APPUNTI PER LA PARTE DI TEORIA

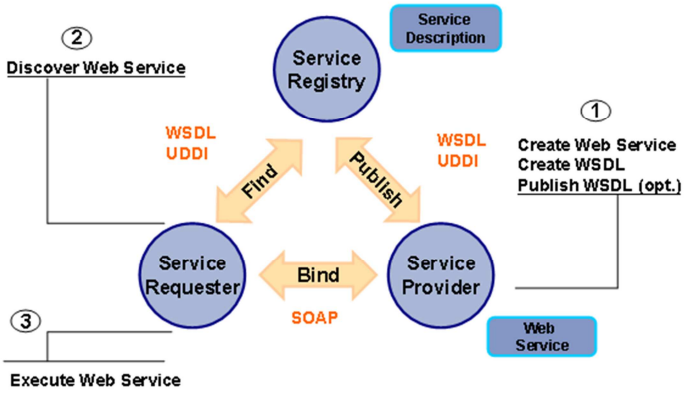
CORSO: PROGRAMMAZIONE SU RETI

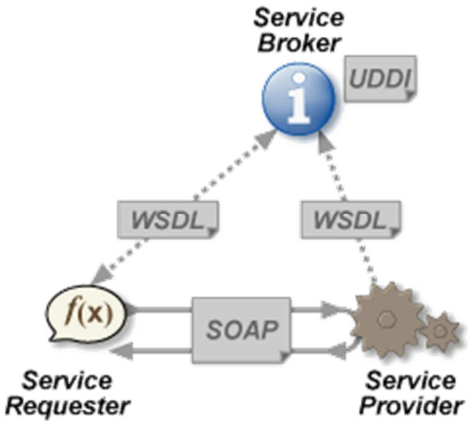
DOCENTE CORSO: VITTORIO SCARANO

DOCENTE LAB: DELFINA MALANDRINO

STUDENTE: NARDI FABIO

ANNO CORSO: 2012/2013





SISTEMI DISTRIBUITI

UN SISTEMA DISTRIBUITO È CARATTERIZZATO DA UNA COLLEZIONE DI COMPUTER, EVENTUALMENTE ETEROGENEI, COLLEGATI IN RETE CHE COMUNICANO E COORDINANO LE LORO ATTIVITÀ SCAMBIANDOSI MESSAGGI.

LA CARATTERISTICA CHE DISTINGUE UN SISTEMA DISTRIBUITO DA UNA NORMALE APPLICAZIONE DI RETE È CHE L'UTENTE CHE UTILIZZA UN SISTEMA DISTRIBUITO LO VEDE COME SE FOSSE UN'UNICA ENTITÀ FUNZIONALE, MENTRE L'UTENTE DI UN'APPLICAZIONE DI RETE È CONSCIO DELLA PRESENZA DI VARIE ENTITÀ.
UNA DELLE CARATTERISTICHE DI UN SISTEMA DISTRIBUITO È CHE AL SUO INTERNO I PROCESSI NON SONO ESEGUITI SU UNA SINGOLA MACCHINA MA POSSONO COINVOLGERE PIÙ MACCHINE. QUESTO IMPLICA CHE UN QUALCHE MECCANISMO DI COMUNICAZIONE INTERPROCESSO DEVE ESSERE INTRODOTTO IN MODO DA CONSENTIRE L'INTERAZIONE TRA PROCESSI IN ESECUZIONE SU MACCHINE DIFFERENTI.
IL CANALE DI COMUNICAZIONE CHE CONSENTE ALLE VARIE COMPONENTI DI UN SISTEMA DISTRIBUITO DI INTERAGIRE SONO I PROTOCOLLI INTERNET. QUESTI PROTOCOLLI CONSENTONO IL TRASPORTO DI DATI TRA DUE HOST, INDIPENDENTEMENTE DALLA LORO LOCALIZZAZIONE.
I PROTOCOLLI INTERNET DEFINISCONO UN STANDARD ATTRAVERSO I QUALI I COMPONENTI DI UN SISTEMA DISTRIBUITO COMUNICANO, SPECIFICANDO IL FORMATO E L'ORDINE DEI MESSAGGI SCAMBIATI E STABILENDO LE AZIONI CHE I COMPONENTI DEVONO ESEGUIRE PER TRASMETTERE/RICEVERE UN MESSAGGIO. UN'APPLICAZIONE PUÒ UTILIZZARE DIRETTAMENTE I PROTOCOLLI INTERNET UTILIZZANDO API DI PROGRAMMAZIONE COME QUELLA DEI SOCKET CHE PERMETTONO DI INTERAGIRE DIRETTAMENTE CON IL SISTEMA OPERATIVO PER COMUNICARE CON UN ALTRO PROCESSO IN ESECUZIONE SU DI UN'ALTRA MACCHINA. I SOCKET CONSENTONO DI INSTAURARE UNA CONNESSIONE TCP CON UN DETERMINATO PROCESSO E LEGGERE/SCRIVERE FLUSSI DI BYTE DA UN ALTRO PROCESSO. PURTROPPO LAVORARE A LIVELLO DI SOCKET È COMPLICATO E PREDISPOSTO AGLI ERRORI PERCHÉ IL PROGRAMMATORE DEVE GESTIRE TOTALMENTE LA TRASMISSIONE CON L'ALTRO PROCESSO (GARANTIRE CHE I DATI VENGANO RICEVUTI CORRETTAMENTE E NEL GIUSTO ORDINE PER ES).
PER QUESTO MOTIVO ABBIAMO BISOGNO DI UN'ASTRAZIONE DI PIÙ ALTO LIVELLO CHE PERMETTA DI NASCONDERE TUTTI I DETTAGLI RELATIVI ALL'INTERAZIONE TRA COMPONENTI ETEROGENEE.

MIDDLEWARE

IL MIDDLEWARE È UN'ASTRAZIONE FONDAMENTALE PER COSTRUIRE LA COMPLESSITÀ E L'ETEROGENEITÀ PRESENTI IN UN SISTEMA DISTRIBUITO. POSSIAMO DEFINIRE IL MIDDLEWARE COME UN SOFTWARE DI CONNETTIVITÀ CHE PERMETTE DI CREARE UN PONTE TRA DUE APPLICAZIONI DIFFERENTI, CONSENTENDO LORO DI COMUNICARE E DI SCAMBIARSI DATI INDIPENDENTEMENTE DALLE PIATTAFORME HARDWARE/SOFTWARE SU CUI OPERANO.

IL COMPITO DEL MIDDLEWARE È DI FACILITARE LA PROGETTAZIONE, LA REALIZZAZIONE E LA GESTIONE DI APPLICAZIONI DISTRIBUITE FORNENDO UN UNICO AMBIENTE DI PROGRAMMAZIONE DISTRIBUITO. POSSIAMO VEDERE IL MIDDLEWARE COME UNO STRATO SOFTWARE POSIZIONATO TRA L'APPLICAZIONE E IL SISTEMA OPERATIVO, FORNENDO ALL'APPLICAZIONE UNA SERIE DI API PIÙ FUNZIONALI DI QUELLE FORNITE DAL SISTEMA OPERATIVO, IN MODO DA NASCONDERE LA COMPLESSITÀ E L'ETEROGENEITÀ DELL'AMBIENTE DISTRIBUITO SOTTOSTANTE. TRA LE ALTRE COSE, UN MIDDLEWARE CONSENTE DI:

- LOCALIZZARE LE APPLICAZIONI SULLA RETE IN MANIERA TRASPARENTE ALL'UTENTE (L'UTENTE CONOSCE IL NOME DEL SERVIZIO CHE VUOLE UTILIZZARE MA NON L'INDIRIZZO DELL'APPLICAZIONE CHE LO FORNISCE)
- NASCONDERE AL PROGRAMMATORE TUTTI I DETTAGLI A BASSO LIVELLO DELLA PROGRAMMAZIONE CON I SOCKET.
- FORNIRE UN INSIEME DI ASTRAZIONI DI ALTO LIVELLO PIÙ VICINE ALLE RICHIESTE DELL'APPLICAZIONE.
- FORNIRE UNA SERIE DI SERVIZI QUALI L'AFFIDABILITÀ, LA SICUREZZA, L'AUTENTICAZIONE.

POSSIAMO VEDERE IL MIDDLEWARE COME COSTITUITO DA DUE STRATI SOFTWARE: UNO STRATO A PIÙ BASSO LIVELLO (CHE INTERAGISCE CON IL SISTEMA OPERATIVO), ED UNO STRATO PIÙ ALTO (CHE INTERAGISCE CON L'APPLICAZIONE).

LO STRATO INFERIORE, DETTO FORNITORE I MECCANISMI DI COMUNICAZIONE INTERPROCESSO CHE CONSENTONO A DUE O PIÙ PROCESSI DI COMUNICARE, INDIPENDENTEMENTE DALLA LORO PIATTAFORMA HARDWARE/SOFTWARE, E DEVE OCCUPARSI DELLA RAPPRESENTAZIONE DEI DATI ALL'INTERNO DEI MESSAGGI SCAMBIATI.
LO STRATO SUPERIORE, FORNISCE SERVIZI DI ALTO LIVELLO ALL'APPLICAZIONE. DISTINGUIAMO QUESTI SERVIZI IN DUE CATEGORIE: SERVIZI NON ORIENTATI AI MESSAGGI (CHE FORNISCONO UN MODELLO DI COMUNICAZIONE SINCRONO, EX MODELLI CLIENT-SERVER), E SERVIZI ORIENTATI AI MESSAGGI (CHE FORNISCONO SERVIZI DI COMUNICAZIONE ASINCRONA E NOTIFICA DI EVENTI).
IL PIÙ SEMPLICE MECCANISMO DI COMUNICAZIONE INTERPROCESSO È FORNITO DALL'ASTRAZIONE DEL MESSAGE PASSING: DUE PROCESSI IN ESECUZIONE SU DUE HOST DIFFERENTI COMUNICANO SCAMBIANDOSI MESSAGGI ATTRAVERSO UNA RETE.
UN MESSAGGIO È COSTITUITO DA 3 ELEMENTI:
→ UN'INTESTAZIONE (HEADER): VIENE USATO DAL SISTEMA DI MESSAGING O DALL'APPLICAZIONE PER FORNIRE INFORMAZIONI SUL MESSAGGIO (EX: DESTINAZIONE O DATA DI SCADENZA)
→ UNA LISTA DI PROPRIETÀ: CONTIENE UNA LISTA DI COPPIE NOME/VALORE, CHE POSSONO ESSERE UTILIZZATE PER EFFETTUARE OPERAZIONI DI FILTRO DA PARTE DEL DESTINATARIO/INTERMEDIARI CHE SI TROVANO A PROCESSARE IL MESSAGGIO.
→ UN CORPO (PAYLOAD): CONTIENE L'EFFETTIVO CONTENUTO DEL MESSAGGIO (FORMATO DEI DOCUMENTI XML)

