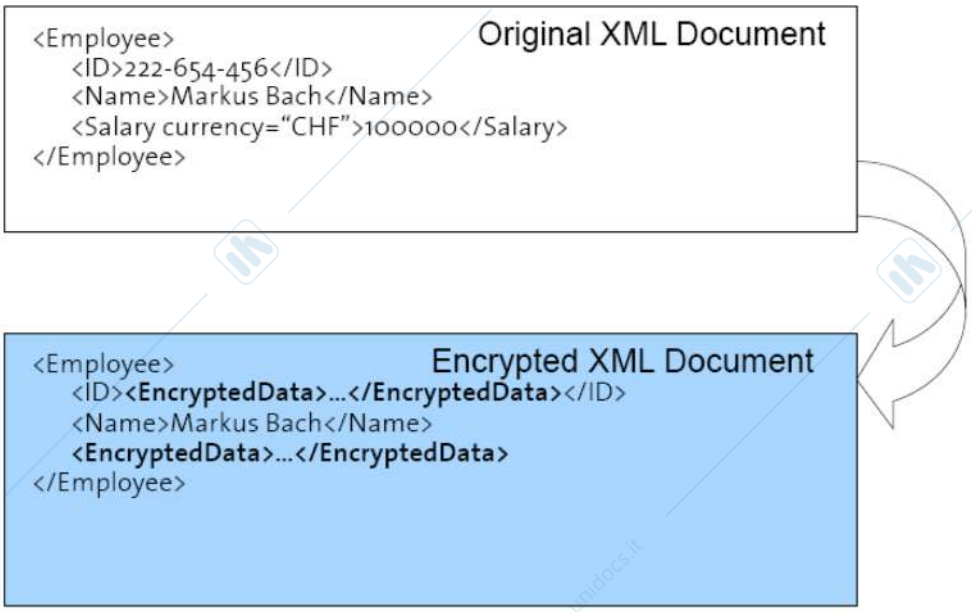

Figura 82: XML Encryption Example

La Figura 83 raffigura la struttura di un file XML crittato.

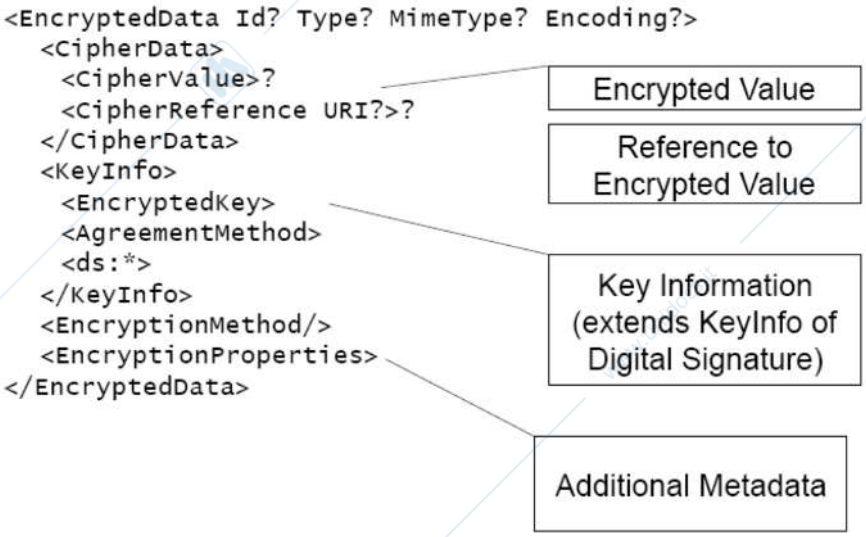

Figura 83: XML Encryption Structure

Il tag contenitore <EncryptedData> sostituisce gli elementi del documento che vengono inviati in forma crittografata. L'elemento <CipherData> racchiude i dati codificati. Al suo interno può essere presente l'elemento <CipherValue> (come in questo caso), che contiene il dato cifrato rappresentato da una sequenza di ottetti in base64, oppure l'elemento <CipherReference>, che indica l'URI contenente il dato cifrato. Insieme agli elementi crittografati, <EncryptedData> contiene metadati e attributi che descrivono come decrittografarli: <EncryptionMethod>, <KeyInfo>. Gli attributi del tag <EncryptedData> sono:

- Tipo = (elemento | contenuto). L'attributo Type permette di specificare quale parte del documento viene cifrata: un solo elemento (#Element), come in questo caso, o un gruppo di elementi (#Content).
- MimeType. Attributo facoltativo che descrive il tipo di elemento non XML crittografato
- Codifica. Come è stato codificato il non XML

Il tag <EncryptionMethod> specifica quale algoritmo è stato utilizzato per crittografare i dati. I metodi attualmente supportati sono:

- Triple-DES
- AES (Advanced Encryption Standard) con chiave a 128, 256 (richiesto) o 192 (opzionale)

L'elemento <CipherData> è obbligatorio e racchiude i dati codificati. Al suo interno. può essere presente l'elemento <CipherValue> (come in questo caso), che contiene il dato cifrato rappresentato da una sequenza di ottetti in base64, oppure l'elemento <CipherReference>, che indica l'URI contenente il dato cifrato. Quindi, <CipherData> memorizza o fa riferimento ai dati crittografati:

- Contenitore <CipherValue> per dati binari crittografati
- <CipherReference> riferimento a un URL dei dati crittografati. Può includere una pipeline di elementi Transform come XML Signature, che specifica come filtrare i dati di riferimento prima che vengano decrittografati

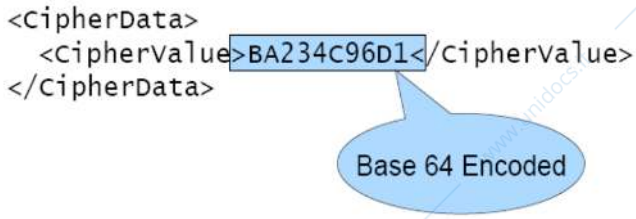

Figura 84: <CipherData>

Le informazioni sulla chiave da utilizzare sono contenute all'interno dell'elemento <ds:KeyInfo>. In questo esempio l'elemento <ds:KeyName> indica l'identificativo della chiave da utilizzare per decifrare il dato. Quindi, <KeyInfo> descrive la chiave utilizzata per crittografare i dati. Mentre in XML Signature, di solito si tratta di una chiave pubblica, nella crittografia XML si tratta generalmente di una chiave di crittografia condivisa. In generale, le chiavi pubbliche possono essere incluse in modo sicuro in un messaggio; mentre, non è sicuro incorporare le chiavi condivise. La crittografia XML offre diversi meccanismi per concordare/recuperare la chiave di decrittazione:

- Chiave omessa (fuori banda)

- Si fa riferimento alla chiave: <KeyName> <RetrievalMethod>. Questi elementi sono usati per identificare quale delle chiavi segrete (condivise tra le parti) dovrebbe essere usata e come la chiave condivisa dovrebbe essere recuperata. Con essi, la stessa chiave può essere utilizzata per crittografare diverse parti dello stesso documento.
- La chiave viene rigenerata: <AgreementMethod>
- La chiave è inclusa in forma crittografata: <EncryptedKey>

É possibile riutilizzare lo stesso elemento <EncryptedKey> per decrittografare più elementi <EncryptedData>.

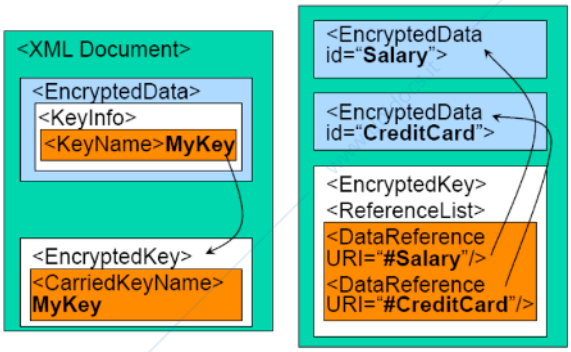

Figura 85: Condividere le chiavi all'interno dello stesso messaggio

Il processo di Encryption è il seguente:

1. Scegliere un algoritmo (3DES, AES)
2. Scegliere una chiave e definire come rappresentarla:
   - La chiave viene generata o cercata
   - La chiave è stata omessa dal messaggio
   - La chiave è descritta nella sezione <KeyInfo>
3. Serializzare i dati XML su un flusso di byte
   - Elemento (con tag)
   - Contenuto (tag omesso)
4. Crittografare il flusso di byte
5. Codificare il risultato nell'elemento <CipherData>
6. Creare l'elemento <EncryptedData> con le informazioni richieste per decrittografarlo.

Mentre, il processo di Decryption è il seguente:

1. Determinare l'algoritmo (3DES, AES).

2. Determinare la chiave: chiave e algoritmo potrebbero essere concordati in anticipo. Invece, se la chiave è crittografata, decodificala (questo è ricorsivo).

3. Chiave di decodifica:

   - CipherValue (decodifica il flusso di Base-64 byte incorporato)
   - CipherReference (dereference l'URI e applica le Transform specificate prima che i dati vengano decifrati)

4. Elaborare il contenuto XML: analizzare l'XML serializzato e sostituire l'elemento <EncryptedData> originale con l'elemento (o il contenuto) XML decrittografato.

5. Elaborare il contenuto non XML descritto dagli attributi MimeType e Encoding dell'elemento <EncryptedData>.

Riservatezza e integrità dei messaggi sono entrambi requisiti importanti per uno scambio di messaggi sicuro. Signature XML e Encryption XML sono state progettate per lavorare insieme per raggiungere questo obiettivo. Il problema che sorge dall'utilizzo complementare delle due tecniche è stabilire in quale ordine devono essere applicati? Firmare o crittografare prima? Una soluzione che può essere adottare è nascondere la firma all'interno dell'XML crittografato. L'ordine per risalire al messaggio e verificarlo è:

1. Decrittografia;

2. Verifica della firma.

Il problema è che i metadati di encryption non sono protetti con una firma. Infatti, se non firmato, i dati/metadati crittografati potrebbero essere danneggiati da un utente malintenzionato per impedire la decrittografia del messaggio.

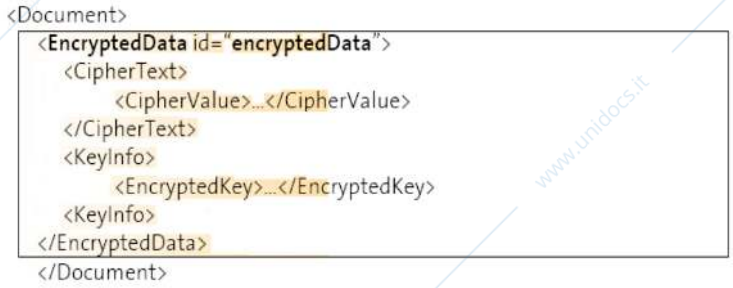

Figura 86: Crittografare i dati firmati

Invece, se le firme vengono inviate in chiaro, gli aggressori potrebbero rimuoverle da un messaggio o sostituirle completamente senza che il destinatario se ne accorga.

Quando viene ricevuto un messaggio, potrebbe non essere chiaro in quale ordine applicare la convalida e la decrittografia della firma. Per rendere esplicito l'ordine di crittografia e firma, la voce Decrypt Transforms è stata aggiunta allo standard di XML Signature. Questa trasformazione viene utilizzata per distinguere se la firma si applica a <EncryptedData> o ai dati decrittografati. Il processore di XML Signature decodificherà tutti gli elementi <EncryptedData> di riferimento tranne quello identificato dall'elemento <Except>. Con questa soluzione, l'elaborazione predefinita applica sempre la decrittografia prima della verifica della firma; a meno che tale trasformazione non sia specificata dal mittente.

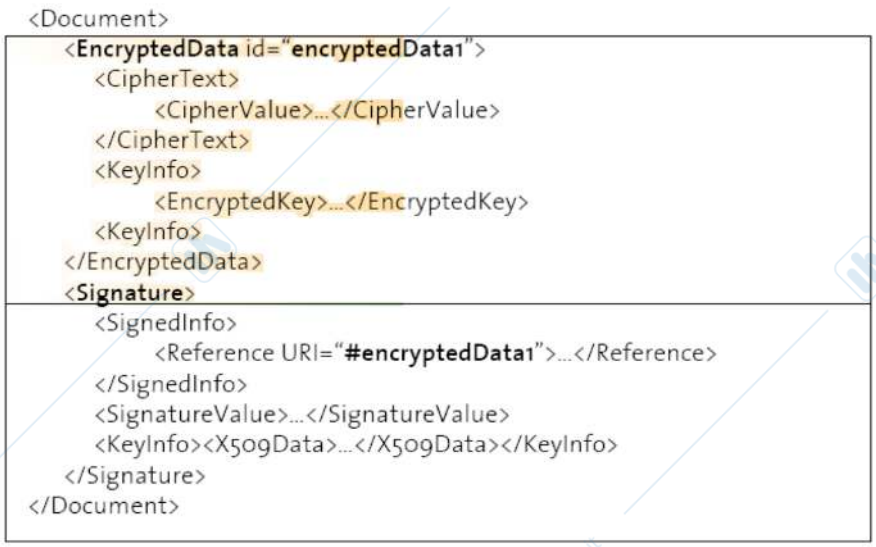

Figura 87: Firmare i dati crittografati

2.1.3 Messaggi SOAP sicuri

Lo standard WS-Security applica la sicurezza XML (Encryption XML e Signature XML) per implementare lo scambio di messaggi SOAP sicuri su domini di fiducia multipli e indipendenti. L'obiettivo è avere sicurezza a livello di messaggio (end-to-end). La soluzione è applicare la crittografia e le firme all'interno di un messaggio SOAP indipendente dal trasporto. Parti del corpo del messaggio possono essere crittografate, le firme sono memorizzate nell'intestazione. WS-Security offre supporto per:

- più tecnologie di firma
- più tecnologie di crittografia
- più formati di token di sicurezza

Vediamo a confronto la sicurezza a livello di messaggio con la sicurezza a livello di trasporto:

- Message Security:
  - Vantaggi:
    - Diverse parti di un messaggio possono essere protette in diversi modi.
    - Asimmetrico: diversi meccanismi di sicurezza possono essere applicati a richiesta e risposta
    - Messaggi autoprotettivi (trasporto indipendente)
  - Svantaggi:
    - Standard immaturi supportati solo parzialmente dagli strumenti esistenti
    - La protezione di XML è complicata
- Transport Security
  - Vantaggi:

    - Tecnologie ampiamente disponibili e mature (SSL, TLS, HTTPS)
    - Compresi dalla maggior parte degli amministratori di sistema
  - Svantaggi:
    - Point 2 Point: il messaggio completo è in chiaro dopo ogni salto
    - Simmetrico: i messaggi di richiesta e risposta devono utilizzare le stesse proprietà di sicurezza
    - Trasporto specifico

Alcune minacce alla sicurezza di un messaggio SOAP sono:

- un messaggio potrebbe essere letto da un utente malintenzionato
- un messaggio potrebbe essere modificato da un utente malintenzionato
- un messaggio potrebbe essere inviato da un utente malintenzionato

Per far fronte a queste minacce, WS-Security applica una combinazione di:

- Encryption (garantire la riservatezza del messaggio)
- Signatures (verifica l'origine e l'integrità di un messaggio)
- Token di sicurezza (autorizzano l'elaborazione del messaggio in base alle credenziali associate al messaggio)

I messaggi con firme non valide e token errati o mancanti vengono rifiutati.