LO SCAMBIO DI MESSAGGI TRA DUE PROCESSI VIENE SUPPORTATO DA 2 OPERAZIONI: SEND E RECEIVE.
I DATI MEMORIZZATI NELLE APPLICAZIONI CHE FORMANO UN SISTEMA DISTRIBUITO, SONO RAPPRESENTATI COME STRUTTURE DATI, MENTRE I MESSAGGI SCAMBIATI SONO UNA SEQUENZA DI BYTE. PER QUESTO MOTIVO È NECESSARIO PER MITTENTE TRASFORMARE I DATI ORIGINARI IN UNA SEQUENZA DI BYTE DA INSERIRE NEL MESSAGGIO E PER DESTINATARIO LEGGERE LA SEQUENZA DI BYTE E RICOSTRUIRE I DATI DA PASSARE ALL'APPLICAZIONE. QUESTE OPERAZIONI SONO DETTE MARSHALLING ED UNMARSHALLING, MA IN JAVA/XML SI USANO I TERMINI DI SERIALIZZAZIONE E DESERIALIZZAZIONE.
SUL MERCATO ESISTONO 2 DIVERSI MIDDLEWARE: -SINCRONA (DIPENDENTE DAL TEMPO)
(A SECONDA DEL TIPO DI COMUNICAZIONE) -ASINCRONA (INDIPENDENTE DAL TEMPO)
LA CARATTERISTICA DI UNA COMUNICAZIONE SINCRONA, È CHE I DUE PROCESSI CHE PRENDONO PARTE ALLA COMUNICAZIONE DEVONO ESSERE ENTRAMBI, CONTEMPORANEAMENTE, DISPONIBILI AD EFFETTUARE LA COMUNICAZIONE. IL CLIENT INVOCA UNA FUNZIONE PER RICHIEDERE QUALCOSA AL SERVER E LA SUA ESECUZIONE RIMANE BLOCCATA FINCHÉ NON RICEVE LA RISPOSTA. LA FORMA PIÙ DIFFUSA DI QUESTO TIPO DI COMUNICAZIONE È DATA DA RPC.
LA CARATTERISTICA DI UNA COMUNICAZIONE ASINCRONA, È CHE IL MITTENTE UTILIZZA UN APPROCCIO DI TIPO SEND AND FORGET: IL MITTENTE SPEDISCE IL MESSAGGIO E POI RIPRENDE A SVOLGERE LA PROPRIA ATTIVITÀ SENZA ASPETTARE L'EVENTUALE RISPOSTA. QUINDI LE DUE APPLICAZIONI NON DEVONO ESSERE CONTEMPORANEAMENTE ATTIVE E DISPONIBILI. I SISTEMI DI MESSAGING ASINCRONO SONO IMPLEMENTATI CON QUALCHE MECCANISMO DI ACCODAMENTO (I PIÙ DIFFUSI SONO STORE AND FORWARD/PUBLISH-SUBSCRIBE).

COMUNICAZIONE DI MESSAGGI SINCRONA: REMOTE PROCEDURE CALL

I MODELLI DI PROGRAMMAZIONE PER MIDDLEWARE CON COMUNICAZIONE SINCRONA PREVEDONO PROCESSI CHE GIRANO SU MACCHINE DIFFERENTI E CHE COOPERANO TRA LORO. GLI APPROCCI PIÙ DIFFUSI SONO LE REMOTE PROCEDURE CALL (RPC) E LE REMOTE METHOD INVOCATION (RMI).

L'ASTRAZIONE DELLE RPC, CONSENTE AD UN CLIENT DI INVOCARE UNA FUNZIONE CHE VERRÀ ESEGUITA IN REMOTO COME SE FOSSE UNA FUNZIONE LOCALE.

IL FUNZIONAMENTO DI RPC È BASATO SU DI UN PROTOCOLLO CHE DEFINISCE IL FORMATO DEI MESSAGGI DI RICHIESTA, CHE DEVONO SPECIFICARE IL NOME DELLA FUNZIONE DA INVOCARE E GLI ARGOMENTI DA PASSARE NELLA CHIAMATA, E DEI MESSAGGI DI RISPOSTA CHE CONTENGONO I VALORI RESTITUITI DALLA FUNZIONE.

L'INTERAZIONE CON IL PROTOCOLLO RPC AVVIENE TRAMITE UN ADATTATORE (STUB) CHE ESPONE UN'INTERFACCIA ANALOGA A QUELLA DELLA FUNZIONE DA INVOCARE.

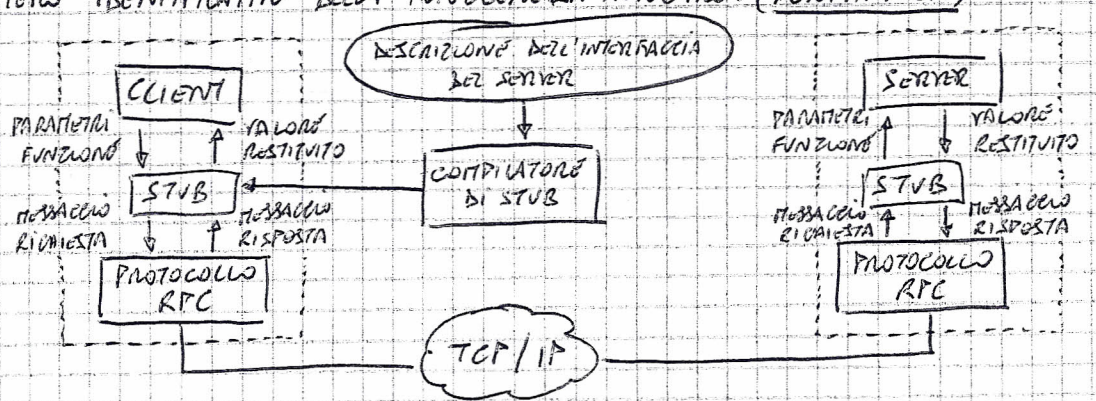
IL PROTOCOLLO RPC DEVE COSTARE 3 PROBLEMI:

⊖ TRASMISSIONE DI GROSSE QUANTITÀ DI DATI (USA UNA TECNICA DI FRAMMENTAZIONE E RIASSEMBLAGGIO SIMILE A QUELLA UTILIZZATA DA IP)

⊖ GARANTIRE L'AFFIDABILITÀ DELLA COMUNICAZIONE E SINCRONIZZAZIONE DEI MESSAGGI DI RICHIESTA E RISPOSTA (USA TECNICHE SIMILI A QUELLE UTILIZZATE DA TCP)

⊖ INDIVIDUARE IL PROCESSO E LA FUNZIONE REMOTA DA INVOCARE. (IN GENERE IL PROCESSO È INDIVIDUATO DAL PROTOCOLLO DI TRASPORTO CON IL NUMERO DI PORTA, LA FUNZIONE È INDIVIDUATA TRAMITE UN SISTEMA DI INDIRIZZAMENTO)

LA SOLUZIONE PIÙ ADOTTATA PER RISOLVERE IL PROBLEMA DELL'INDIRIZZAMENTO È DI COSTRUIRE UN SERVIZIO DI MAPPING CHE GIRA SULLA MACCHINA DEL SERVER E CHE RICEVE IN INPUT IL NOME DEL SERVIZIO DA INVOCARE E RESTITUISCE IL NUMERO DI PORTA DEL PROCESSO DA CONTATTARE ED IL NUMERO IDENTIFICATIVO DELLA FUNZIONE DA INVOCARE. (PORTMAPPER)



L'EVOLUZIONE DI RPC È RMI (REMOTE METHOD INVOCATION), CHE CONSENTE AD UN'APPLICAZIONE JAVA DI INVOCARE OGGETTI JAVA IN ESECUZIONE IN UN PROCESSO REMOTO COME SE FOSSE UN OGGETTO LOCALE.

COMUNICAZIONE DI MESSAGGI ASINCRONA: MESSAGING

QUESTO APPROCCIO PREVEDE CHE LE APPLICAZIONI OPERINO INDIPENDENTEMENTE E SI SCAMBIANO DEI DATI ATTRAVERSO MESSAGGI ASINCRONI. LA CARATTERISTICA PRINCIPALE DI TALE MODELLO È LA COMPLETA LIBERTÀ DI DEFINIRE E RAPPRESENTARE NUOVE STRUTTURE DATI. INFATTI UNO DEI PUNTI DEBOLI DEGLI APPROCCI TRADIZIONALI ALLA COMPUTAZIONE DISTRIBUITA È LA CODIFICA DEI DATI. I SISTEMI BASATI SU MESSAGING HANNO TROVATO, IN XML UNO STRUMENTO POTENTE PER RAPPRESENTARE LE INFORMAZIONI. CON XML TUTTE LE INFORMAZIONI SONO RAPPRESENTATE IN FORMATO TESTUALE, ELIMINANDO I PROBLEMI DI CONVERSIONE DEI FORMATI BINARI.

I MIDDLEWARE CHE SUPPORTANO FORME DI COMUNICAZIONE ASINCRONA SONO BASATI SU MECCANISMI DI ACCODAMENTO. I DUE MODELLI DI ACCODAMENTO PIÙ UTILIZZATI SONO:

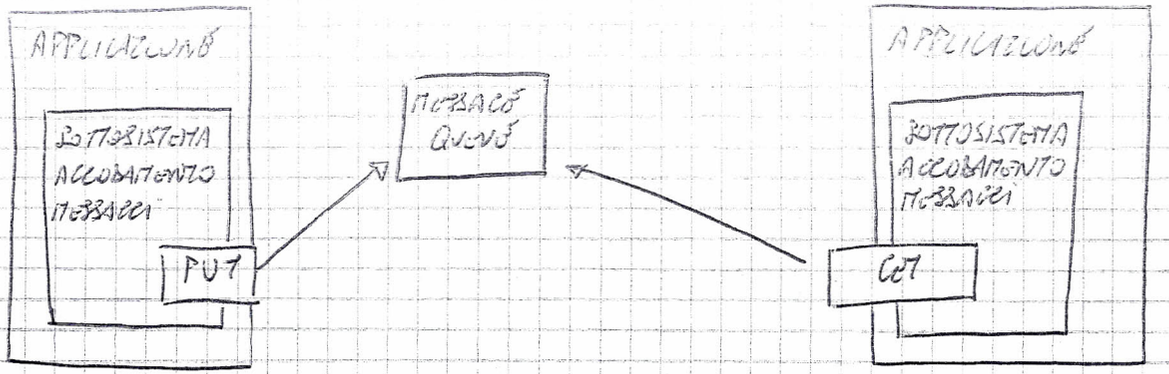
- STORE-AND-FORWARD MESSAGING
- PUBLISH/SUBSCRIBE MESSAGING

STORE-AND-FORWARD MESSAGING

IN UN MECCANISMO DI ACCODAMENTO STORE-AND-FORWARD LO SCAMBIO DI MESSAGGI AVVIENE TRAMITE UNA CODA DI MESSAGGI IN CUI IL MITTENTE INSERISCE IL MESSAGGIO DA SPEDIRE E DA CUI IL RICEVENTE LO ANDRÀ A RECUPERARE.
L'ORGANIZZAZIONE DELLA CODA È NASCOSTA NEL MIDDLEWARE ED È TRASPARENTE ALL'APPLICAZIONE CHE DEVE REGISTRARSI O CONNETTERSI AL SISTEMA DI ACCODAMENTO. IN QUESTO TIPO DI COMUNICAZIONE OGNI APPLICAZIONE PUÒ SVOLGERE SIA IL RUOLO DI MITTENTE CHE DI RICEVENTE.
LA CODA DI MESSAGGI PUÒ IMPLEMENTARE DIVERSE SEMANTICHE DI CONSEGNA DEI MESSAGGI, CORRISPONDENTI A DIVERSI LIVELLI DI QUALITÀ DEL SERVIZIO.
LE SEMANTICHE PIÙ DIFFUSE SONO:

- AT LEAST ONCE DELIVERY: IL MESSAGGIO VIENE MANTENUTO FINO A QUANDO NON È POSSIBILE CONSEGNARE ALMENO UNA COPIA AL DESTINATARIO.
- AT MOST ONCE DELIVERY: LA CONSEGNA DEL MESSAGGIO NON È GARANTITA MA SICURAMENTE NON VERRANNO CONSEGNATI DUPLICATI.
- AT EXACTLY ONCE DELIVERY: IL MESSAGGIO VERRÀ MANTENUTO FINO A QUANDO NON VERRÀ CONSEGNATO ALLA DESTINAZIONE E SENZA DUPLICATI.

QUESTO MECCANISMO DI ACCODAMENTO FORNISCE UN SERVIZIO DI COMUNICAZIONE ASINCRONA ALTAMENTE AFFIDABILE MA CARENTE DAL PUNTO DI VISTA DELL'EFFICIENZA. INFATTI, IL PRELIEVO DEI MESSAGGI DALLA CODA È AFFIDATO ALL'INIZIATIVA DEL RICEVENTE CHE DEVE ESEGUIRE L'OPERAZIONE DI GET. (IN APPLICAZIONI DOVE I TEMPI DI RISPOSTA SONO UN FATTORE CRUCIALE, QUESTO MECCANISMO NON È LA SOLUZIONE ADATTA). IN GENERE STORE-AND-FORWARD VIENE UTILIZZATO DOVE DIVERSE APPLICAZIONI DEVONO INTERAGIRE CON UN'UNICA APPLICAZIONE E I TEMPI DI RISPOSTA NON SONO CRUCIALI.

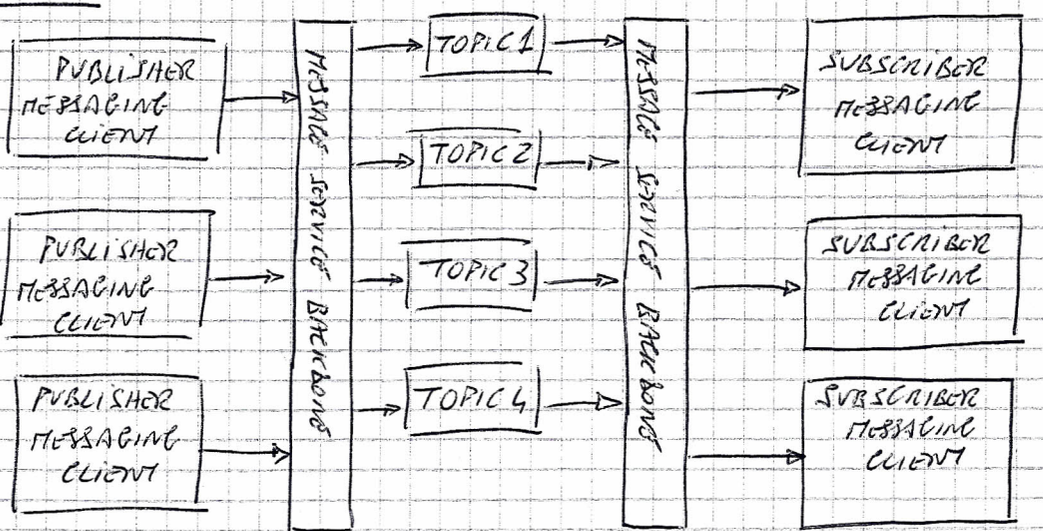


PUBLISH/SUBSCRIBE MESSAGING

PUBLISH/SUBSCRIBE MESSAGING È UN MECCANISMO DI ACCODAMENTO PIÙ SCALABILE DI STORE-AND-FORWARD È PIÙ ADATTO A COMUNICAZIONI MANY-TO-MANY. CON QUESTO TIPO DI MESSAGING ASINCRONO LA COMUNICAZIONE AVVIENE SECONDO I SEGUENTI PASSI:

- UN'APPLICAZIONE PUBBLICA UN MESSAGGIO (LO AFFIDA AL MIDDLEWARE) SPECIFICANDO L'ARGOMENTO (OVVERO IL TOPIC) MA NON L'INDIRIZZO DEL DESTINATARIO.
- LE APPLICAZIONI INTERESSATE A RICEVERE I MESSAGGI RELATIVI AD UN DATO TOPIC SI REGISTRANO CON IL MIDDLEWARE.
- OGNI VOLTA CHE IL MIDDLEWARE RICEVE UN MESSAGGIO RELATIVO AD UN DATO TOPIC LO INVIA A TUTTE LE APPLICAZIONI AL MOMENTO REGISTRATE E ATTIVE.

IN QUESTO MODO I MESSAGGI VENGONO INVIATI AI SOTTOSCRITTORI NON APPENA SI RENDONO DISPONIBILI, RIDUCENDO I TEMPI DI ATTESA RISPETTO AL MECCANISMO STORE-AND-FORWARD.



MECCANISMI EVENT-DRIVEN

I MODELLI DI MIDDLEWARE BASATI SULLA NOTIFICA DI EVENTI (EVENT DRIVEN) SI SONO RIVELATI MOLTO PIÙ ADATTI A GESTIRE LA COMUNICAZIONE ASINCRONA. TALE ASTRAZIONE È SUPPORTATA DA UNA INFRASTRUTTURA TECNICA DETTA EVENT NOTIFICATION SERVICE CHE IMPLEMENTA IL MECCANISMO DI PUBLISH/SUBSCRIBE.

L'IDEA BASE DI QUESTO APPROCCIO È CHE IL PRODUTTORE DI UN MESSAGGIO NON SPECIFICA CHI NE SONO I DESTINATARI MA AFFIDA IL MESSAGGIO AL MIDDLEWARE. SARÀ IL MIDDLEWARE, IN BASE ALLE PROPRIETÀ DEL MESSAGGIO, A DECIDERE A CHI CONSEGNARLO E COME INSTRADARLO.

IN UN EVEN NOTIFICATION SERVICE I CLIENT SI DIVIDONO IN 2 CATEGORIE:

- OBJECT OF INTERESTS: PRODUCONO NOTIFICHE DI EVENTI
- INTERESTED PARTIES: CONSUMANO LE NOTIFICHE E SI REGISTRANO CON IL SISTEMA PER RICEVERE TUTTE LE NOTIFICHE RELATIVE AD UN CERTO TOPIC.

È IMPORTANTE SOTTOLINEARE CHE QUANDO IL PRODUCER INVIA UNA NOTIFICA, NON SA CHI LA RICEVERÀ ED IL SUO MESSAGGIO NON HA UN DESTINATARIO PREFISSATO. I SERVER CHE IMPLEMENTANO IL SERVIZIO DI NOTIFICA, DEVONO INSTRADARE IL MESSAGGIO DI NOTIFICA SULLA BASE DEL SUO CONTENUTO (OVVERO DELLA SUA PROPRIETÀ) E NON DELL'INDIRIZZO DI DESTINAZIONE. QUESTA TECNICA DI INSTRADAMENTO È DETTA CONTENT-BASED ROUTING E SVOLGE UN RUOLO IMPORTANTE NELL'INTEGRAZIONE DI APPLICAZIONI CHE POSSONO COMUNICARE SENZA NECESSITÀ DI SAPERE NULLA UNA DELL'ALTRA.