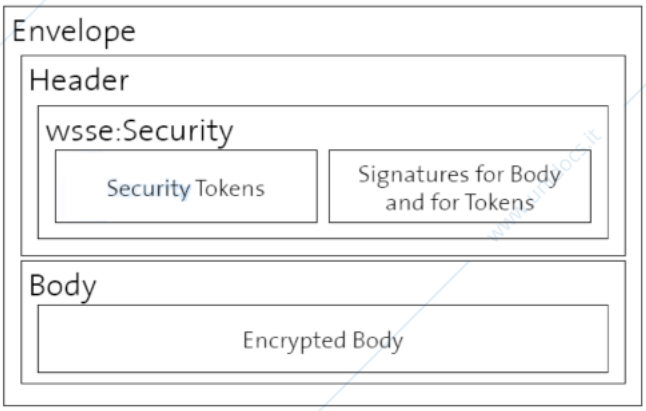

Figura 88: Messaggio SOAP sicuro

2.1.4 Security Tokens

Un token di sicurezza è un dispositivo periferico utilizzato per accedere a una risorsa limitata elettronicamente. Il token viene utilizzato in aggiunta o al posto di una password. Funziona come una chiave elettronica per accedere a qualcosa. WS-Security supporta numerosi meccanismi di autenticazione e autorizzazione includendo i token corrispondenti nell'intestazione Security del messaggio:

- Token semplici:

  - Nome utente - Password in chiaro
  - Nome utente - Password Digest
- Token binari:
  - Certificati X.509
  - Kerberos[3]
- XML Tokens
  - SAML assertions
  - XrML (eXtensible Rights Markup Language)
  - XCBF (XML Common Biometric Format)
- Token reference (WS-SecureConversation)

Un token di sicurezza può essere utilizzato per rivendicare l'identità della fonte di un messaggio. Nome utente / PasswordText è il token più semplice utilizzato per comunicare l'identificazione ma non è anche sicuro (i messaggi SOAP non devono contenere password in chiaro). Username / PasswordDigest risolve questo problema (Figura 89).

```
<UsernameToken>
    <Username>Scott Tiger</Username>
    <Password Type="PasswordDigest">XYZAAAq</Password>
    <Nonce>123521</Nonce>
    <Created>2005-11-24T15:00:00Z</Created>
</UsernameToken>
```

Figura 89: Username/PasswordDigest

Per produrre il digest, la password viene hashata con un timestamp e un nonce. Questa tecnica protegge dagli attacchi reply. Il server deve memorizzare la password di testo normale.

Inoltre, è possibile firmare un token di sicurezza per autenticare una richiesta presentata dal mittente del messaggio. Le firme associate ai token possono essere verificate dal destinatario per autenticare l'identità del mittente. Ad esempio, come mostrato in Figura 90, i certificati X509 (chiavi pubbliche) devono essere firmati per fornire l'autenticazione del mittente (prova del possesso della chiave privata corrispondente).

Figura 90: Token di sicurezza e autenticazione

[3]Kerberos è un protocollo di rete per l'autenticazione tramite crittografia che permette a diversi terminali di comunicare su una rete informatica insicura provando la propria identità e cifrando i dati.

Sistemi diversi possono appartenere a domini di sicurezza diversi, che utilizzano meccanismi e criteri di sicurezza diversi. Sebbene SOAP consenta l'interoperabilità tra questi sistemi, la traduzione di metadati di sicurezza tra domini diversi rimane un problema. WS-Security è un primo passo per fornire sintassi e semantica standardizzate per rappresentare le informazioni di sicurezza. WS-Trust aggiunge un'interfaccia standard per un provider di servizi token di sicurezza utilizzato per:

- Emettere e rinnovare token di sicurezza da allegare a un messaggio SOAP con WS-Security.
- Convalidare token di sicurezza da un dominio diverso.
- Tradurre token di sicurezza tra domini che condividono una relazione di trust (WS-Federation).

Quindi, WS-Trust è una specifica WS- * e standard OASIS che fornisce estensioni a WS-Security, occupandosi in particolare dell'emissione, del rinnovo e della convalida dei token di sicurezza, nonché dei modi per stabilire, valutare la presenza e la fiducia del broker relazioni tra partecipanti in uno scambio di messaggi sicuro.

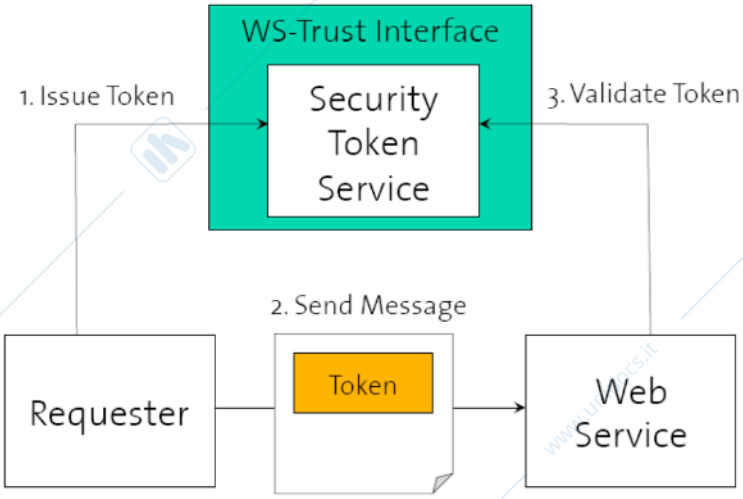

Figura 91: Ws-Trust

WS-SecureConversation è una specifica dei servizi Web, creata da IBM e altri, che funziona in collaborazione con WS-Security, WS-Trust e WS-Policy per consentire la creazione e la condivisione di contesti di sicurezza. Estendendo i casi d'uso di WS-Security, lo scopo di WS-SecureConversation è stabilire contesti di sicurezza per più scambi di messaggi SOAP, riducendo il sovraccarico dell'istituzione delle chiavi. L'handshake di sicurezza che comporta la creazione di token e la loro convalida può imporre un sovraccarico ad alte prestazioni. WS-SecureConversation definisce un contesto di sicurezza condiviso da riutilizzare nello scambio di più messaggi. É possibile riutilizzare la stessa combinazione di credenziali di sicurezza (autenticazione, autorizzazione) e chiavi di crittografia. Una volta stabilita la conversazione, il richiedente e il servizio condividono un segreto: il client non deve includere i metadati di sicurezza per ciascun messaggio e il servizio non deve

riconvalidare gli stessi token per ciascun messaggio. Questo è implementato usando un token speciale: <SecurityContextToken>.

WS-Policy è una specifica che consente ai servizi Web di utilizzare XML per pubblicizzare le proprie politiche (in materia di sicurezza, qualità del servizio, ecc.) e per i consumatori di servizi Web di specificare i propri requisiti delle politiche. WS-Policy è una raccomandazione del W3C a settembre 2007. WS-Policy rappresenta un insieme di specifiche che descrivono le capacità e i vincoli delle politiche di sicurezza (e di altre attività) su intermediari ed end point (ad esempio token di sicurezza richiesti, algoritmi di crittografia supportati e regole sulla privacy) e su come associare le politiche a servizi e punti finali.

2.1.5 SAML (Security Assertion Markup Language)

Security Assertion Markup Language (SAML) è uno standard informatico per lo scambio di dati di autenticazione e autorizzazione (dette asserzioni) tra domini di sicurezza distinti, tipicamente un identity provider (entità che fornisce informazioni di identità) e un service provider (entità che fornisce servizi). Il formato delle asserzioni SAML è basato su XML. SAML è mantenuto da OASIS Security Services Technical Committee. Security Assertion Markup Language (SAML) è precedente a WS-Security, poiché è stato standardizzato presso OASIS nel novembre 2002 (v1.0), agosto 2003 (v1.1), marzo 2005 (v2.0).

L'obiettivo di SAML è abilitare la gestione delle identità vagamente accoppiata. La soluzione consiste nel definire un formato e un protocollo per lo scambio interoperabile di informazioni sulla sicurezza (o asserzioni) su argomenti (utenti umani o sistemi informatici) che devono essere identificati all'interno di un determinato dominio di sicurezza.

SAML richiede che l'utente (detto "principal") sia registrato presso almeno un identity provider. L'identity provider deve provvedere ad autenticare l'utente. Il service provider a questo punto si affida all'identity provider per identificare il principal. Su richiesta del principal l'identity provider passa una asserzione SAML al service provider sulla base della quale quest'ultimo decide se permettere o negare l'accesso ai propri servizi da parte del principal.

I casi d'uso supportati da profili standard sono:

- Single Sign On (SSO) e Single Logout
- Federazione delle identità (Identity Federation)[4]
- Identificazione nel rispetto della privacy
- Protezione dei messaggi dei servizi Web: le asserzioni SAML vengono utilizzate come token WS-Security.

SAML definisce anche il protocollo per i client per richiedere asserzioni dalle "autorità SAML" e per i servizi per verificare le asserzioni con "autorità SAML" affidabili.

Il problema principale che SAML cerca di risolvere è quello del Web Single sign-on (SSO) tra entità appartenenti a organizzazioni e domini di sicurezza distinti. Il Single sign-on (SSO) è la proprietà di un sistema di controllo d'accesso che consente ad un utente

[4]Federation è un nuovo approccio, anche per applicazioni web, che usa un protocollo basato su degli standard che permette a una applicazione di accertare una sola volta l'identità di un utente di servizi diversi, in modo tale da evitare il bisogno di autenticazioni ridondanti. Gli standard per supportare Federation includono SAML e WS-Federation.

di effettuare un'unica autenticazione valida per più sistemi software o risorse informatiche alle quali è abilitato. SAML abilita Single Sign-On e il trasferimento di credenziali di identità tra domini di fiducia diversi. Le credenziali stabilite nel servizio iniziale, in cui l'utente è autenticato, vengono inoltrate ad altri servizi che possono fidarsi di loro. Questo viene fatto senza un registro di autenticazione centralizzato che dovrebbe essere condiviso e considerato affidabile da tutti (esempio: Project Liberty).

Figura 92: Identità portatile e federata

Con questo approccio differenti gestori ("federati" tra loro) gestiscono dati di uno stesso utente. L'accesso ad uno dei sistemi federati permette automaticamente l'accesso a tutti gli altri sistemi. Un viaggiatore potrebbe essere sia passeggero di un aereo che ospite di un albergo. Se la compagnia aerea e l'albergo usassero un approccio federativo, potrebbero stipulare un accordo reciproco sull'autenticazione dell'utente finale. Il viaggiatore potrebbe ad esempio autenticarsi per prenotare il volo ed essere autorizzato, in forza di quella sola autenticazione, ad effettuare la prenotazione della camera d'albergo. Questo approccio è stato sviluppato per rispondere ad un bisogno di gestione decentralizzata degli utenti: ogni gestore federato mantiene il controllo della propria politica di sicurezza.

SAML utilizza XML per descrivere le asserzioni di sicurezza che possono essere comprese in tutti i domini di sicurezza. SAML definisce un protocollo standard per generare, scambiare ed elaborare asserzioni. I binding SAML mappano il modo in cui viene trasportato un documento SAML: SAML richiede HTTPS, ma può anche essere utilizzato all'interno dei messaggi SOAP per rappresentare i token WS-Security. Le asserzioni SAML (Figura 93 e i protocolli corrispondenti vengono utilizzati per:

- Autenticazione: verifica delle credenziali di identità
- Attributi: informazioni associate a soggetti (ad es. L'indirizzo dell'utente o lo stato del saldo corrente dell'account)
- Autorizzazione: concedere (o negare) l'accesso a una risorsa per un soggetto autenticato. (A partire da SAML 2.0, questa funzione utilizza XACML).
- Dichiarazioni personalizzate

```
<Assertion Version="2.0" AssertionID="123042134"
        IssueInstant="2005-11-23...">
    <Issuer>saml.ethz.ch</Issuer>
    <Subject>
        <NameID Format="emailAddress">
    pautasso@inf.ethz.ch</NameID>
        <SubjectConfirmation Method="holder-of-key">
            <SubjectConfirmationData>
                    <ds:KeyInfo>...</ds:KeyInfo>
            </SubjectConfirmationData>
        </SubjectConfirmation>
    </Subject>
    <Conditions NotBefore="2005-11-23..."
                NotOnOrAfter="2005-11-24..."><OneTimeUse/>
    </Conditions>
    ...statements...
</saml:Assertion>
```

Figura 93: SAML Assertion

Una dichiarazione di *asserzione di autenticazione* viene prodotta da un'autorità di autenticazione (emittente) per affermare che: un soggetto (con qualche identificazione) con un certo metodo (o classe di contesto) in un determinato momento è stato identificato con successo. A seconda del metodo, l'affermazione di autenticazione può essere considerata attendibile con un certo livello di confidenza per rappresentare l'identità digitale del soggetto per un certo periodo di tempo.

Figura 94: SAML Authentication Assertion

Per descrivere come è stata autenticata l'identità di un soggetto, SAML 2.0 definisce le seguenti classi di contesto di autenticazione:

- Indirizzo del protocollo Internet
- Nome utente/password su HTTP o HTTPS
- Password remota sicura
- Indirizzo IP e nome utente/password
- Autorizzazione client basata su certificato SSL / TLS
- Ticket Kerberos
- Chiave pubblica (X.509, PGP, SPKI, XML Signature)
- Numero di telefono

- Smartcard: One Factor, Two Factors
- Cellulare: One Factor, Two Factors
- Sessione precedente
- Non specificato

Un'asserzione di attributo SAML contiene informazioni su un utente sotto forma di una serie di attributi. Un'autorità afferma che l'oggetto è associato agli *attributi* specificati. I profili SAML mostrano come applicare gli attributi per standardizzare l'accesso alle directory delle informazioni sugli attributi degli utenti:

- LDAP / X.500
- PAC DCE (certificato di attributo privilegio)
- XACML (eXtensible Access Control Markup Language)

Inoltre, gli attributi possono modellare le informazioni relative alla contabilità: qual è l'importo del credito rimasto nell'account o lo stato del pagamento per un utente.

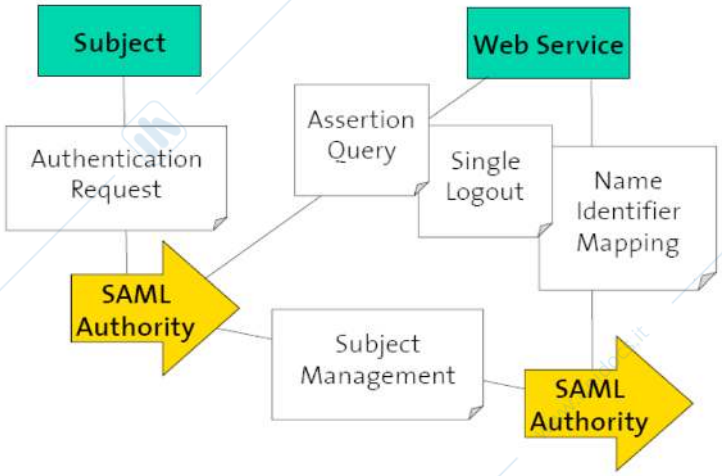

Figura 95: SAML

La Figura 96 rappresenta l'intero processo di autenticazione all'interno di uno scambio di messaggi SOAP.