POINT-TO-POINT QUEUING

IL POINT-TO-POINT QUEUING È SIMILE AL MODELLO STORE-AND-FORWARD IN QUANTO CONSENTE LO SCAMBIO DI MESSAGGI TRA UN CLIENT ED UN SERVER, TRAMITE UNA CODA MA, A DIFFERENZA DI QUELLO, QUESTI MODELLI SONO PUSH-BASED: QUANDO LA CODA RICEVE UN MESSAGGIO PROVVEDERÀ DI SUA INIZIATIVA AD INVIARLO AL DESTINATARIO SENZA ATTENDERE CHE QUESTI ESEGUA UN'OPERAZIONE DI GET.

MIDDLEWARE ORIENTATI AI MESSAGGI (MOM)

UN MIDDLEWARE ORIENTATO AI MESSAGGI (MOM) È UN'INFRASTRUTTURA CHE CONSENTE IL PASSAGGIO DI DATI TRA APPLICAZIONI UTILIZZANDO UN CANALE DI COMUNICAZIONE CHE TRASPORTA MESSAGGI AUTO-CONTENUTI. IN UN AMBIENTE DI COMUNICAZIONE BASATO SU MOM I MESSAGGI SONO SPEDITI E RICEVUTI IN MODO ASINCRONO. L'INSTRADAMENTO DEI MESSAGGI TRA LE APPLICAZIONI È AFFIDATO AL SISTEMA DI MESSAGING ED È IMPLEMENTATO DA UN MODULO DETTO MESSAGE (O INTEGRATION) BROKER.

UN MOM IMPLICA UN SERVIZIO DI MESSAGING ASINCRONO BASATO SUL MODELLO PUBLISH/SUBSCRIBE. L'INTEGRAZIONE TRA IL CLIENT ED IL SERVER AVVIENE NEL SEGUENTE MODO:

- QUANDO SI VERIFICA UN CERTO EVENTO, IL CLIENT AFFIDA AL SISTEMA DI MESSAGING UN MESSAGGIO CHE SARÀ CONSEGNATO A TUTTI I NODI CHE SI SONO REGISTRATI PER QUEL TOPIC.

LE PROPRIETÀ CHE FANNO DEI MOM UNA DELLE SOLUZIONI PIÙ INTERESSANTI PER L'INTEGRAZIONE DI SISTEMI SONO:

- BUFFERING E CONTROLLO DEL FLUSSO DEI MESSAGGI
- AFFIDABILITÀ E GARANZIA DI CONSEGNA DEI MESSAGGI
- SCALABILITÀ E USO OTTIMALE DELLE RISORSE (BILANCIAMENTO DEL CARICO)

L'INTEGRATION BROKER HA IL COMPITO DI TRASFORMARE IL FORMATO ED IL CONTENUTO DEI MESSAGGI CHE DEVE TRASPORTARE IN MODO DA RENDERLI COMPRENSIBILI AI RICEVENTI. PER SVOLGERE QUESTA FUNZIONE L'INTEGRATION BROKER MANTIENE UN REPOSITORY DI SCHEMI CHE DESCRIVONO CHE TIPO DI MESSAGGI I VARI NODI SI ASPETTANO DI RICEVERE.

JAVA MESSAGING SERVICE (JMS)

JAVA MESSAGING SERVICE È UN'API STANDARD CHE CONSENTE AD UN'APPLICAZIONE JAVA DI INTERAGIRE CON UN MOM INDIPENDENTEMENTE DALLA SUA INTERFACCIA NATIVA.
TALE API IMPLEMENTA I MODELLI POINT-TO-POINT QUEUE E PUBLISH/SUBSCRIBE.

MIDDLEWARE ORIENTATI ALLE TRANSAZIONI

I MIDDLEWARE ORIENTATI ALLE TRANSAZIONI INCLUDONO DEI MONITOR PER L'ELABORAZIONE DI TRANSAZIONI. LA TECNOLOGIA DEI MONITOR DI TRANSAZIONE AGGIUNGE AD UN AMBIENTE DISTRIBUITO CLIENT-SERVER LA CAPACITÀ DI SVILUPPARE, ESEGUIRE E GESTIRE APPLICAZIONI BASATE SU TRANSAZIONI. IL MONITOR DELLE TRANSAZIONI CONTROLLA L'ESECUZIONE DI UNA TRANSAZIONE DAL SUO INIZIO ALLA FINE ATTRAVERSO UNA SERIE DI SERVER. IN CASO DI FALLIMENTO IL MONITOR DEFINISCE A QUALE STATO IL SISTEMA DEVE ESSERE RIPORTATO. RISORSE GESTITE DAL MOT SONO: DATABASE, FILE SYSTEM ECC.

COMPUTAZIONE ORIENTATA AI SERVIZI

LA COMPUTAZIONE ORIENTATA AI SERVIZI È BASATA SULL'IDEA DI DESCRIVERE LE RISORSE DISPONIBILI, COME SERVIZI, UTILIZZABILI TRAMITE UN'INTERFACCIA STANDARD BEN DEFINITA. UN SERVIZIO È UN MODULO SOFTWARE AUTO-CONTENUTO E MESSO A DISPOSIZIONE DEGLI UTENTI SU PIATTAFORME MIDDLEWARE CHE PUÒ ESSERE DESCRITTO, PUBBLICATO, PROGRAMMATO, RICERCATO E INTEGRATO CON ALTRI SERVIZI UTILIZZANDO TECNOLOGIE XML-BASED.
PERCHÉ IL PARADIGMA DELLA COMPUTAZIONE ORIENTATA AI SERVIZI POSSA ESSERE MESSO IN PRATICA È NECESSARIO CHE I SERVIZI SIANO:

- NEUTRALI RISPETTO ALLA TECNOLOGIA
- DEBOLMENTE ACCOPPIATI
- TRASPARENTI RISPETTO ALLA LOCALIZZAZIONE

UN ASPETTO IMPORTANTE DEI SERVIZI È CHE ESSI STABILISCONO UNA RIGIDA SEPARAZIONE TRA L'INTERFACCIA DEL SERVIZIO E LA SUA IMPLEMENTAZIONE:

- L'INTERFACCIA DI UN SERVIZIO: DEFINISCE LE FUNZIONALITÀ DEL SERVIZIO VISIBILI AL MONDO ESTERNO E FORNISCE TUTTE LE INFORMAZIONI NECESSARIE A UTILIZZARE IL SERVIZIO.
- L'IMPLEMENTAZIONE DEL SERVIZIO: REALIZZA UNA DATA INTERFACCIA ED I SUOI DETTAGLI IMPLEMENTATIVI SONO COMPLETAMENTE NASCOSTI AGLI UTENTI.

NELLA COMPUTAZIONE ORIENTATA AI SERVIZI DISTINGUIAMO 3 RUOLI:

- SERVICE PROVIDERS: SONO I FORNITORI DEL SERVIZIO, CHE FORNISCONO L'IMPLEMENTAZIONE DEL SERVIZIO E CHE PUBBLICANO LE DESCRIZIONI DEI SERVIZI CONTENENTI LE INFORMAZIONI NECESSARIE AD UTILIZZARLI.
- SERVICE REQUESTORS: SONO GLI UTENTI DEL SERVIZIO, CHE UTILIZZANO IL SERVIZIO BASANDOSI SOLO SULLE INFORMAZIONI CONTENUTE NELLA SUA DESCRIZIONE.
- SERVICE INTEGRATORS: SONO GLI INTEGRATORI DI SERVIZI, CHE UTILIZZANO I SERVIZI FORNITI DA ALTRI PER INTEGRARLI ALL'INTERNO DI UN PROCESSO PIÙ COMPLESSO CHE PUÒ ESSERE REso DISPONIBILE AD ALTRI PROVIDERS.

ARCHITETTURE ORIENTATE AI SERVIZI (SOA)

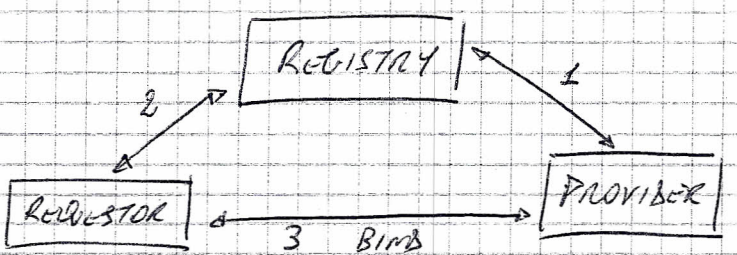
(CARATTERISTICA PRINCIPALE DELL'ARCHITETTURA SOA, È L'INTEROPERABILITÀ TRA LE APPLICAZIONI ovvero NON HANNO BISOGNO DI SAPERE NULLA L'UNA DELL'ALTRA SIA SUL PUNTO DI VISTA SOFTWARE CHE HARDWARE)

L'ARCHITETTURA ORIENTATA AI SERVIZI È UNO STILE ARCHITETTURALE FLESSIBILE E CONSENTE DI PROGETTARE APPLICAZIONI DISTRIBUITE A PARTIRE DA UN INSIEME DI UNITÀ FUNZIONALI DISPONIBILI SULLA RETE E RIUTILIZZABILI (SERVIZI) INTERAGENDO CON ESSI ATTRAVERSO INTERFACCE BEN DEFINITE.

L'OBIETTIVO PRINCIPALE DELLE ARCHITETTURE SOA È DI GARANTIRE L'INTEROPERABILITÀ TRA APPLICAZIONI DIFFERENTI IN RETE DA COSTRUIRE SISTEMI SOFTWARE BASATI SU COMPONENTI DEBOLMENTE ACCOPPIATI CHE SONO COMBINATI DINAMICAMENTE.

UN'ARCHITETTURA SOA È BASATA SU 3 ELEMENTI FONDAMENTALI:

- IL SERVICE REQUESTOR
- IL SERVICE PROVIDER
- IL REGISTRY



- IL SERVICE PROVIDER È COLUI CHE RENDE DISPONIBILE UN SERVIZIO E LO METTE A DISPOSIZIONE DI ALTRI TRAMITE UNA RETE E NE COSTITUISCE LA SUA IMPLEMENTAZIONE. IL PROVIDER È RESPONSABILE DI CREARE UNA DESCRIZIONE DEL SERVIZIO CHE HA MESSO A DISPOSIZIONE E DI PUBBLICARLA IN UNO O PIÙ REGISTRI. ESSI DEVE ANCHE RICEVERE TUTTE LE RICHIESTE DI INVOCAZIONE DEL SERVIZIO E RESTITUIRE LE RISPOSTE.
- IL SERVICE REQUESTOR È L'UTENTE DEL SERVIZIO CHE LO INVOCA PER RICHIEDERE UNA DETERMINATA FUNZIONALITÀ. IL REQUESTOR DEVE RECUPERARE LA DESCRIZIONE DEL SERVIZIO CHE VUOLE UTILIZZARE E DEVE USARE QUESTA DESCRIZIONE PER FARE IL BINDING. L'OPERAZIONE DI RICERCA NEL REGISTRY È IN GENERE BASATA SUI NOMI. INFATTI OGNI SERVIZIO È IDENTIFICATO UNIVOCAMENTE DA UN NOME CHE FUNGE DA CHIAVE DI RICERCA NEL REGISTRY. IL REQUESTOR È RESPONSABILE DELLA TRADUZIONE DELLA DESCRIZIONE DEL SERVIZIO NELLE STRUTTURE DATI NECESSARIE PER EFFETTUARE IL BINDING.
- IL SERVICE REGISTRY È UN REGISTRO CHE VIENE UTILIZZATO PER PUBBLICIZZARE LE DESCRIZIONI DEI SERVIZI PUBBLICATE DAI PROVIDER E CONSENTIRE AI REQUESTOR DI EFFETTUARE RICERCHE TRA LE DESCRIZIONI PUBBLICATE.

L'ARCHITETTURA SOA PREVEDE ANCHE 3 [redacted] OPERAZIONI FONDAMENTALI, OGNUNA SVOLTA DA UNO DEI 3 RUOLI:

1) L'OPERAZIONE DI PUBLISH È EFFETTUATA DAL PROVIDER SUL REGISTRY E CONSISTE DI 2 PASSI: IL PRIMO PASSO CONSISTE NELLA DEFINIZIONE DI UNA DESCRIZIONE DEL SERVIZIO CHE CONTENGA TUTTE LE INFORMAZIONI NECESSARIE PER POTERLO UTILIZZARE; IL SECONDO PASSO CONSISTE NELLA PUBBLICAZIONE DI QUESTA DESCRIZIONE SU UNO O PIÙ REGISTRY (SITI) PUBBLICAMENTE DISPONIBILI.
2) L'OPERAZIONE DI FIND È EFFETTUATA DAL REQUESTOR SUL REGISTRY. CON QUESTA OPERAZIONE IL REQUESTOR SPECIFICA DEI CRITERI DI RICERCA E IL REGISTRY CONTROLLA SE AL SUO INTERNO ESISTONO DELLE DESCRIZIONI CORRISPONDENTI A QUESTI CRITERI. IL RISULTATO DELL'OPERAZIONE DI FIND È UNA LISTA DI SERVIZI CHE CORRISPONDONO ALLA RICHIESTA DEL REQUESTOR.
3) L'OPERAZIONE DI BIND È ESEGUITA DAL REQUESTOR SUL PROVIDER. IL REQUESTOR UTILIZZA LE INFORMAZIONI RICEVUTE DAL REGISTRY PER COLLEGARSI AL PROVIDER E RICHIEDERE IL SERVIZIO NEL MODO CORRETTO. IL BINDING PUÒ ESSERE STATICO (INCLUSIONE DI UN FILE DI INTESTAZIONE CONTENENTE L'INTERFACCIA DELLO STUB) OPPURE DINAMICO (GENERAZIONE AL VOLO DELLO STUB SULLA BASE DELLA DESCRIZIONE DEL SERVIZIO OTTENUTA DAL REGISTRY).

CON QUESTO TIPO DI ARCHITETTURA SI GARANTISCE LA NEUTRALITÀ RISPETTO ALLA PIATTAFORMA ED AL [redacted] LINGUAGGIO DI PROGRAMMAZIONE. INOLTRE, VIENE GARANTITA LA NEUTRALITÀ ANCHE RISPETTO ALLA LOCALIZZAZIONE, INFATTI SE IL PROVIDER VIENE SPOSTATO SU UN ALTRO SERVER O CAMBIA PORTA È SUFFICIENTE AGGIORNARE LE INFORMAZIONI DEL REGISTRY.
LA CHIAVE DI TUTTA QUESTA ARCHITETTURA È LA DESCRIZIONE DEL SERVIZIO: È LA DESCRIZIONE DEL SERVIZIO CHE VIENE PUBBLICATA DAL PROVIDER, RECUPERATA DAL REQUESTOR ED UTILIZZATA PER COSTRUIRE I MESSAGGI CHE DEVONO ESSERE INVIATI E RICEVUTI E PER CAPIRE COME TALI MESSAGGI DEVONO ESSERE TRASMESSI.

IN AMBITO AZIENDALE LE ARCHITETTURE SOA POSSONO ESSERE UTILIZZATE PER LA GESTIONE DEI PROCESSI AZIENDALI. IL PROCESSO DI ELABORAZIONE DEGLI ORDINI PUÒ ESSERE MESSO A DISPOSIZIONE DEI CLIENTI DELL'AZIENDA COME SERVIZIO. QUESTO SERVIZIO PUÒ ESSERE BASATO SU ALTRI SERVIZI PIÙ SEMPLICI.

IL FATTO CHE UN SERVIZIO POSSA ESSERE ORCHESTRATO ALL'INTERNO DI UN PROCESSO PIÙ COMPLESSO PER CREARE UN SERVIZIO DI LIVELLO PIÙ ELEVATO PORTA NATURALMENTE A IMMAGINARE LE ARCHITETTURE SOA COME ORGANIZZATE A LIVELLI. AD OGNI LIVELLO SI DEFINISCONO DEGLI OGGETTI SOA CHE VENGONO RESI DISPONIBILI COME SERVIZI E UTILIZZABILI COME ELEMENTI AL LIVELLO SUPERIORE PER CREARE OGGETTI PIÙ COMPLESSI. POSSIAMO IMMAGINARE CHE QUESTA GERARCHIA SIA ORGANIZZATA IN ALMENO 3 LIVELLI:

- ORCHESTRAZIONE DI SERVIZI A LIVELLO AZIENDALE: LE SINGOLE APPLICAZIONI DISPONIBILI ALL'INTERNO DELL'AZIENDA VENGONO RESI DISPONIBILI COME SERVIZI TRAMITE LA DEFINIZIONE DI INTERFACCE E ORCHESTRATI TRA LORO PER CREARE PROCESSI AZIENDALI.
- DEPLOYMENT DI SERVIZI A LIVELLO AZIENDALE: I PROCESSI AZIENDALI OTTENUTI INTEGRANDO LE APPLICAZIONI DEI SINGOLI DIPARTIMENTI SONO RESI A DISPOSIZIONE DELL'INTERA AZIENDA COME SERVIZI.
- IMPLEMENTAZIONE DI PROCESSI TRA AZIENDE DIVERSE: PROCESSI AZIENDALI DI DIVERSE AZIENDE VENGONO INTEGRATI INSIEME PER IMPLEMENTARE PROCESSI COMPLESSI.

I WEB SERVICES

UN WEB SERVICE È UN MODULO SOFTWARE AUTO-CONTENUTO E AUTO-DESCRIVENTE CHE È INVOCABILE TRAMITE UNA RETE.

UN WEB SERVICE PUÒ IMPLEMENTARE UN SEMPLICE TASK, FORNIRE L'ACCESSO AD UNA RISORSA OPPURE IMPLEMENTARE UN INTERO PROCESSO AZIENDALE.

DEFINIZIONE RICONOSCIUTA DELLA W3C: UN WEB SERVICE È UN SISTEMA PROGETTATO PER SUPPORTARE L'INTERAZIONE MACCHINA-MACCHINA SU UNA RETE GARANTENDO L'INTEROPERABILITÀ.

UN WEB SERVICE HA UN'INTERFACCIA DESCRITTA IN UN FORMATO PROCESSABILE DA UNA MACCHINA (NEL NOSTRO CASO WSDL). GLI ALTRI SISTEMI INTERAGISCONO CON IL WEB SERVICE NEI MODI PRESCRITTI DALL'INTERFACCIA SCAMBIANDOSI MESSAGGI (NEL NOSTRO CASO SOAP), TIPICAMENTE TRASFERITI UTILIZZANDO HTTP CON UNA SERIALIZZAZIONE XML.

DAL PUNTO DI VISTA TECNICO I WEB SERVICE NON SONO ALTRO CHE UNA COLLEZIONE DI OPERAZIONI CHE SONO ACCESSIBILI TRAMITE UNA RETE E CHE SONO UTILIZZABILI TRAMITE UNA DESCRIZIONE DEL SERVIZIO. I WEB SERVICES SONO ESSENZIALMENTE MESSAGGI UNIDIREZIONALI ED ASINCRONI CHE VENGONO MAPPATI SU APPLICAZIONI SOFTWARE ESEGUIBILI. IL PROCESSORE XML CHE EFFETTUA LA CONVERSIONE DEL FORMATO DEI DATI, DEVE ESSERE SOLO ISTRUITO SU COME DEVE VALIDARE I MESSAGGI RICEVUTI E CONVERTIRLI IN DATI DELL'APPLICAZIONE.

CARATTERISTICHE DI UN WEB SERVICE

POSSIAMO DISTINGUERE DUE CATEGORIE DI WEB SERVICES:

- I WEB SERVICES SEMPLICI (INFORMATIVI): CHE IMPLEMENTANO OPERAZIONI DI TIPO RICHIESTA/RISPOSTA E CHE IN GENERE IMPLEMENTANO UNA COMUNICAZIONE DI TIPO SINCRONO. (SONO STATELESS)
- I WEB SERVICES COMPLESSI: CHE IMPLEMENTANO QUALCHE FORMA DI COORDINAMENTO TRA VARI WEB SERVICES DI TIPO SEMPLICE. (SONO STATEFUL)

ESEMPI DI UTILIZZO DEI WEB SERVICES SEMPLICI: SERVIZI CHE CONSENTONO L'ACCESSO A CONTENUTI REMOTI COME PREVISIONI METEOROLOGICHE, QUOTAZIONI DI AZIONI, INFORMAZIONI FINANZIARIE.