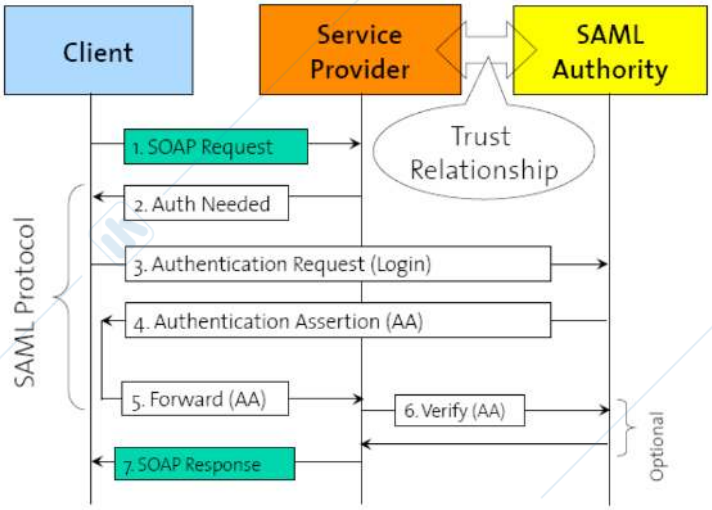

Figura 96: SAML

2.1.6 XACML (eXtensible Access Control Markup Language)

XACML (eXtensible Access Control Markup Language) è uno standard pubblicato da OASIS (Organization for the Advancement of Structured Information Standards) che definisce un linguaggio per la definizione di politiche di controllo degli accessi in formato XML, e illustra come valutare le richieste di autorizzazione. La versione attualmente stabile è la 3.0 del Gennaio 2013. L'obiettivo dello standard è rappresentare le politiche di controllo degli accessi in XML. La soluzione consise nel definire uno schema XML per rappresentare le regole di autorizzazione per concedere (o rifiutare) ai soggetti l'accesso alle risorse target per eseguire azioni specifiche. Le caratteristiche di XACML sono:

- controllo accurato: obiettivi referenziati tramite URL
- coerenti e basati su SAML

Lo standard XACML si basa sulla definizione di policy per il controllo degli accessi a risorse. La richiesta di accesso è definita con una richiesta XACML e l'esito del processo di decisione autorizzativa è dato dalla combinazione delle regole (rules) definite nelle politiche (policies) applicabili a tale richiesta. I vantaggi che nascono dall'utilizzo di XACML sono:

- interoperabilità di diversi strumenti di sicurezza (migrazione delle regole tramite importazione / esportazione).
- modo uniforme per specificare le politiche di controllo degli accessi.
- riutilizzo del servizio di controllo di accesso generico.
- consentito il consolidamento delle politiche di controllo degli accessi in tutta l'azienda: la centralizzazione riduce i costi.

Il linguaggio si basa su un processo di valutazione in stile Policy Based Access Control (PBAC). Le singole policy applicano il modello Attribute Based Access Control (ABAC).

Poiché la definizione dello standard è XML, il linguaggio può essere implementato per qualsiasi architettura.

L'autorizzazione è il permesso concesso a un soggetto di eseguire alcune azioni su una risorsa di destinazione.

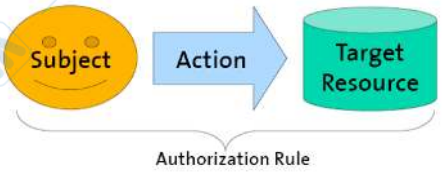

Figura 97: Regola di autorizzazione

Gli strumenti di gestione dei diritti controllano se un soggetto ha i diritti di autorizzazione. I diritti di accesso possono essere concessi a singoli soggetti, ma anche a gruppi di soggetti (o ruoli). Ogni Policy è formata da alcuni elementi:

- Una o più Rule, ovvero la singola regola nella politica. Essa a sua volta è formata da:
  - L'effetto della regola (permit/deny).
  - La condizione da verificare (opzionale).
- Il Target, usato per controllare l'applicabilità della richiesta ed indicizzare le varie policy per il PDP. Contiene:
  - Uno o più Soggetti, può contenere l'elenco degli attributi legato al soggetto al quale è rivolta la policy.
  - Un'Azione, ciò che la policy permette di fare (view, execute, ecc).
  - Le Risorse, cioè il riferimento alle risorse da proteggere (URI).

```
<Rule RuleId="1" Effect="Permit">
<Description>Allow Daniel to send a message</Description>
<Target>
   <Subjects>
      <Subject><SubjectMatch MatchID="string-equal">
            <AttributeValue>Daniel</AttributeValue>
            <SubjectAttributeDesignator AttributeId="subject-id"/>
      </SubjectMatch></Subject>
   </Subjects>
   <Resources><Resource><ResourceMatch MatchID="anyURI-equal">
      <AttributeValue>uri:message</AttributeValue>
      <ResourceAttributeDesignator AttributeID="resource=id"/>
   </ResourceMatch></Resource></Resources>
   <Actions><Action><ActionMatch MatchID="string-equal">
      <AttributeValue>send</AttributeValue>
      <ActionAttributeDesignator AttributeID="action-id"/>
   </ActionMatch></Action></Actions>
</Target>
</Rule>
```

Figura 98: Esempio di regola XACML

Il linguaggio XACML definisce due categorie di entità, entrambe definite da un Uniform Resource Identifier:

- Subject: ovvero l'elemento che richiede l'accesso (es. Utenti, Computer, Servizi web...).
- Resources: cioè gli elementi a cui il Subject vuole accedere (es. File, Documenti, Database...).

Per svolgere la sua funzione, XACML ha bisogno di vari componenti logici che svolgono funzioni separate:

- PEP: il Policy Enforcement Point, il quale protegge una risorsa e ne permette l'accesso solo se la verifica di compatibilità con la policy è positiva.
- PDP: Policy Decision Point, il quale riceve tutti i vari parametri (Policy, Soggetto, Risorsa richiesta, Tipo di accesso, Contesto...) e decide se concedere o meno l'accesso. Presa la decisione, questa sarà comunicata al PEP.
- PIP: Policy Information Point, fornisce le informazioni relative all'accesso richiesto.
- PAP: Policy Access Point, fornisce la policy applicabile all'accesso richiesto.

Il tipico funzionamento dello schema XACML (Figura 99) è il seguente:

1. Un soggetto (ad es. un Utente) vuole accedere a delle informazioni presenti su un Database. Per far ciò, deve comunicare al PEP la sua identità, la risorsa a cui vuole accedere e l'operazione che vuole fare (Lettura, Scrittura...).
2. Il PEP, ricevuta la richiesta, la congela temporaneamente e consulta (attraverso un "traduttore di richiesta") il PDP. Il traduttore è necessario in quanto ogni richiesta deve essere "contestualizzata" insieme alle informazioni provenienti dal PIP. Queste informazioni sono comunicate con un linguaggio apposito (SAML).
3. Il PDP, a sua volta, richiede la corrispondente policy di sicurezza al PAP, il quale avendo accesso al Policy Repository potrà fornire le informazioni necessarie. Il PDP, avendo ora a disposizione tutti gli elementi (la richiesta, il contesto e la policy di sicurezza) deciderà se consentire o meno l'accesso e comunicherà il risultato al PEP.
4. Il PEP, ricevuta la decisione del PDP, autorizzerà (o bloccherà) l'accesso dell'utente alla risorsa.

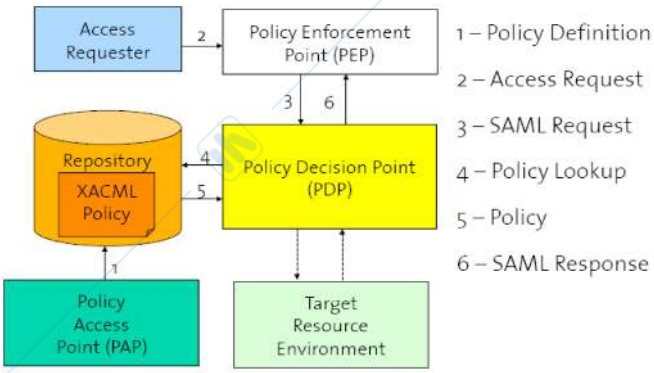

Figura 99: Architettura XACML

La Figura 100 rappresenta il modello di flusso di dati in XACML:

1. I PAP scrivono politiche e serie di politiche e le rendono disponibili per il PDP. Queste politiche o serie di politiche rappresentano la politica completa per un obiettivo specificato
2. Il richiedente l'accesso invia una richiesta di accesso al PEP
3. Il PEP invia la richiesta di accesso al gestore del contesto nel suo formato di richiesta nativo, opzionalmente includendo gli attributi di soggetti, risorse, azione e ambiente
4. Il gestore di contesto crea un contesto di richiesta XACML e lo invia al PDPi
5. Il PDP richiede ulteriori attributi soggetto, risorsa, azione e ambiente dal gestore del contesto
6. Il gestore di contesto richiede gli attributi da un PIP
7. Il PIP ottiene gli attributi richiesti
8. Il PIP restituisce gli attributi richiesti al gestore del contesto
9. Facoltativamente, il gestore di contesto include la risorsa nel contesto
10. Il gestore di contesto invia gli attributi richiesti e (facoltativamente) la risorsa al PDP. Il PDP valuta la politica
11. Il PDP restituisce il contesto di risposta (compresa la decisione di autorizzazione) al gestore del contesto
12. Il gestore di contesto traduce il contesto di risposta nel formato di risposta nativo del PEP. Il gestore di contesto restituisce la risposta al PEP
13. Il PEP soddisfa gli obblighi
14. (Non mostrato) Se l'accesso è consentito, il PEP consente l'accesso alla risorsa; in caso contrario, nega l'accesso

Esistono molti linguaggi proprietari e specifici dell'applicazione specifici per fare questo tipo di cose, ma XACML ha diversi punti a suo favore:

- É standard. Usando un linguaggio standard, stai usando qualcosa che è stato rivisto da una grande comunità di esperti e utenti, non è necessario implementare il tuo sistema ogni volta e non devi pensare a tutti i problemi coinvolto nella progettazione di una nuova lingua. Inoltre, poiché XACML viene distribuito in modo più diffuso, sarà più facile interagire con altre applicazioni utilizzando lo stesso linguaggio standard.
- É generico Ciò significa che, anziché cercare di fornire il controllo dell'accesso per un determinato ambiente o un tipo specifico di risorsa, può essere utilizzato in qualsiasi ambiente. È possibile scrivere una politica che può essere utilizzata da molti diversi tipi di applicazioni e quando viene utilizzata una lingua comune, la gestione delle politiche diventa molto più semplice.

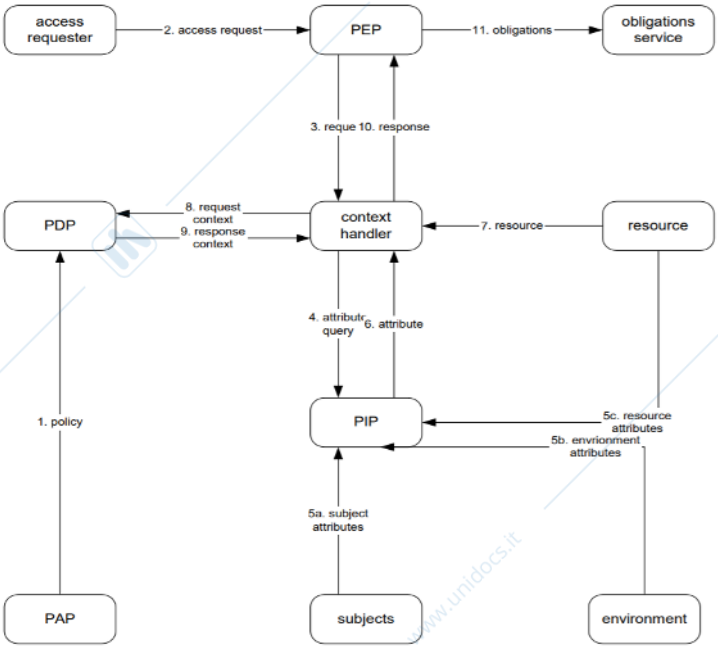

Figura 100: Data Flow XACML

- É distribuito. Ciò significa che è possibile scrivere una politica che a sua volta si riferisce ad altre politiche mantenute in posizioni arbitrarie. Il risultato è che anziché dover gestire una singola politica monolitica, persone o gruppi diversi possono gestire sotto-parti di politiche come appropriato, e XACML sa come combinare correttamente i risultati di queste diverse politiche in un'unica decisione.

- É potente. Mentre ci sono molti modi per estendere la lingua di base, molti ambienti non dovranno farlo. Il linguaggio standard supporta già un'ampia varietà di tipi di dati, funzioni e regole sulla combinazione dei risultati di diverse politiche. Inoltre, ci sono già gruppi di standard che lavorano su estensioni e profili che agganceranno XACML ad altri standard come SAML e LDAP, aumentando il numero di modi in cui XACML può essere usato.

La Figura 101 mostra gli schemi in XACML.

Alla radice di tutti i criteri XACML c'è un Policy o un PolicySet. Un PolicySet è un contenitore che può contenere altri Policy o PolicySet, nonché riferimenti a policy presenti in posizioni remote. Una politica rappresenta una singola politica di controllo dell'accesso, espressa attraverso un insieme di regole. Ogni documento della politica XACML contiene esattamente un tag XML radice Policy o PolicySet. Poiché un Policy o PolicySet può contenere più policy o Regole, ognuna delle quali può valutare decisioni di controllo dell'accesso diverse, XACML ha bisogno di un modo per conciliare le decisioni prese da ciascuna. Questo viene fatto attraverso una raccolta di algoritmi combinati. Ogni algoritmo rappresenta un modo diverso di combinare più descrizioni in una singola descrizione. Esistono algoritmi di combinazione di criteri (utilizzati da PolicySet) e algoritmi di combinazione di regole (utilizzati da Policy). Un esempio di questo è l'algoritmo Nega Ignora, che dice che, indipendentemente da cosa, se una valutazione restituisce Nega o nessun permesso di valutazione, anche il risultato finale è Nega. Questi algoritmi

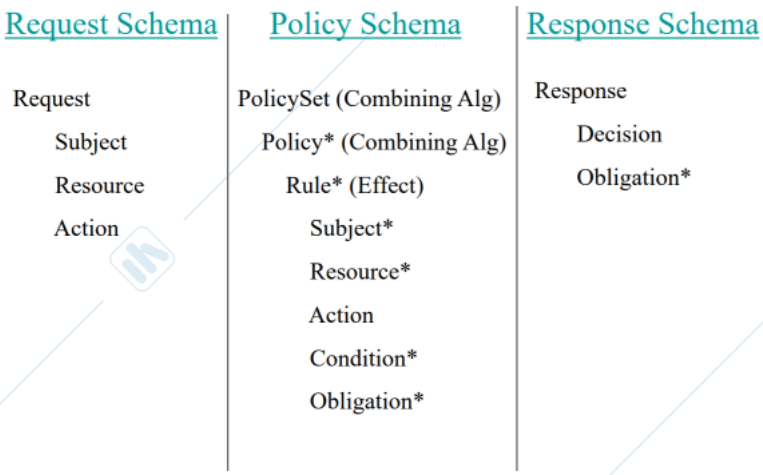

Figura 101: XACML Schemas

combinati vengono utilizzati per creare politiche sempre più complesse e, sebbene esistano sette algoritmi standard, è possibile creare i propri per soddisfare le proprie esigenze.

L'elemento chiave di livello superiore è <PolicySet> che aggrega altri elementi <PolicySet> o <Policy> L'elemento <Policy> è composto principalmente da <Target>, <RuleSet> e <Obligation> e viene valutato dal PDP per produrre e accedere alle decisioni. Poiché è possibile trovare più politiche applicabili a una decisione di accesso (e poiché una singola politica può contenere più regole), gli algoritmi di combinazione vengono utilizzati per riconciliare più risultati in un'unica decisione L'elemento <Target> viene utilizzato per associare una risorsa richiesta a una politica applicabile. Contiene le condizioni che l'oggetto, la risorsa o l'azione richiesti devono soddisfare affinché una serie di politiche, una politica o una regola siano applicabili alla risorsa. L'obiettivo include uno schema integrato per un'indicizzazione / ricerca efficiente delle politiche. Le regole forniscono le condizioni che verificano gli attributi rilevanti all'interno di una politica. È possibile utilizzare qualsiasi numero di elementi della regola, ciascuno dei quali genera un risultato vero o falso. La combinazione di questi risultati produce un'unica decisione per la Politica, che può essere "Autorizza", "Rifiuta", "Indeterminata" o "Non applicabile".