I WEB SERVICES COMPLESSI RICHIEDONO:

- UN MECCANISMO DI ORCHESTRAZIONE DEI WEB SERVICES (CHE VIENE FORNITO IN TERMINI DI WORKFLOW)
- LA GESTIONE DI UN CONCETTO DI STATO DEL WEB SERVICE TRA L'INVOCAZIONE DI UN SERVIZIO E QUELLA SUCCESSIVA
- LA GESTIONE DI TRANSAZIONI

PROPRIETÀ DI STATO

COME ABBIAMO VISTO UN ASPETTO IMPORTANTE RELATIVO AI WEB SERVICES È IL CONCETTO DI STATO. UN WEB SERVICE È STATELESS (SENZA STATO) O STATEFUL (CON STATO).
UN WEB SERVICE STATELESS, È UN SERVIZIO CHE NON HA MEMORIA DELLE OPERAZIONI PRECEDENTI. QUESTO SIGNIFICA CHE OGNI OPERAZIONE È INDIPENDENTE DALLE ALTRE E PRODUCE LO STESSO RISULTATO INDIPENDENTEMENTE DA COSA È SUCCESSO IN PRECEDENZA.
UN WEB SERVICE STATEFUL, MANTIENE INFORMAZIONI SUL SUO STATO TRA DIVERSE INVOCAZIONI DELLA STESSA OPERAZIONE. QUESTO VUOL DIRE CHE IL RISULTATO DI UN'INVOCAZIONE PUÒ DIPENDERE DALLA STORIA DELLE OPERAZIONI/INTERAZIONI PRECEDENTI.

CAMPI DI APPLICAZIONI

GLI AMBIENTI DI SVILUPPO ESISTENTI SONO SOPRATTUTTO PER JAVA E .NET.
I PRINCIPALI CAMPI DI APPLICAZIONE DEI WEB SERVICES SONO:
- INTEGRAZIONE DI APPLICAZIONI
- SVILUPPO DI APPLICAZIONI B2B/B2C
- SVILUPPO DI APPLICAZIONI WEB PIÙ EFFICIENTI CHE SUPERINO I LIMITI DI HTML E URL

INTEGRAZIONE DI APPLICAZIONI

INTEGRAZIONE DI APPLICAZIONI SIGNIFICA FAR LAVORARE INSIEME APPLICAZIONI SOFTWARE CHE SONO STATE PROGETTATE INDIPENDENTEMENTE. I WEB SERVICES FORNISCONO UN INSIEME DI TECNOLOGIE ATTRAVERSO LE QUALI LE APPLICAZIONI LEGACY POSSONO ESSERE INCAPSULATE IN UN WEB SERVICE E RESE INTEGRABILI CON ALTRE APPLICAZIONI CHE GIRANO SU PIATTAFORME DIFFERENTI E SONO SCRITTE IN [redacted] LINGUAGGI DI PROGRAMMAZIONE DIFFERENTI.

APPLICAZIONI B2B/B2C

LA COMPUTAZIONE B2B È L'INTEGRAZIONE TRA I SISTEMI DI DUE O PIÙ AZIENDE PER POTER SVILUPPARE PROCESSI INTER-AZIENDALI COME LA GESTIONE DELLA CATENA DELLE FORNITURE. L'IDEA DI BASE DI UN'APPLICAZIONE B2B È L'INTERAZIONE TRA DUE APPLICAZIONI SENZA L'INTERVENTO DI UN OPERATORE. L'IDEA DI BASE DI UNA APPLICAZIONE B2C PREVEDE L'INTERAZIONE TRA UN'APPLICAZIONE CHE METTE A DISPOSIZIONE DATI E/O SERVIZI ED UN OPERATORE UMANO.

STACK DI INTEROPERABILITÀ DEI WEB SERVICES

(PROPRIETÀ IMPORTANTE DELL'ARCHITETTURA SOA È L'INTEROPERABILITÀ TRA LE APPLICAZIONI)

I WEB SERVICES SONO BASATI SU UN INSIEME DI TECNOLOGIE XML. PER FORNIRE UN QUADRO DI LAVORO IN CUI INQUADRARE TUTTE QUESTE TECNOLOGIE È STATO DEFINITO UNO STACK DI INTEROPERABILITÀ, CHE RESTRINGE LE RELAZIONI TRA LE DIVERSE TECNOLOGIE. QUESTO STACK È BASATO SUI SEGUENTI 5 LIVELLI:

COMPOSITIONAL	BPEL4WS, WS-NOTIFICATION
QUALITY OF EXPERIENCE	WS-SECURITY, WS-RELIABLEMESSAGING WS-TRANSACTIONS, WS-RESOURCELIFETIME
DESCRIPTION	WSDL, UDDI, WS-POLICY, WS-RESOURCEPROPERTIES
MESSAGING	XML, SOAP, WS-ADDRESSING
TRANSPORT	HTTP, HTTPS, SMTP, FTP

IL LIVELLO TRANSPORT: AL LIVELLO DI TRASPORTO I WEB SERVICES NON SONO ALTRO CHE UN MECCANISMO DI MESSAGING E TALE LIVELLO SI PREOCCUPA DI SPECIFICARE IN CHE MODO TALI MESSAGGI POSSANO ESSERE TRASFERITI.

IL LIVELLO MESSAGING: IL LIVELLO DI MESSAGING DESCRIVE COME DEVONO ESSERE STRUTTURATI I MESSAGGI CHE SI SCAMBIANO I WEB SERVICES. LE TECNOLOGIE PRINCIPALI UTILIZZATE A QUESTO LIVELLO SONO XML, SOAP E WS-ADDRESSING.

XML È IMPORTANTE PERCHÉ IL PAYLOAD DEI MESSAGGI SCAMBIATI È SEMPRE STRUTTURATO COME UN DOCUMENTO XML.

SOAP DEFINISCE UN MECCANISMO DI INCAPSULAMENTO CHE PERMETTE DI SPECIFICARE LA STRUTTURA DEI MESSAGGI CHE VENGONO SCAMBIATI TRA DUE APPLICAZIONI.

WS-ADDRESSING È UNA SPECIFICA PER FORMALIZZARE IL CONCETTO DI PUNTATORE A UN WEB SERVICE IN MODO DA CONSENTIRE IL RIFERIMENTO AD UN WEB SERVICE DALL'INTERNO DI UN ALTRO WEB SERVICE.

IL LIVELLO DESCRIPTION: IL LIVELLO DESCRIPTION FORNISCE LA DESCRIZIONE DEL SERVIZIO CHE CONSENTE DI OTTENERE LE PROPRIETÀ DI LOOSE COUPLING E DYNAMIC BINDING IMPORTANTI PER LE ARCHITETTURE SOA.

LE PRINCIPALI TECNOLOGIE CHE VENGONO UTILIZZATE A QUESTO LIVELLO SONO: WSDL, UDDI, WS-POLICY, WS-RESOURCEPROPERTIES.

WSDL CONSENTE DI DESCRIVERE L'INTERFACCIA DEL SERVIZIO.

UDDI CONSENTE DI DEFINIRE LA STRUTTURA DEL REGISTRY, SPECIFICANDO I META-DATI CHE DEVONO ESSERE AGGIUNTI ALLA DESCRIZIONE DEL SERVIZIO PER FAVORIRE LE RICERCHE DEL REQUESTOR.

WS-POLICY È UNA SPECIFICA CHE ARRICCHISCE LA POTENZA DI WSDL, CONSENTE DI AGGIUNGERE ALLA SEMPLICE DEFINIZIONE DELL'ASPETTI FUNZIONALI DI UN SERVIZIO ANCHE INFORMAZIONI RIGUARDANTI LE SUE CARATTERISTICHE NON FUNZIONALI (ES: QUALITÀ DEL SERVIZIO, LA SICUREZZA, L'AFFIDABILITÀ).

WS-RESOURCE PROPERTIES È UNA SPECIFICA RELATIVA ALLA GESTIONE DI INFORMAZIONI DI STATO PER IMPLEMENTARE WEB SERVICES STATEFUL.

IL LIVELLO QUALITY OF EXPERIENCE: QUESTO LIVELLO CONTIENE SPECIFICHE RELATIVE AD ASPETTI PARTICOLARI DELLA COMPUTAZIONE BASATA SU WEB SERVICES (ES: LA SICUREZZA, L'AFFIDABILITÀ, LA GESTIONE DELLE TRANSAZIONI).

WS-SECURITY È UNA FAMIGLIA DI SPECIFICHE CHE DEFINISCONO COME RENDERE SICURA L'INTERAZIONE TRA WEB-SERVICES.

WS-RELIABLEMESSAGING È UNA SPECIFICA CHE CONSENTE DI GARANTIRE L'AUTENTICITÀ E LA NON RIPUDIABILITÀ DEI MESSAGGI SCAMBIATI TRA WEB SERVICES.

WS-TRANSACTIONS È UNA FAMIGLIA DI SPECIFICHE CHE DEFINISCONO COME I WEB-SERVICES SI COLLEGANO ALLA GESTIONE DELLE TRANSAZIONI.

IL LIVELLO COMPOSITIONAL: QUESTO LIVELLO CONTIENE SPECIFICHE CHE DESCRIVONO COME I WEB SERVICES POSSONO ESSERE COMBINATI INSIEME PER CREARE NUOVI SERVIZI.

LA SPECIFICA PRINCIPALE È BPEL4WS CHE È UNO STANDARD PER DEFINIRE COME I WEB-SERVICES POSSONO ESSERE COMBINATI IN UN PROCESSO AD LIVELLO PIÙ ALTO, SPECIFICANDO IL WORKFLOW E L'ORCHESTRAZIONE DELLO SCAMBIO DEI VARI SERVIZI.

WS-NOTIFICATION È UNA SPECIFICA CHE DESCRIVE COME SI POSSA GESTIRE LA PUBBLICAZIONE DI MESSAGGI DI NOTIFICA E LA SOTTOSCRIZIONE A TALI SERVIZI IN UN AMBIENTE BASATO SU WEB-SERVICES.

XML

XML È STATO CREATO PER CONSENTIRE LO SCAMBIO E L'ELABORAZIONE DI GENERICI DOCUMENTI TRAMITE IL WEB GARANTENDO ALLO STESSO TEMPO LA FACILITÀ DI IMPLEMENTAZIONE E L'INTEROPERABILITÀ DELLE APPLICAZIONI CHE DEVONO UTILIZZARE TALI DOCUMENTI.
LE TECNOLOGIE XML HANNO TROVATO APPLICAZIONE NEI CAMPI PIÙ DISPARATI. POSSIAMO DIVIDERE QUESTE APPLICAZIONI IN 2 GRANDI FAMIGLIE:

- LE APPLICAZIONI DOCUMENT-CENTRIC: XML HA TROVATO GRANDE DIFFUSIONE NELL'AMBITO DEI SISTEMI DI PUBBLICAZIONE PER LA SUA CAPACITÀ DI RAPPRESENTARE DOCUMENTI SEMISTRUTTURATI QUALI MANUALI TECNICI, CATALOGHI.
- LE APPLICAZIONI DATA-CENTRIC: XML VIENE UTILIZZATO PER RAPPRESENTARE DATI ALTAMENTE STRUTTURATI, COME RECORD DI UN DATABASE, TRANSAZIONI FINANZIARIE.

STRUTTURA DI UN DOCUMENTO XML

OGNI DOCUMENTO XML HA SIA UNA STRUTTURA FISICA CHE UNA STRUTTURA LOGICA.
DAL PUNTO DI VISTA FISICO: I DOCUMENTI XML SONO COSTITUITI DA ENTITÀ, LE ENTITÀ POSSONO ESSERE PARSED O UNPARSED.

- UNPARSED: SONO RISORSE IL CUI CONTENUTO PUÒ ANCHE NON ESSERE TESTO.
- PARSED: SONO COSTITUITE DA CARATTERI UNICODE CHE FORMANO SIA DATI CHE MARCATORI E SONO PARTE INTEGRANTE DEL DOCUMENTO.

XML FORNISCE UN MECCANISMO PER IMPORRE VINCOLI SULLA STRUTTURA FISICA E LOGICA DEL DOCUMENTO. I MARKUP SONO IDENTIFICATI PERCHÉ RACCHIUSI TRA CARATTERI DELIMITATORI (< E > OPPURE & E ;).
I MARKUP POSSONO ASSUMERE LE SEGUENTI FORME:

- START-TAG E END-TAG: IDENTIFICANO L'INIZIO E LA FINE DI UN ELEMENTO XML.
- ENTITÀ REFERENCES E CHARACTER REFERENCES: SONO RACCHIUSI TRA I CARATTERI & E ; E SONO RIFERIMENTI AD ALTRI ELEMENTI
- COMMENTI: SONO RACCHIUSI TRA I TAG <!-- > E <-- >.
- ISTRUZIONI DI ELABORAZIONE (PROCESSING INSTRUCTIONS): SONO RACCHIUSE TRA I TAG <? > E <? >. CONTENGONO ISTRUZIONI PER L'APPLICAZIONE CHE DEVE PROCESSARE IL DOCUMENTO.

- SEZIONI CDATA: RACCHIUSE ALL'INTERNO DEL TAG <![CDATA[ ]]>
- DOCUMENT TYPE DECLARATION (DTD): RACCHIUSA NEL TAG <!DOCTYPE> CONTIENE O FA RIFERIMENTO AD UNA O PIÙ DICHIARAZIONI DI MARKUP. VIENE UTILIZZATA PER VINCOLARE LA STRUTTURA LOGICA E FISICA DI UN DOCUMENTO.

DAL PUNTO DI VISTA LOGICO, OGNI DOCUMENTO XML È COMPOSTO DI ELEMENTI I CUI CONFINI SONO IDENTIFICATI DA MARCATORI START-TAG (<NOME_ELEMENTO) ED END-TAG (</NOME_ELEMENTO>). OGNI ELEMENTO HA UN TIPO, IDENTIFICATO DAL NOME DELL'ELEMENTO, E PUÒ AVERE UNO O PIÙ ATTRIBUTI CHE DEFINISCONO ULTERIORI CARATTERISTICHE DELL'ELEMENTO. L'XML DISTINGUE TRA CARATTERI MAIUSCOLI E MINUSCOLI (CASE-SENSITIVE). GLI ATTRIBUTI COMPAIONO ALL'INTERNO DELLO START-TAG E SONO COPPIA NOME-VALORE. IN BASE AL TIPO DI CONTENUTO GLI ELEMENTI SI DIVIDONO IN 3 CATEGORIE:

- SOLO ELEMENTI: IL CONTENUTO DELL'ELEMENTO È FORMATO SOLO DA ELEMENTI ANNIDATI E NON C'È TESTO LIBERO.
- CONTENUTO MISTO: IL CONTENUTO DELL'ELEMENTO È FORMATO SIA DA TESTO CHE DA ALTRI ELEMENTI ANNIDATI.
- CONTENUTO VUOTO: L'ELEMENTO NON HA CONTENUTO.

I DOCUMENTI XML HANNO UNA STRUTTURA GERARCHICA E PUÒ ESSERE VISTO COME UN ALBERO. GLI ELEMENTI I CUI START-TAG ED END-TAG COMPAIONO ALL'INTERNO DEL CONTENUTO DELL'ELEMENTO X SONO DETTI FIGLI DI X ED X È DETTO LORO PADRE. ALL'INTERNO DI UN DOCUMENTO XML C'È UN ELEMENTO CHE NON È FIGLIO DI NESSUNO ED È DEFINITO ELEMENTO RADICE (ROOT)

PROCESSORE XML

PER LEGGERE I DOCUMENTI XML E FORNIRE ACCESSO AL SUO CONTENUTO E ALLA SUA STRUTTURA VIENE UTILIZZATO UN MODULO SOFTWARE CHIAMATO PROCESSORE XML.

IL PROCESSORE XML PUÒ RESTITUIRE 2 TIPI DI ERRORI:

- ERROR: È UNA VIOLAZIONE ALLE REGOLE DELLA SPECIFICA XML. IL PROCESSORE XML PUÒ INDIVIDUARE E RIPORTARE QUESTI ERRORI E CERCARE DI CORREGGERLI.
- FATAL ERROR: È UN ERRORE CHE OGNI PROCESSORE XML DEVE OBBLIGATORIAMENTE INDIVIDUARE E RIPORTARE ALL'APPLICAZIONE. DOPO UN FATAL ERROR IL PROCESSORE PUÒ CONTINUARE A LEGGERE ED ELABORARE IL DOCUMENTO MA NON PUÒ PROSEGUIRE NELLA NORMALE ELABORAZIONE.

NELLA SPECIFICA DI XML ABBIAMO 2 CATEGORIE DI REGOLE CHE UN DOCUMENTO XML DEVE SODDISFARE:

- REGOLE DI WELL-FORMED: UN DATA OBJECT È UN DOCUMENTO XML SE È WELL-FORMED. I DOCUMENTI WELL-FORMED DEVONO SODDISFARE LE REGOLE DI WELL-FORMEDNESS. LE VIOLAZIONI DI WELL-FORMED SONO FATAL ERROR.
- REGOLE DI VALIDITÀ: UN DOCUMENTO XML WELL-FORMED È VALIDO SE SODDISFA LE REGOLE DI VALIDITÀ. LE VIOLAZIONI DI VALIDITÀ SONO ERRORI.

REGOLE DI WELL-FORMED

UN DATA OBJECT È UN DOCUMENTO XML WELL-FORMED SE RISPETTA LE SEGUENTI REGOLE:

- IL DOCUMENTO È COSTITUITO DA UNO O PIÙ ELEMENTI: CI DEVE ESSERE ESATTAMENTE UN SOLO ELEMENTO, DETTO ROOT, CHE NON COMPARE NEL CONTENUTO DI NESSUN ALTRO ELEMENTO.
- OGNI ELEMENTO C DEVE AVERE START-TAG ED END-TAG CONTENUTI NELLO STESSO ELEMENTO P.
- IL NOME INDICATO NELL'START-TAG ED END-TAG DI UN ELEMENTO DEVONO ESSERE UGUALI.
- UN ELEMENTO NON PUÒ AVERE DUE ATTRIBUTI CON LO STESSO NOME.
- IL VALORE DI UN ATTRIBUTO DEVE ESSERE TESTO (TRA APICI O VIRGOLETTE).

REGOLE DI VALIDITÀ

UN DOCUMENTO XML WELL-FORMED È COMPOSTO DA:

- PROLOGO (OPZIONALE): CONTIENE LA DICHIARAZIONE XML E UNA DOCUMENT TYPE DECLARATION (DTD)
- DOCUMENTO

XML NAMESPACE

UNO DEI PUNTI DI FORZA DI XML È LA FACILITÀ CON CUI DOCUMENTI XML DIVERSI POSSONO ESSERE COMBINATI PER COSTRUIRE DOCUMENTI XML PIÙ COMPLESSI. PURTROPPO PERÒ, ANCHE SEMPLICI COMPOSIZIONI POSSONO PROVOCARE PROBLEMI DI CONFLITTI DI NOMI E CONSEGUENTE DELLA AMBIGUITÀ, E QUESTO PUÒ PROVOCARE DUE TIPI DI PROBLEMI:

- RICONOSCIMENTO: UN'APPLICAZIONE CHE PROCESSA IL DOCUMENTO XML COME FA A DISTINGUERE TRA I DUE DIVERSI ELEMENTI?
- COLLISIONE: UN ELEMENTO "ITEM" SI RIFERISCE AD UN ELEMENTO IN "ATTACHMENT" AL MESSAGGIO OPPURE AD UNO DEGLI ELEMENTI DELL'ORDINE?