La Politica è il più piccolo elemento che PDP può valutare. Contiene: descrizione, valori predefiniti, target, regole, obblighi, algoritmo di combinazione delle regole. Il Set di politiche consente di combinare politiche e serie di politiche. L'uso non è richiesto. Contiene: descrizione, valori predefiniti, target, politiche, serie di politiche, riferimenti alle politiche, riferimenti alle politiche, obblighi, algoritmo di combinazione delle politiche. É possibile anche combinare gli algoritmi:

- Permit Overrides: Se una singola regola consente una richiesta, indipendentemente dalle altre regole, il risultato del PDP è Autorizzazione
- Deny Overrides: Se una singola regola rifiuta una richiesta, indipendentemente dalle altre regole, il risultato del PDP viene negato.
- First Applicable: La prima regola applicabile che soddisfa la richiesta è il risultato del PDP
- Only-One-applicabile: Se ci sono due regole con effetti diversi per la stessa richiesta, il risultato è indeterminato

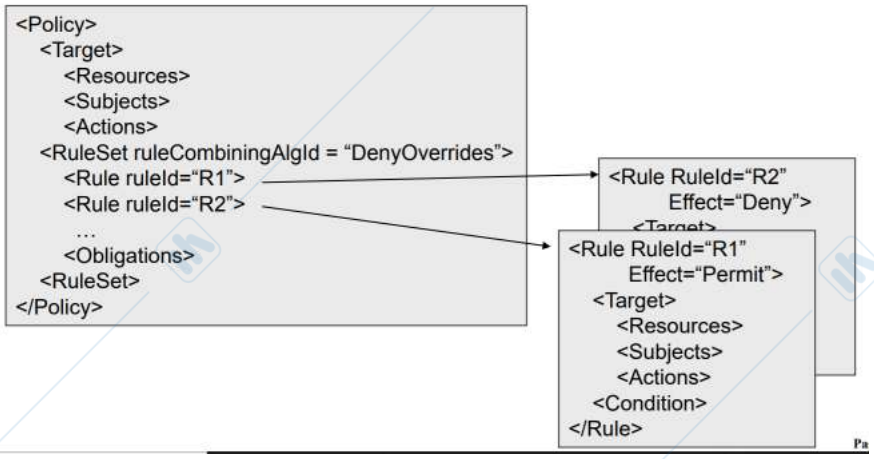

Figura 102: Overview of the Policy Element

Parte di ciò che un PDP XACML deve fare è trovare una politica che si applica a una determinata richiesta. Per fare ciò, XACML fornisce un'altra funzionalità chiamata Target. Un obiettivo è fondamentalmente un insieme di condizioni semplificate per soggetto, risorsa e azione che devono essere soddisfatte affinché un gruppo di politiche, una politica o una regola si applichino a una determinata richiesta. Questi usano funzioni booleane (spiegate più avanti nella prossima sezione) per confrontare i valori trovati in una richiesta con quelli inclusi nella Target. Se vengono soddisfatte tutte le condizioni di un Target, allora il PolicySet, Policy o Rule associato si applica alla richiesta. Oltre ad essere un modo per verificare l'applicabilità, le informazioni sul target forniscono anche un modo per indicizzare le politiche, che è utile se è necessario archiviare molte politiche e quindi passarle rapidamente in rassegna per trovare quelle che si applicano. Ad esempio, una politica può essere associata a una destinazione che si applica solo alle richieste su un servizio specifico. Quando arriva una richiesta di accesso a quel servizio, il PDP saprà dove cercare le politiche che potrebbero applicarsi a questa richiesta perché le politiche sono indicizzate in base ai loro vincoli Target. Si noti che un Target può anche specificare che si applica a qualsiasi richiesta. Una volta trovata e verificata una politica da applicare a una richiesta, le sue regole vengono valutate. Una politica può avere un numero qualsiasi di Regole che contengono la logica principale di una politica XACML. Il cuore della maggior parte delle Regole è una Condizione, che è una funzione booleana. Se la condizione viene valutata come vera, viene restituito l'effetto della regola (un valore di Consenti o Nega associato a una valutazione corretta della regola). La valutazione di una condizione può anche comportare un errore (indeterminato) o l'individuazione che la condizione non si applica alla richiesta (non applicabile). Una Condizione può essere piuttosto complessa, costruita da un annidamento arbitrario di funzioni e attributi non booleani.

La regola è la più piccola unità di amministrazione, non può essere valutata da sola. Gli elementi di una regola sono:

- Descrizione: documentazione
- Target: selezionare le regole applicabili
- Condizione: funzione di decisione booleana
- Effetto: "Permetti" o "Nega"

Il risultato che risulta della valutazione della regola sono:

- Se la condizione è vera, restituisce il valore dell'effetto
- In caso contrario, restituire NotApplicable
- Se l'errore o i dati mancanti restituiscono il codice di stato Indeterminato-Plus

Il target è progettato per trovare in modo efficiente le politiche che si applicano a una richiesta. Rende possibile avere condizioni molto complesse (attributi di soggetti, risorse e azioni). Corrisponde al valore, utilizzando la funzione di corrispondenza:

- Espressione regolare
- Nome RFC822 (e-mail)
- Nome X.500
- Definito dall'utente

Gli attributi sono specificati dall'espressione Id o XPath: normalmente usa Soggetto o Risorsa, non entrambi. I componenti principali dell'elemento <rule> sono:

- <target>. L'elemento <target> è costituito da:
  - un insieme di elementi <resource>
  - un insieme di elementi <action>
  - un ambiente

  L'elemento <target> può essere assente da una <regola>. In questo caso il <target> della regola è uguale a quello dell'elemento <policy> padre.
- <effect>. Sono ammessi due valori: "Autorizza" e "Nega".
- <condiction>

I componenti principali di un elemento <policy> sono:

- un elemento <target>. L'elemento <target> è costituito da:
  - un insieme di elementi <resource>
  - un insieme di elementi <action>
  - un ambiente

  L'elemento <target> può essere dichiarato esplicitamente o può essere calcolato; due possibili approcci:
  - Crea l'unione di tutti gli elementi target nelle regole interne
  - Crea l'intersezione di tutti gli elementi target nelle regole interne
- un identificativo algoritmo che combina le regole
- un insieme di elementi <rule>

- obblighi

I componenti principali di un elemento <policyset> sono:

- a <target>
- un identificativo algoritmo che combina le politiche
- un insieme di elementi <policy>
- obblighi

Una condizione è una funzione booleana per decidere se applicare l'effetto. Gli input provengono dal contesto della richiesta. I valori possono essere primitivi, complessi o bags. La condizione può essere specificata dall'id o dall'espressione XPath. Esistono quattordici tipi primitivi. Definizione di una ricca gamma di funzioni tipizzate. Funzioni per la gestione delle borse. Ordine di valutazione non specificato. Permesso di uscire quando il risultato è noto. Nelle condizioni gli effetti collaterali non sono ammessi.
Le funzioni possibili sono:

- Equality predicates
- Arithmetic functions
- String conversion functions
- Numeric type conversion functions
- Logical functions
- Arithmetic comparison functions
- Date and time arithmetic functions
- Non-numeric comparison functions
- Bag functions
- Set functions
- Higher-order bag functions
- Special match functions
- XPath-based functions
- Extension functions and primitive types

La valuta in cui si occupa XACML sono gli attributi. Gli attributi sono valori nominati di tipi noti che possono includere un identificatore dell'emittente o una data e ora di emissione. In particolare, gli attributi sono caratteristiche dell'oggetto, della risorsa, dell'azione o dell'ambiente in cui viene effettuata la richiesta di accesso. Il nome di un utente, la sua autorizzazione di sicurezza, il file a cui desidera accedere e l'ora del giorno sono tutti valori di attributo. Quando una richiesta viene inviata da un PEP a un PDP, tale richiesta è formata quasi esclusivamente da attributi e verranno confrontati con i

valori degli attributi in una politica per prendere le decisioni di accesso. Una politica risolve i valori degli attributi da una richiesta o da qualche altra fonte attraverso due meccanismi: AttributeDesignator e AttributeSelector. Un AttributeDesignator consente alla politica di specificare un attributo con un determinato nome e tipo, e facoltativamente anche un emittente, e quindi il PDP cercherà quel valore nella richiesta, o altrove se non è possibile trovare valori corrispondenti nella richiesta. Esistono quattro tipi di designatori, uno per ciascuno dei tipi di attributi in una richiesta: Oggetto, Risorsa, Azione e Ambiente. Poiché gli attributi del soggetto possono essere suddivisi in diverse categorie, SubjectAttributeDesignators può anche specificare una categoria in cui cercare. AttributeSelector consente a un criterio di cercare i valori degli attributi attraverso una query XPath. Vengono forniti un tipo di dati e un'espressione XPath, che possono essere utilizzati per risolvere alcuni set di valori nel documento di richiesta o altrove. Sia AttributeDesignator che AttributeSelector possono restituire più valori (poiché potrebbero esserci più corrispondenze in una richiesta o altrove), quindi XACML fornisce un tipo di attributo speciale chiamato Bag. Le borse sono raccolte non ordinate che consentono duplicati e sono sempre ciò che i designatori e i selettori restituiscono, anche se è stato trovato un solo valore. Nel caso in cui non siano state effettuate corrispondenze, viene restituito un sacchetto vuoto, anche se un designatore o un selettore può impostare un flag che causa un errore invece in questo caso. Una volta recuperati alcuni valori di Borsa di attributi, è necessario confrontarli in qualche modo con i valori previsti per prendere le decisioni di accesso. Questo viene fatto attraverso un potente sistema di funzioni. Le funzioni possono funzionare su qualsiasi combinazione di valori di attributo e possono restituire qualsiasi tipo di valore di attributo supportato nel sistema. Le funzioni possono anche essere nidificate, quindi è possibile avere funzioni che operano sull'output di altre funzioni e questa gerarchia può essere arbitrariamente complessa. È possibile scrivere funzioni personalizzate per fornire un linguaggio sempre più ricco per esprimere le condizioni di accesso. Una cosa da notare quando si creano queste gerarchie di funzioni è che la maggior parte delle funzioni sono definite come funzionanti su tipi specifici (come stringhe o numeri interi) mentre i designatori e i selettori restituiscono sempre Borse di valori. A tale scopo, XACML definisce una raccolta di funzioni standard della forma [tipo] -un-e-solo, che accettano un sacco di valori del tipo specificato e restituiscono il valore singolo se nel sacchetto è presente esattamente un elemento, oppure un errore se ci sono zero o più valori nella borsa. Questa è una delle funzioni più comuni che vedrai in una Condizione. [target] le funzioni solo-solo non sono necessarie nei target, poiché il PDP applica automaticamente la funzione di abbinamento a ciascun elemento di un sacchetto.

I contesti di XACML sono:

- Request context: Attributi di:
  - Soggetti - richiedente, intermediario, destinatario, ecc.
  - Resource: nome, può essere gerarchico
  - Risorse contenuto - specifico per il tipo di risorsa, ad es. Documento XML
  - Azione - ad es. Leggere
  - Ambiente - altro, ad es. ora della richiesta
- Response context:
  - ID risorsa

- Decisione
- Stato (valori di errore)
- Obblighi

Ecco un semplice esempio di politica che utilizza quelle funzionalità discusse sopra. Il target indica che il criterio si applica solo alle richieste per il server chiamato "SampleServer". La Politica ha una Regola con un Target che richiede un'azione di "login" e una Condizione che si applica solo se il Soggetto sta tentando di accedere tra le 9:00 e le 17:00. Si noti che questo esempio può essere esteso per includere altre Regole per azioni diverse. Se la prima regola fornita qui non si applica, viene utilizzata una regola predefinita che restituisce sempre Nega (le regole vengono valutate in ordine).

Le Figure 124 e 125 mostrano l'implementazione di questa policy.

3 Resource-oriented services

Il Web è nato negli anni '90 come una piattaforma per la condivisione di documenti distribuiti su diverse macchine e tra loro interconnessi. Il tutto è nato e si è evoluto grazie alla standardizzazione di alcuni semplici concetti:

- URI – meccanismo per individuare risorse in una rete
- HTTP – protocollo semplice e leggero per richiedere una risorsa ad una macchina
- HTML – linguaggio per la rappresentazione dei contenuti

Questa semplice idea iniziale si è evoluta nel corso degli anni non tanto nei concetti di base, quanto nel modo di intenderli e di utilizzarli. Al testo si sono aggiunti contenuti multimediali, i documenti sono generati dinamicamente e non pubblicati come pagine statiche, e poi il Web esce dalla visione esclusivamente ipertestuale per diventare un il contenitore di applicazioni software interoperabili: una piattaforma applicativa distribuita. Questa prospettiva ha dato origine, intorno all'anno 2000, al concetto di Web Service: un sistema software progettato per supportare un'interazione tra applicazioni, utilizzando le tecnologie e gli standard Web. Il meccanismo dei Web Service consente di far interagire in maniera trasparente applicazioni sviluppate con linguaggi di programmazione diversi, che girano su sistemi operativi eterogenei. Questo meccanismo consente di realizzare porzioni di funzionalità in maniera indipendente e su piattaforme potenzialmente incompatibili facendo interagire i vari pezzi tramite tecnologie Web e creando un'architettura facilmente componibile. Allo stato attuale esistono due approcci alla creazione di Web Service:

- Un approccio è basato sul protocollo standard SOAP (Simple Object Access Protocol), per lo scambio di messaggi per l'invocazione di servizi remoti, si prefigge di riprodurre in ambito Web un approccio a chiamate remote, Remote Procedure Call, tipico di protocolli di interoperabilità come CORBA, DCOM e RMI.
- Un secondo approccio, introdotto a partire dal 2007, è ispirato ai principi architetturali tipici del Web e si concentra sulla descrizione di risorse, sul modo di individuarle nel Web e sul modo di trasferirle da una macchina all'altra. Questo è l'approccio che prende il nome di REST (REpresentational State Transfer) - Figura 103.

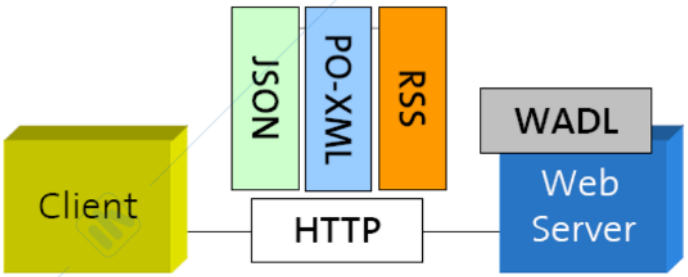

Figura 103: RESTful Web Services

I Web services affrontano il problema della standardizzazione del software aziendale, cioè Enterprise Computing Standards for Interoperability (WS iniziato 2001). L'architettura dei servizi Web mette in relazione vari componenti e tecnologie che comprendono uno "stack" di servizi Web o un'implementazione completamente funzionale. Le implementazioni valide includono sottoinsiemi o parti dello stack, ma devono almeno fornire i componenti all'interno dell'architettura di base. I componenti e le tecnologie che estendono l'architettura di base sono rappresentati all'interno dell'architettura estesa. L'architettura di base include tecnologie di servizi Web in grado di:

- Scambio di messaggi
- Descrizione di servizi Web
- Pubblicazione e scoperta delle descrizioni dei servizi Web

Quindi, si tratta di un'architettura a più livelli con una varietà di specifiche di messaggistica, descrizione e scoperta. Gli strumenti nasconderanno la complessità delle operazioni. I web service affrontano il problema dell'eterogeneità, tra applicazioni web e tra composizioni aziendali.

I big web service utilizzano messaggi XML che seguono lo standard SOAP (Simple Object Access Protocol), un linguaggio XML che definisce un'architettura e i formati dei messaggi. Tali sistemi contengono spesso una descrizione leggibile meccanicamente delle operazioni offerte dal servizio, scritta nel WSDL (Web Services Description Language), un linguaggio XML per la definizione sintattica delle interfacce. I big web service hanno un'elevata complessità percepita e un processo di standardizzazione problematico, per via di:

- Infighting: conflitto nascosto o competitività all'interno di un'organizzazione.
- Mancanza di coerenza architettonica
- Frammentazione
- Prodotto del committee
- Feature Bloat (Unione di specifiche concorrenti)
- Mancanza di implementazioni di riferimento

- Standardizzazione degli standard (WS-I)

Le domande che sorgono spontanee sono: Sta iniziando a sembrare CORBA? Quando l'interoperabilità dei servizi Web inizierà a funzionare davvero? Dobbiamo davvero acquistare appliance XML per ottenere buone prestazioni?

3.1 REST

REST definisce un insieme di principi architetturali per la progettazione di sistemi distribuiti. Rappresenta uno stile architetturale, cioè non si riferisce ad un sistema concreto e ben definito né si tratta di uno standard stabilito da un organismo di standardizzazione. La sua definizione è apparsa per la prima volta nel 2000 nella tesi di Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures, discussa presso l'Università della California, ad Irvine. In questa tesi si analizzavano alcuni principi alla base di diverse architetture software, tra cui appunto i principi di un'architettura software che consentisse di vedere il Web come una piattaforma per l'elaborazione distribuita. É bene precisare che i principi REST non sono necessariamente legati al Web, nel senso che si tratta di principi astratti di cui il World Wide Web ne risulta essere un esempio concreto. I suoi quattro principi possono spiegare il successo e la scalabilità del protocollo HTTP che li implementa:

- Identificazione delle risorse tramite URI
- Interfaccia uniforme per tutte le risorse:
  - GET: Richiedi lo stato. Utilizzare le richieste GET per recuperare solo la rappresentazione/informazione delle risorse e non modificarle in alcun modo. Il metodo GET è un metodo sicuro, ovvero un safe method o nullipotente, il che significa che la sua invocazione non produce alcun effetto collaterale: recuperare o accedere un record non lo modifica. Inoltre, le API GET devono essere idempotenti, il che significa che effettuare più richieste identiche deve produrre lo stesso risultato ogni volta fino a quando un'altra API (POST o PUT) ha modificato lo stato della risorsa sul server.
  - POST: Crea una risorsa figlio. Utilizzare le API POST per creare nuove risorse subordinate, ad esempio un file è subordinato a una directory che lo contiene o una riga è subordinata a una tabella del database. Parlando rigorosamente in termini di REST, i metodi POST vengono utilizzati per creare una nuova risorsa nella raccolta di risorse.
  - PUT: Aggiorna, trasferisci un nuovo stato. Utilizzare le API PUT principalmente per aggiornare la risorsa esistente (se la risorsa non esiste, l'API potrebbe decidere di creare una nuova risorsa o meno).
  - DELETE: Elimina una risorsa
  - PATCH: Le richieste HTTP PATCH devono effettuare un aggiornamento parziale su una risorsa. Il metodo PATCH è la scelta corretta per aggiornare parzialmente una risorsa esistente e PUT dovrebbe essere usato solo se stai sostituendo una risorsa nella sua interezza.
- Messaggi di "auto-descrizione" tramite metadati e rappresentazioni di risorse multiple

- Collegamenti ipertestuali per definire le transizioni dello stato dell'applicazione e le relazioni tra le risorse

Le risorse sono gli elementi fondamentali su cui si basano i Web Service RESTful, a differenza dei Web Service SOAP-oriented che sono basati sul concetto di chiamata remota. Per risorsa si intende un qualsiasi elemento oggetto di elaborazione. Per fare qualche esempio concreto, una risorsa può essere un cliente, un libro, un articolo, un qualsiasi oggetto su cui è possibile effettuare operazioni. Il principio di Resource Identification stabilisce che ciascuna risorsa deve essere identificata univocamente. Essendo in ambito Web, il meccanismo più naturale per individuare una risorsa è dato dal concetto di URI (Standard Internet per la denominazione e l'identificazione delle risorse (originariamente dal 1994, rivisto fino al 2005)).Nella maggior parte degli stack HTTP gli URI (Uniform Resource Identifier) non possono avere lunghezza arbitraria (4Kb).

Figura 104: Uniform Resource Identifier (URI)

Il formato segue schemi prefissati. Il registro ufficiale viene mantenuto presso la Internet Assigned Numbers Authority. La sintassi di un URI dipende dallo schema. In generale, URI assoluti si scrivono come segue *<scheme>:<scheme-specific-part>*. Un URI assoluto contiene il nome dello schema usato (*<scheme>*) seguito dai due punti (":") e quindi da una stringa (*<scheme-specific-part>*) la cui interpretazione dipende dallo schema. La sintassi non prescrive che la componente scheme-specific-part abbia una struttura generale o una semantica comune a tutti gli URI. Tuttavia, un sottoinsieme di URI condivide una sintassi comune per rappresentare relazioni gerarchiche nel namespace. Questa sintassi di "URI generico" consiste di una sequenza di 4 componenti: *<scheme>://<authority><path>?<query>*, ognuna delle quali, tranne *<scheme>*, può anche non comparire in un determinato URI. Ad esempio, alcuni schemi di URI non ammettono una componente <authority> mentre altri non utilizzano la componente <query>. Come esempio, lo schema completo di una URL è del tipo (non tutte le componenti sono obbligatorie) *<scheme>://<domain>:<port>/<path>?<querystring>#<fragmentid>*.

Il principale beneficio nell'adottare lo schema URI per identificare le risorse consiste nel fatto che esiste già, è ben definito e collaudato e non occorre pertanto inventarsene uno nuovo. I seguenti sono esempi di possibili identificatori di risorse:

- http://www.myapp.com/clienti/1234
- http://www.myapp.com/ordini/2011/98765
- http://www.myapp.com/prodotti/7654
- http://www.myapp.com/ordini/2011
- http://www.myapp.com/prodotti?colore=rosso

Gli URI sono abbastanza autoesplicativi: i primi tre identificano rispettivamente un determinato cliente, un ordine ed un prodotto. Il quarto URI identifica l'insieme degli ordini del 2011, mentre l'ultimo URI identifica l'insieme dei prodotti di colore rosso. L'interpretazione che abbiamo dato a questi URI è però desunta dalla semantica delle parole contenute nelle sue parti. Dal punto di vista di un Web Service un URI è soltanto una stringa che identifica una risorsa, per cui anche http://www.myapp.com/tgw34/2099ww può essere un URI valido per identificare un cliente. Esiste alcune linee guida per il design di URI:

- evitare l'uso di verbi negli URI ma limitarsi ad utilizzare nomi, ricordandosi che un URI identifica una risorsa.
- Mantenere gli URI brevi
- Seguire uno schema di passaggio dei parametri "posizionali" (anziché la codifica key = value & p = v)
- I postfissi URI possono essere utilizzati per specificare il tipo di contenuto
- Non modificare gli URI
- Utilizzare il reindirizzamento se è necessario modificarli

Una volta spiegato come individuare una risorsa abbiamo bisogno di un meccanismo per indicare quali operazioni effettuare su di esse. Il principio dell'uso esplicito dei metodi HTTP ci indica di sfruttare i metodi (o verbi) predefiniti di questo protocollo, e cioè GET, POST, PUT e DELETE. Facciamo un semplice esempio per provare a chiarire questo concetto. Quando inseriamo un URI nella barra degli indirizzi di un browser stiamo in realtà chiedendo al browser di eseguire un metodo HTTP sulla risorsa individuata dall'URI. Il metodo che implicitamente stiamo eseguendo è GET, il cui effetto è l'accesso ad una rappresentazione della risorsa identificata dall'URI. Dal punto di vista del codice, non avremo bisogno di un metodo del tipo getISBN(222) per ottenere la rappresentazione del cliente con codice 222, sarà sufficiente sfruttare il metodo standard GET del protocollo HTTP sull'URI che identifica quel determinato cliente. Questo rende uniforme l'invocazione di operazioni sulle risorse, cioè il client non ha bisogno di sapere qual è la specifica interfaccia da utilizzare per invocare il metodo che consente di ottenere la rappresentazione di un cliente. In un contesto non RESTful potremmo avere metodi come getISBN() o getID() o altra specifica dipendente dalle scelte di chi ha sviluppato il Web Service. In un contesto RESTful sappiamo già a priori come ottenere la rappresentazione di una risorsa. In altre parole, questo principio REST stabilisce una mappatura uno a uno tra le tipiche operazioni CRUD (creazione, lettura, aggiornamento, eliminazione di una risorsa) e i metodi HTTP.

Metodo HTTP	Operazione CRUD	Descrizione
POST	Create	Crea una nuova risorsa
GET	Read	Ottiene una risorsa esistente
PUT	Update	Aggiorna una risorsa o ne modifica lo stato
DELETE	Delete	Elimina una risorsa

Figura 105: Uniform Interface Principle (CRUD)

Le risorse di per sè sono concettualmente separate dalle rappresentazioni restituite al client. Ad esempio, un Web Service non invia al client direttamente un record del suo database, ma una sua rappresentazione in una codifica dipendente dalla richiesta del client e/o dall'implementazione del servizio. I principi REST non pongono nessun vincolo sulle modalità di rappresentazione di una risorsa. Virtualmente possiamo utilizzare il formato che preferiamo senza essere obbligati a seguire uno standard. Di fatto, però, è opportuno utilizzare formati il più possibile standard in modo da semplificare l'interazione con i client. Inoltre, sarebbe opportuno prevedere rappresentazioni multiple di una risorsa, per soddisfare client di tipo diverso. Il tipo di rappresentazione inviata dal Web Service al client è indicato nella stessa risposta HTTP tramite un tipo MIME, così come avviene nella classica comunicazione tra Web server e browser. Un client a sua volta ha la possibilità di richiedere una risorsa in uno specifico formato sfruttando l'attributo Accept di una richiesta HTTP di tipo GET.

Un altro vincolo dei principi REST consiste nella necessità che le risorse siano tra loro messe in relazione tramite link ipertestuali. Questo principio è anche noto come HATEOAS, dall'acronimo di Hypermedia As The Engine Of Application State, e pone l'accento sulle modalità di gestione dello stato dell'applicazione. In sostanza, tutto quello che un client deve sapere su una risorsa e sulle risorse ad essa correlate deve essere contenuto nella sua rappresentazione o deve essere accessibile tramite collegamenti ipertestuali. Ad esempio, la rappresentazione di un ordine in un linguaggio XML-based deve contenere gli eventuali collegamenti agli articoli ed al cliente correlati (Figura 106).

```
<ordine>
  <numero>12345678</numero>
  <data>01/07/2011</data>
  <cliente rif="http://www.myapp.com/clienti/1234" />
  <articoli>
    <articolo rif="http://www.myapp.com/prodotti/98765" />
    <articolo rif="http://www.myapp.com/prodotti/43210" />
  </articoli>
</ordine>
```

Figura 106: Esempio collegamenti tra risorse

In questo modo il client può accedere alle risorse correlate seguendo semplicemente i collegamenti contenuti nella rappresentazione della risorsa corrente. Il fatto di utilizzare un URI come identificatore di una risorsa, quindi un meccanismo standard e consolidato, consente al client di accedere anche a risorse messe a disposizione da altre applicazioni

che girano eventualmente su altri server. Inoltre, la possibilità di avere collegamenti ipertestuali all'interno della rappresentazione di una risorsa rappresenta un meccanismo per poter gestire lo stato dell'applicazione.

É importante sottolineare che sebbene REST preveda la comunicazione stateless, non vuol dire che un'applicazione non deve avere stato. La responsabilità della gestione dello stato dell'applicazione non deve essere conferita al server, ma rientra nei compiti del client. La principale ragione di questa scelta è la scalabilità: mantenere lo stato di una sessione ha un costo in termini di risorse sul server e all'aumentare del numero di client tale costo può diventare insostenibile. Inoltre, con una comunicazione senza stato è possibile creare cluster di server che possono rispondere ai client senza vincoli sulla sessione corrente, ottimizzando le prestazioni globali dell'applicazione.

Le Best Practices differenziano il REST in:

- High REST, che prevede:
  - l'uso di URI "simpatici"
  - pieno utilizzo dei 4 verbi: GET, POST, PUT e DELETE
  - risposte utilizzando Plain Old XML (POX). POX è un riferimento a determinati tipi o usi di eXtensible Markup Language (XML) che sono meno complessi o sofisticati rispetto ad altre versioni del linguaggio. In generale, POX si riferisce ai semplici modi di utilizzare XML nei progetti.
- Low REST, che prevede:
  - l'uso di HTTP GET per richieste idempotenti, mentre l'uso del metodo POST per tutto il resto
  - risposte in qualsiasi tipo MIME (ad es. XHTML)

JSON (Javascript Object Notation) e XML (Extended markup language), entrambi, più che linguaggi, sono stili di formattazione, ossia formati di scambio per le informazioni. Questi stili di formattazione standard, garantiscono interoperabilità fra linguaggi, piattaforme e tecnologie di sviluppo diverse. Sono molto utilizzati come formati per le API. Contribuiscono alla definizione di web semantico. L'importanza di formattare e strutturare i dati, con tag precisi che identifichino meglio la risorsa, ha un'importanza estrema per l'organizzazione delle informazioni. XML è un linguaggio di markup in grado di gestire in modo completo qualsiasi tipo di informazione, mentre JSON è un formato di interscambio dati che risulta essere di più agile utilizzo in determinati contesti. XML ha sicuramente uno spettro di utilizzo molto più vasto dovuto sia alle caratteristiche intrinseche nel linguaggio stesso sia al fatto che intorno a questo formato si è da tempo evoluto un mondo molto significativo di direttive, metodologie, strumenti ed applicazioni. JSON invece, in forte ascesa solo da qualche anno, si caratterizza per la semplicità nella rappresentazione ed è ottimo per la serializzazione e la trasmissione di dati di tipo classico. Vediamo alcune caratteristiche di entrambi gli stili:

- XML
  - Esistono diversi linguaggi, basati su XML: PO-XML, SOAP (WS-*), RSS, ATOM
  - Sintassi testuale standard per dati semi-strutturati

  - Molti strumenti disponibili: Schema XML, DOM, SAX, XPath, XSLT, XQuery
  - Tutti possono analizzarlo (non necessariamente capirlo)
  - Lento e dettagliato (verbose)

- JSON
  - Introdotto per le applicazioni Web AJAX (comunicazione Browser-Web Server)
  - Sintassi testuale per la serializzazione di strutture dati non ricorrenti
  - Supportato nella maggior parte delle lingue (non solo JavaScript)
  - Non estensibile (non è necessario che lo sia)
  - "JSON è diventato la X in Ajax"

Non è possibile mettere in discussione l'universalità o il successo dell'XML, tanto meno le sue potenzialità. Sono rappresentabili gerarchie semplici o estremamente complesse con la possibilità di ricostruire oggetti e variabili dalla lettura di attributi presenti nei nodi. Esiste però un neo connaturato a XML: il numero elevato di caratteri necessari a rappresentare le informazioni. Se è vero che per un server leggere un file da 10Kb o da 1Mb è pressapoco equivalente, non lo è altrettanto per un client. Questa caratteristica è tanto più evidente quanto più piccola è l'informazione da trasportare. Si arriva al paradosso di avere più caratteri per la descrizione del documenti, che per i dati veri e propri. Un utilizzo massiccio di attributi potrebbe aiutare a rosicchiare qualche carattere ma oltre a perdere in linearità e leggibilità non si riuscirebbe comunque a guadagnare troppo spazio. É proprio in queste occasioni che JSON fa capolino, permettendo di rappresentare anche dati complessi senza aggiungere nulla all'infuori di qualche parentesi o qualche segno di punteggiatura. Il rapporto "peso del contenuto/informazioni presenti" va quindi a migliorare usando JSON invece di XML.

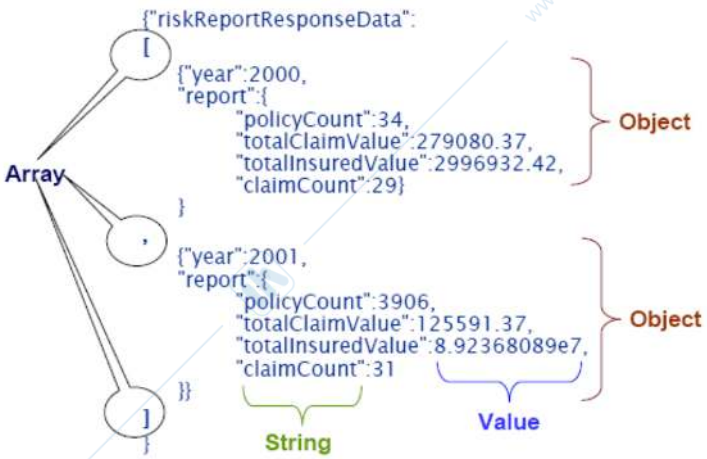

Figura 107: Esempio JSON

I punti di forza di REST sono:

- Semplicità: L'interfaccia uniforme è immutabile (nessun problema di rottura dei client)
- HTTP / POX è onnipresente (passa attraverso i firewall)
- Interazione stateless/sincrona
- Comprovata scalabilità: "dopo tutto il Web funziona", memorizzazione nella cache, server farm cluster per QoS
- Facilità percepita di adozione (infrastruttura leggera): è sufficiente un browser per iniziare e non è necessario acquistare il middleware WS-*.
- Approccio di livello basico
- Sfruttato da tutte le principali applicazioni Web 2.0: L'85% dei clienti preferisce l'API Amazon RESTful. Google non supporta più la sua API SOAP/WSDL

Mentre, i punti deboli sono:

- Confusione (High REST vs. Low REST): Sono davvero 4 verbi? (HTTP 1.1. Ha 8 verbi: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS e CONNECT)
- La mappatura della semantica sincrona in stile REST su sistemi back-end crea disallineamenti di progettazione (quando sono basati su messaggistica asincrona o interazione guidata da eventi)
- Non è possibile fornire "abilità" di tipo aziendale oltre HTTP/SSL
- É impegnativa l'identificazione e l'individuazione delle risorse in modo appropriato in tutte le applicazioni
- Apparente mancanza di standard (diversi da URI, HTTP, XML, MIME, HTML)
- Descrizione semantica/sintassi molto informale (orientata all'utente/all'uomo)

La metodologia di progettazione di servizi Web RESTful consiste nei seguenti passaggi:

1. Identificare le risorse da esporre come servizi (ad es. Report annuale sui rischi, catalogo di libri, ordine di acquisto, bug aperti, sondaggi e voti)
2. Definisci gli URL "nice" per indirizzarli
3. Capire cosa significa fare GET, POST, PUT, DELETE su un determinato URI di risorsa
4. Progettare e documentare le rappresentazioni
5. Modellare le relazioni (ad es. Contenimento, riferimento, transizioni di stato) tra risorse con collegamenti ipertestuali che possono essere aggiunti per ottenere maggiori dettagli (o eseguire transizioni di stato)
6. Implementare e distribuire sul server Web
7. Provare con un browser Web

3.1.1 Esempio di Doodle API

L'API Doodle ti consente di creare e condurre un sondaggio. É possibile utilizzare il servizio Doodle per pianificare un evento o raggiungere un consenso su un argomento. I sondaggi Doodle possono anche essere integrati in una pagina Facebook o igoogle. É un api di tipo REST. Le operazioni che sono possibili sull'API Doodle sono:

- Creazione di un sondaggio (trasferire lo stato di un nuovo sondaggio sul servizio Doodle) - Figura 108

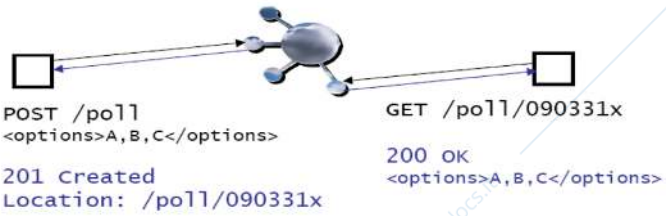

Figura 108: Creazione di un sondaggio

- Lettura di un sondaggio (trasferire lo stato del sondaggio dal servizio Doodle)
- Partecipazione a un sondaggio creando una nuova risorsa secondaria di voto - Figura 109

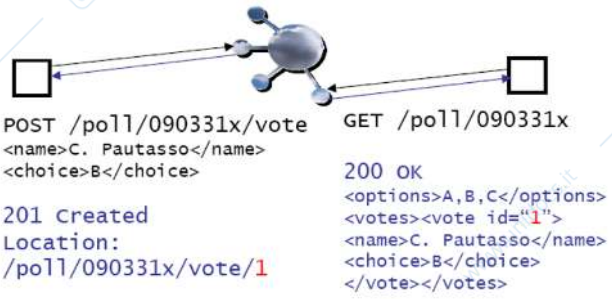

Figura 109: Partecipazione ad un sondaggio

- I voti esistenti possono essere aggiornati (Headers del controllo di accesso non visualizzate) - Figura 110

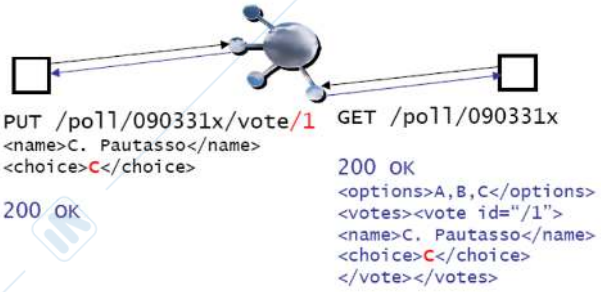

Figura 110: Update di voti

- I sondaggi possono essere eliminati una volta presa una decisione - Figura 111

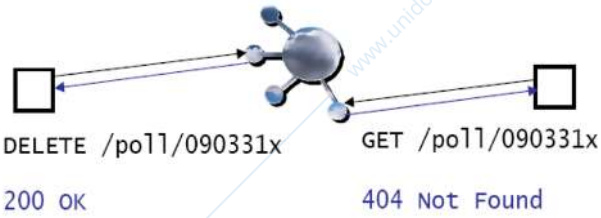

Figura 111: Cancellazione di un sondaggio

3.2 Avanzamenti

WS-Transaction fa parte di una serie di specifiche di un gruppo industriale che include IBM, Microsoft e BEA Systems. L'interfaccia WS-Transaction definisce cosa costituisce una transazione e cosa determinerà quando è stata completata correttamente. Ogni transazione fa parte di un insieme complessivo di attività che costituiscono un processo aziendale che viene eseguito collaborando con i servizi Web. L'intero processo aziendale viene formalmente descritto utilizzando il Business Process Execution Language (BPEL). WS-Coordination è una specifica di accompagnamento che definisce il contesto e il modo esatto in cui le informazioni vengono scambiate durante il processo aziendale. WS-Transaction è uno standard di interoperabilità che include le specifiche WS-AtomicTransaction, WS-BusinessActivity e WS-Coordination.

Il supporto WS-AT (Web Services Atomic Transaction) nel server delle applicazioni fornisce qualità transazionale del servizio all'ambiente dei servizi web. Le applicazioni di servizi Web distribuiti e le risorse che utilizzano possono prendere parte a transazioni globali distribuite. Con il supporto WS-BA (Web Services Business Activity) nel server delle applicazioni, i servizi Web su sistemi diversi possono coordinare attività che sono accoppiate più liberamente rispetto alle transazioni atomiche. Tali attività possono essere difficili o impossibili da ripristinare atomicamente e pertanto richiedono un processo di compensazione in caso di errore. Coordinamento dei servizi Web (WS-COOR) specifica un CoordinationContext e un servizio di registrazione con cui i servizi Web dei partecipanti possono arruolarsi per prendere parte ai protocolli offerti da specifici tipi di coordinamento.

È possibile configurare il sistema in modo da consentire la propagazione dei contesti di messaggi WS-AT (Web Services Atomic Transactions) e di WS-BA (Web Service Business Activities) attraverso i firewall o al di fuori del dominio WebSphere Application Server. Con queste configurazioni, è possibile distribuire applicazioni di servizi Web che utilizzano WS-AT o WS-BA su sistemi diversi.

Un'attività commerciale è una raccolta di attività collegate tra loro in modo che abbiano un risultato concordato. A differenza delle transazioni atomiche, attività come l'invio di un'e-mail possono essere difficili o impossibili da ripristinare atomicamente e pertanto richiedono un processo di compensazione in caso di errore. Il supporto delle attività commerciali di WebSphere Application Server fornisce questa capacità di compensazione attraverso gli ambiti di attività commerciali.

È necessario considerare l'abilitazione e il comportamento del tipo di politica WS-Transaction e il livello di specifica WS-Transaction da utilizzare, quando una cella contiene server in versioni diverse; ad esempio, WebSphere Application Server versione 7.0 o successive e WebSphere Application Server versione 6.1 Feature Pack per i servizi Web.

Utilizzare l'interfaccia di programmazione delle applicazioni di attività commerciali (API) per creare attività commerciali e gestori di compensazione per un componente dell'applicazione e per registrare i dati necessari per compensare un'attività se si verifica un errore nell'attività commerciale complessiva.

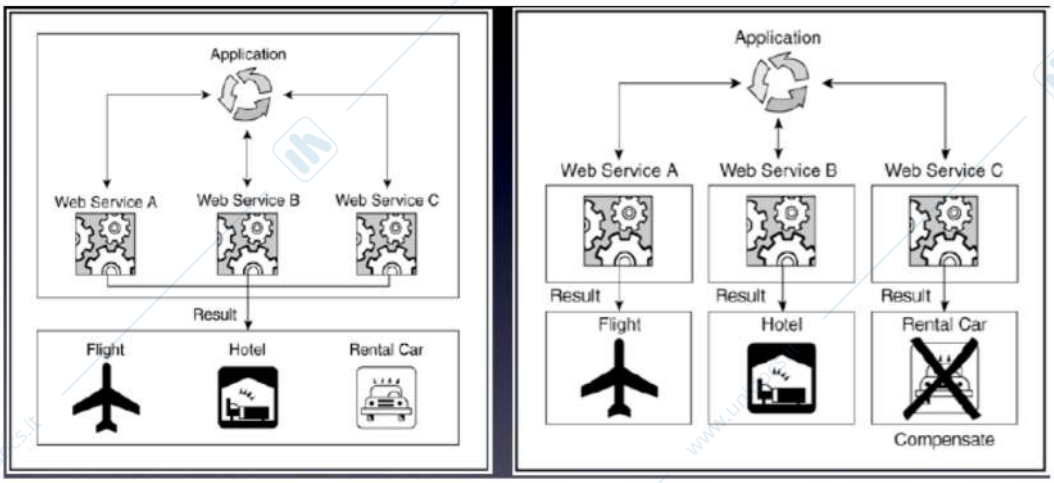

Figura 112: Transazioni classiche e commerciali

"Coordinamento" anziché "Transazione" perché il framework può essere utilizzato per più di semplici transazioni.

WS-Coordination è una specifica di servizi Web sviluppata da BEA Systems, IBM e Microsoft e accettata da OASIS Web Services Transaction TC nella sua versione 1.2. Descrive un framework estensibile per fornire protocolli che coordinano le azioni delle applicazioni distribuite. Tali protocolli di coordinamento sono utilizzati per supportare una serie di applicazioni, comprese quelle che devono raggiungere un accordo coerente sull'esito delle transazioni distribuite.

Il framework definito in questa specifica consente a un servizio applicativo di creare un contesto necessario per propagare un'attività ad altri servizi e registrarsi per i protocolli di coordinamento. Il framework consente l'elaborazione delle transazioni, il flusso di lavoro e altri sistemi di coordinamento esistenti per nascondere i loro protocolli proprietari e operare in un ambiente eterogeneo.

Inoltre WS-Coordination descrive una definizione della struttura del contesto e dei requisiti per la propagazione del contesto tra servizi cooperanti.

Tuttavia, questa specifica non è sufficiente per coordinare le transazioni tra i servizi Web. Fornisce solo un quadro di coordinamento e altre specifiche come WS-Atomic Transaction o WS-BusinessActivity sono necessarie per questo scopo.

CoordinationContext viene utilizzato dalle applicazioni per trasmettere informazioni di coordinamento alle parti coinvolte in un'attività. Gli elementi CoordinationContext vengono propagati alle parti che potrebbero dover registrare i partecipanti per l'attività. La propagazione del contesto può essere realizzata utilizzando meccanismi definiti dall'applicazione, ad es. come elemento di intestazione di un messaggio dell'applicazione SOAP inviato a tali soggetti. (Il trasferimento di un contesto in un messaggio di applicazione viene comunemente definito flusso del contesto.) Un CoordinationContext fornisce l'accesso a un servizio di registrazione del coordinamento, un tipo di coordinamento e le estensioni pertinenti.

Creare attività (coordinate), fornendo una scelta del coordinatore e le opzioni e informazioni per le scelte del protocollo di coordinamento (completamento). Si tratta di un contesto creato, registrazione, con un identificatore univoco per l'attività e un indirizzo del servizio di registrazione con opzioni di protocollo di completamento

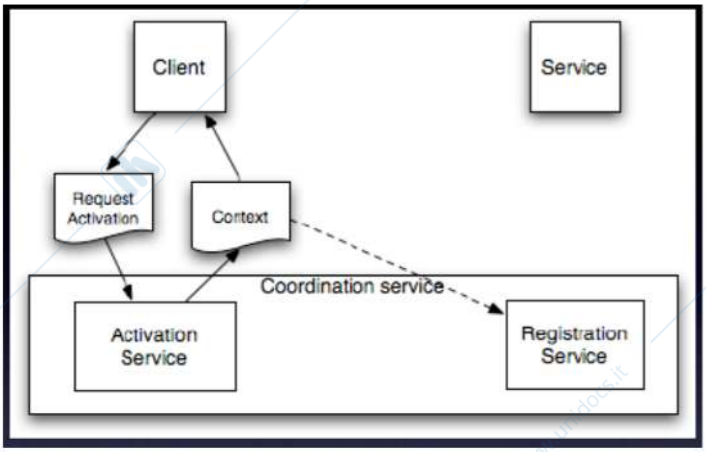

Figura 113: Attivazione del coordinamento

Il contesto viene inviato con la richiesta del Web Service. I registri di servizio (da coinvolgere nel protocollo di completamento) e possono scegliere il protocollo da utilizzare. A seconda del protocollo, il client o il servizio può identificare chi è il coordinatore (endpoint responsabile dell'esecuzione del protocollo di completamento). Al termine, i messaggi inviati tra il coordinatore e i partecipanti registrati.

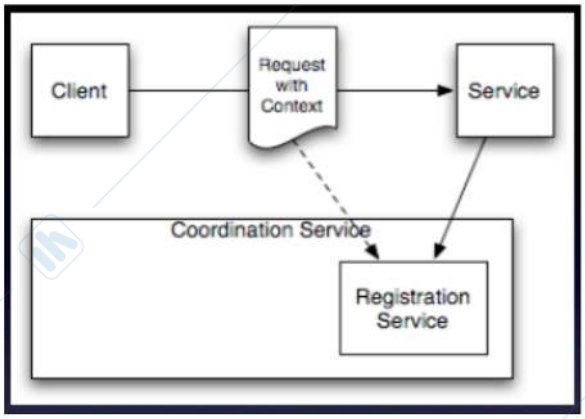

Figura 114: Attivazione del coordinamento

La sicurezza è fondamentale per lo sviluppo di servizi Web reali (controllo dell'accesso, autenticazione semplice per una gamma di servizi distribuiti, trasmissione confidenziale di informazioni sulla sicurezza, non ripudio, ..). Sorge la necessità di supporto di sicurezza specifico per WS:

- Problemi con SSL / TLS e IPSec da soli
- Protocolli generici per sfruttare i concetti di sicurezza esistenti (certificati, ticket Kerberos, CA, ...)
- Esempio che illustra come nascondere, rivelare, autenticarsi con più parti e sicurezza federata (politiche)

A fronte di una richiesta è necessario presentare token di sicurezza (ad esempio: Ticket Kerberos, certificato X509). La presentazione di token nei messaggi e nelle richieste di servizio avviene sfruttando gli standard per XML:

- Crittografia e firma utilizzando gli standard di crittografia XML e firma XML
- SAML - linguaggio di marcatura delle asserzioni di sicurezza
- Modi da includere in SOAP (chiavi e crittografare le informazioni nelle intestazioni, possibilmente parzialmente crittografate / firmate) per l'accesso intermedio

La sicurezza nei web service viene implementata attraverso diverse modalità:

- Generazione e distribuzione di token (fornitore di token di sicurezza come servizio)
- Descrizione delle politiche (ad es. Per inserire WSDL)
- Attività di verifica (controllo / gestione delle politiche)

Alcuni strumenti di sicurezza avanzata (in sviluppo) sono:

- WS-Trust: specifica i modelli per il servizio di fornitore di token di sicurezza
- WS-SecurityPolicy: modalità per esprimere la politica di sicurezza per il framework WS-Policy

- WS-SecureConversation:
  - evitare la PKI per tutti i messaggi
  - analogo a SSL / TLS come livello tra HTTP e TCP
  - scambia chiavi di sessione (usando PKI solo agli end point per impostare)
- WS-Privacy: permette di esprimere politiche che devono essere elaborate da clienti e servizi come parte del framework WS-Policy e limitare il modo in cui i dati possono essere utilizzati
- WS-Federation: permette l'accesso singolo a più imprese che condividono la fiducia dell'identità autenticata.
- WS-Authorization: permette di aggiungere agli standard WS-Trust per l'uso dei token di autorizzazione.

3.2.1 Composizione di servizi Web

Come dovremmo descrivere la combinazione (composizione) di servizi web per creare nuovi servizi? I requisiti per la composizione sono:

- le descrizioni dovrebbero essere relativamente semplici (test della SOA come buon approccio architettonico) e completamente indipendenti dalla piattaforma
- serve una notazione / linguaggio abbastanza formale (eseguibile!)
- necessità di tener conto sia dei servizi astratti che di servizi concreti
- necessità di orchestrare o coreografare le interazioni (descrizione del processo e collaborazioni)
- dovrebbe essere collegato alla teoria dei processi, ad es. Pi Calcolo per comprendere completezza / proprietà / analisi

Un linguaggio descrittivo potrebbe essere usato per descrivere come funziona (viene eseguito) il servizio, tuttavia deve essere implementato. WSDL dice solo come interagire con un servizio. Utilizzando la nuova lingua dovremmo essere in grado di scrivere un'implementazione di un nuovo servizio e descrivere anche come utilizzare i servizi esistenti in una combinazione. Una nuova lingua standardizzata consentirebbe l'indipendenza completa della piattaforma e le possibilità di esecuzione diretta (ovvero non è necessaria la traduzione). BPEL permette di descrivere i processi, attraverso:

- flusso (sequenza, ramificazione, parallelo)
- scambi di messaggi
- struttura nidificata / ricorsiva (processi all'interno di processi)

BPEL può essere implementato in maniera astratta o eseguibile: stesse notazioni (l'abstract può includere dettagli nascosti / non specificati). Eseguibile significa che la descrizione può essere implementata localmente (in alcuni altri PL), o con un motore BPEL (ovvero la notazione è un linguaggio eseguibile portatile). Combinare servizi significa mettere insieme i servizi esistenti anziché implementare localmente i servizi atomici in un

PL convenzionale. BPEL (sigla di Business Process Execution Language) è un linguaggio basato sull'XML costruito per descrivere formalmente i processi commerciali e industriali in modo da permettere una suddivisione dei compiti tra attori diversi. Un'applicazione BPEL viene invocata come Web service e interagisce con il mondo esterno esclusivamente invocando altri Web service. In questo senso, essa stessa rappresenta una forma di coordinazione di servizi Web, permettendo altresì di comporre questi ultimi in maniera ricorsiva. L'ambiente runtime all'interno del quale viene eseguito il generico processo è detto motore BPEL. Il linguaggio BPEL permette di descrivere un processo di business mediante un insieme di attività, che possono essere semplici o strutturate. Le attività semplici esprimono una generica azione (ad es. invoca servizio, ricevi risposta, assegna valore ad una variabile, termina processo, etc...), mentre quelle strutturate sono normalmente utilizzate per raggruppare attività semplici allo scopo di esprimere loop, operazioni condizionali, esecuzione sequenziale, esecuzione concorrente, ecc. L'intero processo è descritto mediante un'unica attività strutturata (top-level activity), generalmente di tipo sequenziale. Un tag "scope" racchiude l'insieme di attività che compone una transazione atomica, ovvero un processo che può terminare con un "commit" o un "abort", senza stati intermedi, nel quale l'arresto del processo su un'attività comporta l'interruzione del processo e la cancellazione delle modifiche in scrittura al database durante le attività precedenti ("undo" delle attività). Questo è necessario ad esempio in una transazione bancaria/finanziaria nella quale ad ogni addebito deve corrispondere un accredito di somme. Un diagramma di workflow contiene tipicamente operazioni, messaggi, attori (umani o applicativi), applicazioni che definiscono il web-service, condizioni logiche (IF), parallelismi, loop e task di sincronizzazione fra operazioni. BPEL è in particolare adatto a modellare workflow completamente automatizzati, per la composizione di Web service l'integrazione di servizi (e applicazioni che li eseguono) eterogenee per hardware che li esegue, architetture di rete e linguaggio del relativo codice. BPEL mette altresì a disposizione dei costrutti per esprimere le cosiddette transazioni di lungo periodo (long running transactions, LRT), che rappresentano un'estensione delle transazioni ACID al caso di processi di lunga durata mediante la nozione di compensazione delle attività eseguite. Ancora, il meccanismo della correlazione è utilizzato per tener traccia di una certa conversazione a livello business, identificando così una sorta di sessione tra più partecipanti ad una stessa istanza di processo. BPEL consente di descrivere un workflow esistente oppure un processo astratto non eseguibile, trasformando in codice di programmazione una modellazione grafica che contiene la semantica descrivibile con i costrutti di UML. Questo è particolarmente utile per far comunicare software proprietari per la modellazione dei processi, che utilizzano terminologie e icone differenti per rappresentare quanto può essere descritto con una notazione UML. BPEL permette di esportare e importare questi diagrammi in un file.bpel da un database proprietario all'altro senza perdere il contenuto della rappresentazione. La "receive" di un messaggio crea un'istanza del processo; istanze del processo differenti variano per il contenuto del messaggi scambiati. Perciò, un campo del messaggio identifica univocamente l'istanza di appartenenza in modo da inviare i corretti dati a ogni istanza di processo. I messaggi sono delle "Input/Output variable" per le quali BPEL crea in automatico il tipo appropriato (stringa, testo, numero), ossia ciò che serve alla persistenza dell'informazione durante l'esecuzione del workflow; messaggi con identico contenuto informativo vengono rappresentati con un'istruzione di "assign" che permette di associare ad una variabile il contenuto di un'altra.

Nel panorama attuale, aumenta sempre di più la propensione a sviluppare degli applicativi che rispettano le direttive della Service Oriented Architecture: l'affermarsi dei

Web Service, infatti, rende più agevole e funzionale lo sviluppo di processi di business, in cui ogni servizio esposto corrisponde a un'attività che può essere messa a disposizione per essere invocata da agenti esterni. In questo modo, è possibile gestire dei workflow operativi, generati da attività atomiche e sapientemente invocati da un unico gestore logico esterno. Si è resa quindi evidente la necessità di mettere a disposizione uno strumento standardizzato che avesse la possibilità di interagire con diversi servizi web, esposti da diversi fornitori, componendoli al fine di creare nuovi servizi dal valore aggiunto. Il coordinamento dei vari servizi può essere realizzato principalmente in due modi:

- Orchestrazione: Presuppone che il controllo del workflow venga mantenuto da un solo gestore logico, che interagisce, anche per processi di lunga durata, con altri servizi, interni od esterni.
- Coreografia: Consiste in un approccio più "collaborativo", ossia ogni partecipante esegue il suo lavoro, e il sistema di coreografia si occupa esclusivamente di tenere traccia delle interazioni avvenute.

Queste due tipologie di architetture garantiscono un accoppiamento debole tra chi richiede e chi eroga il servizio; inoltre in ambedue le tipologie si evidenzia l'impossibilità di tracciare lo stato del servizio: si parla quindi di processi stateless. Per chiarire meglio il concetto di orchestrazione può essere effettuato un paragone con il concetto di workflow. La definizione di un workflow, infatti, si traduce nella definizione del flusso di lavoro che effettua operazioni passando da un task all'altro, gestendo stati e risultati; d'altro canto, la definizione di un processo orchestrato significa introdurre un elemento centrale nel processo, che possiede il controllo del flusso che circola tra i servizi, che quindi possono essere assimilati ai task dello stesso workflow. Il linguaggio BPEL è stato riconosciuto come standard per l'orchestrazione di servizi web, quindi per la definizione dei processi di business: esso, infatti, mette a disposizione diverse funzioni per l'elaborazione dei dati ricevuti dai web service partner del processo, e, tramite funzioni X-Path, esegue operazioni anche complesse su di essi.

Business Process Model and Notation (BPMN) è una rappresentazione grafica per specificare i processi aziendali in un modello di processo aziendale.

4 Architetture

4.1 Caratteristiche architettoniche

Business Process Model and Notation (BPMN) è uno standard per la modellazione dei processi aziendali che fornisce una notazione grafica per specificare i processi aziendali in un Business Process Diagram (BPD), basato su una tecnica di flowcelling molto simile ai diagrammi di attività di Unified Modeling Language (UML). L'obiettivo di BPMN è supportare la gestione dei processi aziendali, sia per gli utenti tecnici che per gli utenti aziendali, fornendo una notazione intuitiva per gli utenti aziendali, ma in grado di rappresentare la semantica dei processi complessi. La specifica BPMN fornisce anche una mappatura tra la grafica della notazione e i costrutti sottostanti dei linguaggi di esecuzione, in particolare Business Process Execution Language (BPEL). BPMN è stato progettato per fornire una notazione standard facilmente comprensibile da tutte le parti interessate, in genere tra cui analisti aziendali, sviluppatori tecnici e dirigenti aziendali. BPMN può quindi essere utilizzato per supportare l'obiettivo generalmente desiderabile

di tutte le parti interessate in un progetto che adotta un linguaggio comune per descrivere i processi, contribuendo a evitare lacune comunicative che possono sorgere tra la progettazione e l'implementazione dei processi aziendali. BPMN è uno dei numerosi standard di linguaggio di modellazione dei processi aziendali utilizzati dagli strumenti e dai processi di modellazione. Mentre l'attuale varietà di lingue può adattarsi a diversi ambienti di modellazione, ci sono quelli che sostengono lo sviluppo o l'emergere di un unico standard globale, che combina i punti di forza di diverse lingue esistenti. Si suggerisce che, nel tempo, ciò potrebbe contribuire a unificare l'espressione dei concetti di base dei processi aziendali (ad es. Processi pubblici e privati, coreografie), nonché dei concetti di processo avanzati (ad es. Gestione delle eccezioni, compensazione delle transazioni). Sono stati sviluppati due nuovi standard, usando un approccio simile a BPMN, che si occupano di modellizzazione della gestione dei casi (modello e notazione della gestione dei casi) e modellizzazione delle decisioni, il (modello decisionale e notazione).

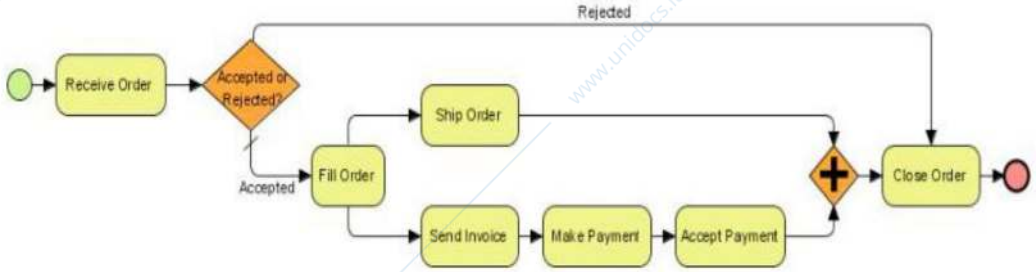

Figura 115: Esempio di BPMN

La Figura 116 riporta gli elementi grafici in BPMN.

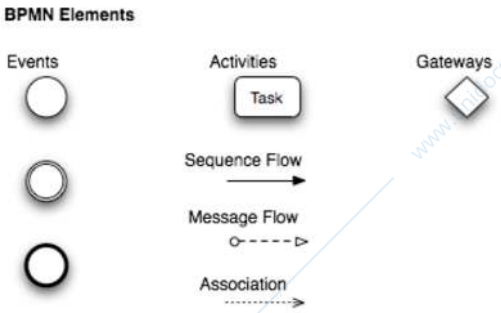

Figura 116: Elementi di BPMN

Riassumendo, l'orchestrazione riguarda la descrizione e l'esecuzione di un modello a punto di vista singolo. La coreografia riguarda la descrizione e la guida di un modello globale. É possibile derivare il modello del punto di vista singolo dal modello globale proiettando in base al partecipante.

5 OAuth 2.0

OAuth è un protocollo di rete aperto e standard, progettato specificamente per lavorare con l'Hypertext Transfer Protocol (HTTP). Essenzialmente consente l'emissione di un

token di accesso da parte di un server autorizzativo ad un client di terze parti, previa approvazione dell'utente proprietario della risorsa cui si intende accedere. Questo meccanismo, detto di "access delegation" (delega di accesso), è utilizzato da compagnie quali: Amazon, Google, Facebook, Microsoft, Twitter per consentire all'utente di condividere informazioni circa il proprio account con altre applicazioni o siti web. È comunemente utilizzato come modalita' per gli utenti in Internet per garantire alle applicazioni ed ai siti web l'accesso alle proprie informazioni senza fornire loro alcuna password. Tale protocollo permette l'autorizzazione di API di sicurezza con un metodo standard e semplice in varie situazioni: applicazioni "mobile", applicazioni "web", applicazioni per personal computer. Per gli sviluppatori di applicazioni è un metodo per pubblicare e interagire con dati protetti. OAuth garantisce ai service provider l'accesso da parte di terzi ai dati degli utenti proteggendo contemporaneamente le loro credenziali. Per esempio permette all'utente di dare ad un sito, chiamato consumer, l'accesso alle informazioni presenti su un altro sito, detto service provider, senza condividere la propria identità.

OAuth 2 è un framework di autorizzazione che consente alle applicazioni di ottenere un accesso limitato agli account utente su un servizio HTTP, come Facebook, GitHub e DigitalOcean. Funziona delegando l'autenticazione dell'utente al servizio che ospita l'account utente e autorizzando le applicazioni di terzi ad accedere all'account utente. OAuth 2 fornisce flussi di autorizzazione per applicazioni Web e desktop e dispositivi mobili.

OAuth definisce quattro ruoli:

- l' "Utente" o "Resource Owner": la persona che può dare l'accesso ad una qualche porzione del proprio account. È la persona che può garantire l'accesso alle risorse protette presenti sul "Resource Server".
- il "Client" o "Consumer": l'applicazione che cerca di ottenere l'accesso all'account dell'utente e che richiede il permesso dell' "Utente" prima di poter accedere alle risorse protette.
- il "Resource Server" o "Service Provider": il server utilizzato per accedere alle risorse protette dell'utente.
- l' "Authorization Server": il server che presenta l'interfaccia dove l' "Utente" approva o nega la richiesta di accesso. In piccoli ambienti può coincidere con il "Resource Server".

Il "Client" interagisce con il "Resource Server" per ottenere delle credenziali temporanee. Per accedere alle risorse protette, il "Client" richiede all' "Utente" il permesso, detto token di accesso. Una volta ottenuto il token di accesso può accedere e interagire con le risorse stabilite per un breve periodo.

Il proprietario della risorsa è l'utente che autorizza un'applicazione ad accedere al proprio account. L'accesso dell'applicazione all'account dell'utente è limitato all'ambito di applicazione dell'autorizzazione concessa (ad es. Accesso in lettura o scrittura).

Il server risorse ospita gli account utente protetti e il server delle autorizzazioni verifica l'identità dell'utente, quindi emette token di accesso all'applicazione. Dal punto di vista di uno sviluppatore di applicazioni, l'API di un servizio può soddisfare sia i ruoli del server di risorse che quelli di autorizzazione. Faremo riferimento a entrambi questi ruoli combinati, come ruolo Servizio / API.

Il client è l'applicazione che desidera accedere all'account dell'utente. Prima di poterlo fare, deve essere autorizzato dall'utente e l'autorizzazione deve essere convalidata dall'API.

La Figura 117 riporta il flusso del protocollo.

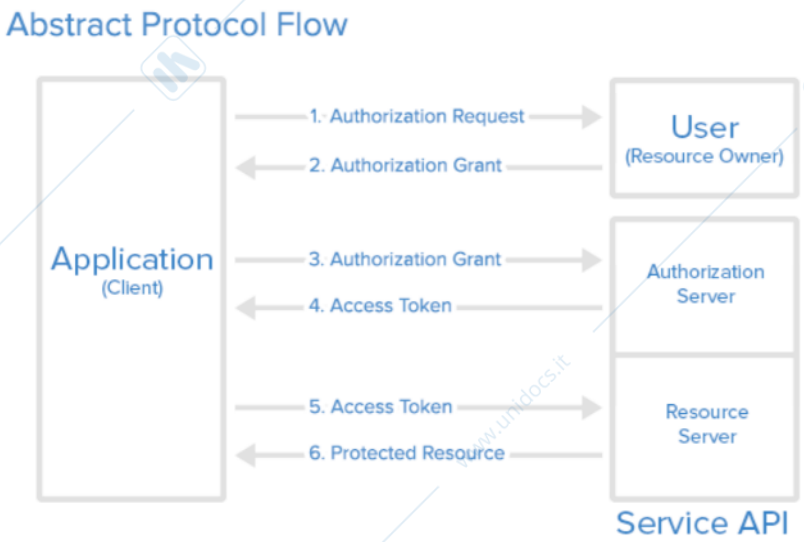

Figura 117: Flusso del protocollo

L'applicazione richiede l'autorizzazione per accedere alle risorse del servizio da parte dell'utente. Se l'utente ha autorizzato la richiesta, la domanda riceve una concessione di autorizzazione. L'applicazione richiede un token di accesso al server di autorizzazione (API) presentando l'autenticazione della propria identità e la concessione di autorizzazione. Se l'identità dell'applicazione è autenticata e la concessione dell'autorizzazione è valida, il server delle autorizzazioni (API) emette un token di accesso all'applicazione. L'autorizzazione è completa. L'applicazione richiede la risorsa dal server delle risorse (API) e presenta il token di accesso per l'autenticazione. Se il token di accesso è valido, il server delle risorse (API) serve la risorsa all'applicazione. Il flusso effettivo di questo processo differirà a seconda del tipo di concessione di autorizzazione in uso.

Prima di utilizzare OAuth con l'applicazione, è necessario registrare l'applicazione con il servizio. Ciò avviene tramite un modulo di registrazione nella parte "sviluppatore" o "API" del sito Web del servizio, in cui fornirai le seguenti informazioni (e probabilmente i dettagli sulla tua applicazione):

- Nome dell'applicazione
- Sito Web dell'applicazione
- Reindirizzamento o URL di richiamata
- Il callback è il punto in cui il servizio reindirizzerà l'utente dopo aver autorizzato (o rifiutato) l'applicazione e quindi la parte dell'applicazione che gestirà i codici di autorizzazione o i token di accesso.

Una volta registrata l'applicazione, il servizio emetterà le "credenziali del client" sotto forma di un identificativo del cliente e un segreto del cliente. L'ID client è una stringa

esposta pubblicamente che viene utilizzata dall'API del servizio per identificare l'applicazione e viene anche utilizzata per creare URL di autorizzazione presentati agli utenti. Il segreto client viene utilizzato per autenticare l'identità dell'applicazione nell'API di servizio quando l'applicazione richiede di accedere all'account di un utente e deve essere mantenuta privata tra l'applicazione e l'API.

Nel Abstract Protocol Flow, i primi quattro passaggi riguardano l'ottenimento di una concessione di autorizzazione e un token di accesso. Il tipo di concessione dell'autorizzazione dipende dal metodo utilizzato dall'applicazione per richiedere l'autorizzazione e dai tipi di concessione supportati dall'API. OAuth 2 definisce quattro tipi di concessione, ognuno dei quali è utile in diversi casi:

- Codice di autorizzazione: utilizzato con le applicazioni lato server
- Implicito: utilizzato con app per dispositivi mobili o applicazioni Web (applicazioni eseguite sul dispositivo dell'utente)
- Credenziali password proprietario risorsa: utilizzate con applicazioni attendibili, come quelle di proprietà del servizio stesso
- Credenziali client: utilizzate con l'accesso all'API delle applicazioni

Il tipo di concessione del codice di autorizzazione è il più comunemente utilizzato poiché è ottimizzato per le applicazioni lato server, in cui il codice sorgente non è esposto pubblicamente e la riservatezza del segreto client può essere mantenuta. Questo è un flusso basato sul reindirizzamento, il che significa che l'applicazione deve essere in grado di interagire con l'agente utente (ovvero il browser Web dell'utente) e di ricevere i codici di autorizzazione API instradati attraverso l'agente utente.

Innanzitutto, all'utente viene fornito un collegamento al codice di autorizzazione simile al a seguire: https://cloud.digitalocean.com/v1/oauth/authorize?response_type=code& client_id = CLIENT_ID & redirect_uri = CALLBACK_URL & scope = lettura. Ecco una spiegazione dei componenti del collegamento:

- https://cloud.digitalocean.com/v1/oauth/authorize: l'endpoint di autorizzazione API
- client_id = client_id: l'ID client dell'applicazione (come l'API identifica l'applicazione)
- redirect_uri = CALLBACK_URL: dove il servizio reindirizza l'agente utente dopo la concessione di un codice di autorizzazione
- response_type = code: specifica che l'applicazione richiede una concessione del codice di autorizzazione
- scope = read: specifica il livello di accesso richiesto dall'applicazione

Quando l'utente fa clic sul collegamento, deve prima accedere al servizio, per autenticare la sua identità (a meno che non abbia già effettuato l'accesso). Quindi le verrà richiesto dal servizio di autorizzare o negare l'accesso dell'applicazione al proprio account. La Figura 118 un esempio del prompt di autorizzazione dell'applicazione.

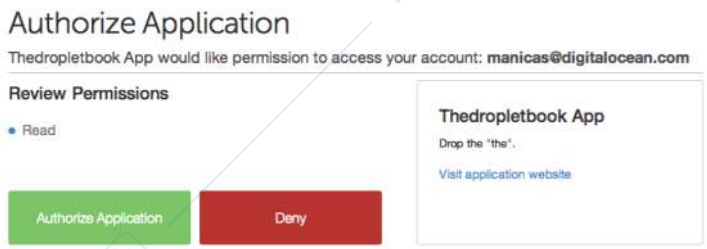

Figura 118: Prompt di autorizzazione dell'applicazione

Se l'utente fa clic su "Autorizza applicazione", il servizio reindirizza l'utente all'agente di reindirizzamento URI, che è stato specificato durante la registrazione del client, insieme a un codice di autorizzazione. Il reindirizzamento sarebbe simile a questo (supponendo che l'applicazione sia "dropletbook.com"): https://dropletbook.com/callback?code=AUTHORIZATION_CODE.

L'applicazione richiede un token di accesso dall'API, passando il codice di autorizzazione insieme ai dettagli di autenticazione, incluso il segreto del client, all'endpoint del token API. Ecco un esempio di richiesta POST HTTP all'endpoint token di DigitalOcean:

```
https://cloud.digitalocean.com/v1/oauth/token?client_id=
CLIENT_ID&client_secret=CLIENT_S
```

Figura 119: HTTP POST request to DigitalOcean's token endpoint

Se l'autorizzazione è valida, l'API invierà all'applicazione una risposta contenente il token di accesso (e facoltativamente un token di aggiornamento). L'intera risposta sarà simile a quella presentata nella Figura 120.

```
{"access_token":"ACCESS_TOKEN","token_type":"bearer","exp
ires_in":2592000,"refresh_token":"REFRESH_TOKEN","scope":
"read","uid":100101,"info":{"name":" E.
Dam","email":"edam@mac.com"}}
```

Figura 120: Risposta dell'API

Ora l'applicazione è autorizzata. Può utilizzare il token per accedere all'account dell'utente tramite l'API del servizio, limitato all'ambito di accesso, fino alla scadenza o alla revoca del token. Se è stato emesso un token di aggiornamento, può essere utilizzato per richiedere nuovi token di accesso se il token originale è scaduto.

Il tipo di concessione implicita viene utilizzato per le app mobili e le applicazioni Web (ovvero le applicazioni eseguite in un browser Web), in cui la riservatezza segreta del client non è garantita. Il flusso di concessione implicito funziona sostanzialmente come segue: all'utente viene richiesto di autorizzare l'applicazione, quindi il server di autorizzazione passa il token di accesso all'agente utente, che lo passa all'applicazione. Alcuni Commenti:

- Anche il tipo di concessione implicita si basa sul reindirizzamento, ma il token di accesso viene fornito all'utente-agente per inoltrare all'applicazione, quindi potrebbe essere esposto all'utente e ad altre applicazioni sul dispositivo dell'utente.

- Inoltre, questo flusso non autentica l'identità dell'applicazione e si basa sull'URI di reindirizzamento (che è stato registrato con il servizio) per servire a questo scopo.
- Il tipo di concessione implicita non supporta i token di aggiornamento.

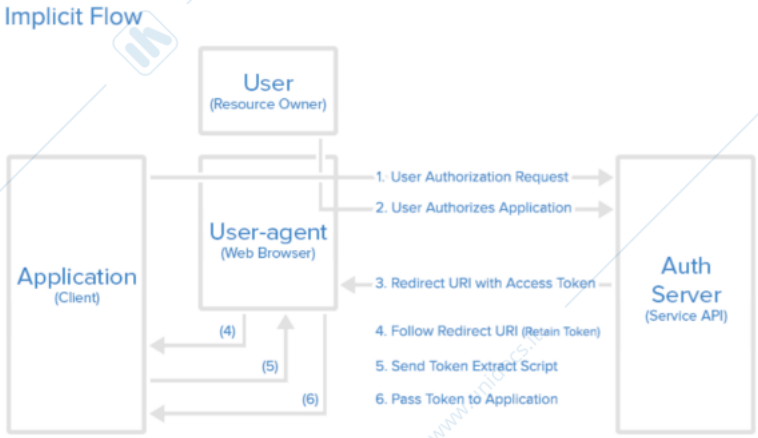

Figura 121: Flusso implicito

Con il tipo di concessione delle credenziali della password del proprietario della risorsa, l'utente fornisce le proprie credenziali del servizio (nome utente e password) direttamente all'applicazione, che utilizza le credenziali per ottenere un token di accesso dal servizio. Questo tipo di concessione deve essere abilitato sul server di autorizzazione solo se altri flussi non sono vitali. Inoltre, dovrebbe essere utilizzato solo se l'applicazione è considerata attendibile dall'utente (ad esempio, è di proprietà del servizio o del sistema operativo desktop dell'utente).

Dopo che l'utente ha fornito le proprie credenziali all'applicazione, l'applicazione richiederà quindi un token di accesso dal server di autorizzazione. La richiesta POST potrebbe assomigliare a quella in Figura 122.

```
https://oauth.example.com/token?grant_type=password&use
rname=USERNAME&password=PASSWORD&client_id=CLIENT_ID
```

Figura 122: Richiesta POST

Se le credenziali dell'utente vengono verificate, il server delle autorizzazioni restituisce un token di accesso all'applicazione.

Il tipo di concessione delle credenziali del client fornisce a un'applicazione un modo per accedere al proprio account di servizio. Esempi di quando ciò potrebbe essere utile includono se un'applicazione desidera aggiornare la sua descrizione registrata o reindirizzare l'URI o accedere ad altri dati memorizzati nel suo account di servizio tramite l'API.

L'applicazione richiede un token di accesso inviando le proprie credenziali, l'ID client e il segreto client al server di autorizzazione. Una richiesta POST di esempio potrebbe essere simile alla Figura 123.

```
https://oauth.example.com/token?grant_type=client_credentials&client_id=CLIENT_ID&client_secret=CLIENT_SECRET
```

Figura 123: Richiesta POST

Se le credenziali dell'applicazione vengono verificate, il server delle autorizzazioni restituisce un token di accesso all'applicazione.

```
<Policy PolicyId="SamplePolicy"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">

  <!-- This Policy only applies to requests on the SampleServer -->
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">SampleServer</AttributeValue>
        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                                     AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
      </ResourceMatch>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>

  <!-- Rule to see if we should allow the Subject to login -->
  <Rule RuleId="LoginRule" Effect="Permit">

    <!-- Only use this Rule if the action is login -->
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">login</AttributeValue>
          <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                                     AttributeId="ServerAction"/>
```

Figura 124: Esempio policy (1)

```
        </ActionMatch>
      </Actions>
    </Target>

    <!-- Only allow logins from 9am to 5pm -->
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
          <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
          <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
      </Apply>
    </Condition>

  </Rule>

  <!-- We could include other Rules for different actions here -->

  <!-- A final, "fall-through" Rule that always Denies -->
  <Rule RuleId="FinalRule" Effect="Deny"/>

</Policy>
```

Figura 125: Esempio policy (2)