IL PROBLEMA DELLE COLLISIONI PUÒ ESSERE RISOLTO INTRODUCENDO IL CONCETTO DI NAMESPACE. L'IDEA È DI QUALIFICARE OGNI NOME DI ELEMENTO XML CON UN ULTERIORE IDENTIFICATORE CHE LO RENDA UNIVOCO ALL'INTERNO DEL DOCUMENTO. IL NOME DI UN ELEMENTO, QUINDI, SARÀ FORMATO DA DUE PARTI:

- UN IDENTIFICATORE DI NAMESPACE
- UN NOME LOCALE

L'UNIONE DI QUESTE DUE PARTI VIENE IN GENERE DEFINITO COME NOME QUALIFICATO DELL'ELEMENTO (QNAME). UN XML NAMESPACE VIENE IDENTIFICATO TRAMITE UN UNIFORM RESOURCE IDENTIFIER (URI). IN GENERALE UNA URI NON FA RIFERIMENTO A NESSUNA RISORSA EFFETTIVAMENTE DISPONIBILE. LO SCOPO DI UNA URI È QUELLO DI AVERE UNA STRINGA CHE IDENTIFICA UNIVOCAMENTE UNA RISORSA

L'UTILIZZO DEI NAMESPACE XML AVVIENE IN DUE PASSI:

- NELLO START-TAG DI UN ELEMENTO VIENE DEFINITO UN NAMESPACE E GLI VIENE ASSOCIATO UN PREFISSO.
- NOMI QUALIFICATI DEGLI ELEMENTI SONO FORMATI DAL PREFISSO ASSOCIATO AL NAMESPACE E DAL NOME LOCALE SEPARATI DAL CARATTERE ":"

I NAME-SPACE VENGONO DICHIARATI UTILIZZANDO L'ATTRIBUTO "XMLNS" A CUI GLI VIENE ASSEGNATO UN PREFISSO. NEL DOCUMENTO OGNI ELEMENTO VIENE QUALIFICATO CON IL NOME DEL NAMESPACE A CUI APPARTIENE. A QUESTO PUNTO NON C'È PIÙ AMBIGUITÀ.

QUANDO NEL DOCUMENTO COMPARE UN NOME NON QUALIFICATO SI ASSUME CHE IL NOME SIA QUALIFICATO CON IL NAMESPACE DI DEFAULT.

XML SCHEMA

DTD E VALIDAZIONE DI DOCUMENTI XML

XML FORNISCE UN MECCANISMO PER DEFINIRE LA STRUTTURA DI UN DOCUMENTO, ESSO È COSTITUITO DALLE DOCUMENT-TYPE-DEFINITIONS (DTDS), CHE SONO ELEMENTI OPZIONALI CHE POSSONO ESSERE PRESENTI O MENO ALL'INTERNO DI UN DOCUMENTO XML WELL-FORMED. SE LA DTD È PRESENTE, FORNISCE UN INSIEME DI REGOLE RIGUARDO GLI ELEMENTI E GLI ATTRIBUTI CHE POSSONO ESSERE PRESENTI NEL DOCUMENTO E LE LORO RELAZIONI (AD ESEMPIO: UNA DTD PUÒ SPECIFICARE CHE L'ELEMENTO RADICE DEL DOCUMENTO DEBBA AVERE 3 FIGLI TUTTI CON LO STESSO NOME E CIASCUNO DI LORO DEVE AVERE UN ATTRIBUTO DI NOME VAL).
PER DEFINIRE UNA DTD SI UTILIZZA UNA SINTASSI AD-HOC CHE NON È MOLTO INTUITIVO. LE DTD NON SUPPORTANO I NAMESPACE.
IN PRATICA UNA DTD CONSENTE DI STABILIRE LA STRUTTURA DEL DOCUMENTO, FISSANDO IL NUMERO DI FIGLI DI OGNI NODO E I LORO NOMI, MA NON CONSENTE DI FISSARE IL TIPO DI CONTENUTI (QUESTA LIMITAZIONE LE RENDONO NON ADATTE ALLE ESIGENZE DI APPLICAZIONI DATA-ORIENTED).
PER QUESTO MOTIVO SI È VOLUTO CREARE UN META-LINGUAGGIO CHE CONSENTA DI DESCRIVERE LA STRUTTURA DI UN DOCUMENTO XML E DI ASSOCIARE AGLI ELEMENTI XML IL CONCETTO DI TIPO DI DATI.
IL RISULTATO È STATA LA SPECIFICA XML SCHEMA.

XML SCHEMA

PER VERIFICARE LA VALIDITÀ DI UN DOCUMENTO XML GLI SCHEMI CONSENTONO DI:
- IDENTIFICARE GLI ELEMENTI CHE POSSONO ESSERE PRESENTI NEL DOCUMENTO E SPECIFICARE IL LORO TIPO
- IDENTIFICARE L'ORDINE E LE RELAZIONI TRA GLI ELEMENTI
- IDENTIFICARE GLI ATTRIBUTI DI OGNI ELEMENTO, SPECIFICANDO SE SONO OPZIONALI, OBBLIGATORI O SE HANNO ALTRE PROPRIETÀ
- IDENTIFICARE IL TIPO DI DATI DEI VALORI DI OGNI ATTRIBUTO

LA DEFINIZIONE DEGLI SCHEMI, FACILITA LA COMPOSIZIONE DI SCHEMI, PERMETTENDO DI RIUTILIZZARE SCHEMI XML PER CREARE ALTRI SCHEMI PIÙ COMPLESSI.
XML SCHEMA È MOLTO POTENTE PERCHÉ CONSENTE DI DEFINIRE IL CONTENUTO DI DOCUMENTI XML IN MANIERA PRECISA ED ESPRESSIVA, SUPERANDO I LIMITI DELLE DTD.
QUESTA SUA POTENZA ESPRESSIVA, PERÒ, IMPLICA ANCHE UNA NOTEVOLE COMPLESSITÀ.

LA SPECIFICA DI XML È DIVISA IN 3 PARTI:

1) XML SCHEMA PART 0: PRIMER
È UN DOCUMENTO NON NORMATIVO CHE CERCA DI SPIEGARE XML SCHEMA IN MANIERA INFORMALE E UTILIZZANDO MOLTI ESEMPI.

2) XML SCHEMA PART 1: STRUCTURES
FORNISCE UN MODELLO PER DEFINIRE LA STRUTTURA DI UN DOCUMENTO XML, ORIENTANDOSI SULLE NECESSITÀ DI APPLICAZIONI DOCUMENT-ORIENTED

3) XML SCHEMA PART 2: DATATYPES
AGGIUNGE ALLE REGOLE DELLA PART 1, DELLE CAPACITÀ AGGIUNTIVE PER SODDISFARE LE ESIGENZE DI APPLICAZIONI DATA-ORIENTED (AD ESEMPIO: LA DEFINIZIONE DI TIPI DI DATI).

STRUTTURA DI UNO SCHEMA

L'ELEMENTO RADICE DI UNO SCHEMA È L'ELEMENTO XSD:SCHEMA. QUESTO ELEMENTO PUÒ CONTENERE DICHIARAZIONI DI ELEMENTI E ATTRIBUTI, DEFINIZIONI DI TIPI DI DATI. ALL'INTERNO DELLO START-TAG DELL'ELEMENTO SCHEMA VIENE DEFINITO L'ATTRIBUTO TARGETNAMESPACE IL CUI VALORE È IL NOME DEL NAMESPACE A CUI APPARTENGONO TUTTI GLI ELEMENTI ED I TIPI DI DATO DEFINITI DALLO SCHEMA. A QUESTO SI AFFIANCANO LE DEFINIZIONI DEI NAMESPACE UTILIZZATI ALL'INTERNO DELLO SCHEMA.
PER DICHIARARE ELEMENTI ED ATTRIBUTI SI UTILIZZANO RISPETTIVAMENTE GLI ELEMENTI XSD:ELEMENT E XSD:ATTRIBUTE. ENTRAMBI HANNO DUE ATTRIBUTI:
- NAME: IL CUI VALORE RAPPRESENTA IL NOME CON CUI SI DEVE FAR RIFERIMENTO A QUESTO ELEMENTO.
- TYPE: IL CUI VALORE SPECIFICA IL TIPO DI DATO DELL'ELEMENTO

A PARTIRE DAI TIPI DI DATI PREDEFINITI È POSSIBILE DEFINIRE NUOVI TIPI DI DATI. XML SCHEMA DISTINGUE 3 CLASSI DI TIPI DI DATI:
- SIMPLE TYPE: SONO ELEMENTI CHE NON HANNO FIGLI NÉ ATTRIBUTI
- COMPLEXTYPE CON SIMPLECONTENT: SONO ELEMENTI CHE NON HANNO FIGLI MA POSSONO AVERE ATTRIBUTI
- COMPLEXTYPE CON COMPLEXCONTENT: SONO ELEMENTI CHE POSSONO AVERE SIA FIGLI CHE ATTRIBUTI

DICHIARAZIONI DI TIPI SEMPLICI

UN TIPO SEMPLICE VIENE DEFINITO ALL'INTERNO DELL'ELEMENTO XSD:SIMPLETYPE A PARTIRE DA UN ALTRO TIPO SEMPLICE NOTO, USANDO L'ELEMENTO XSD:RESTRICTION PER RESTRINGERE L'INSIEME DEI VALORI CHE PUÒ ASSUMERE L'ELEMENTO (ESEMPIO: LUNGHEZZA DELLE STRINGHE, INTERVALLI DI VALORI AMMISSIBILI).

DICHIARAZIONI DI TIPI COMPLESSI

IN XML SCHEMA, I TIPI COMPLESSI DEFINISCONO TIPI DI DATI CON MODELLO DI CONTENUTO COMPLESSO, CHE POSSONO CONTENERE ELEMENTI ANNIDATI O ATTRIBUTI. LA DEFINIZIONE DI UN TIPO COMPLESSO VIENE FATTA ALL'INTERNO DELL'ELEMENTO XSD:COMPLEXTYPE. LA DISTINZIONE TRA CONTENUTO SEMPLICE E COMPLESSO VIENE FATTO INSERENDO LA DEFINIZIONE DEL CONTENUTO ALL'INTERNO O DEGLI ELEMENTI XSD:SIMPLECONTENT O XSD:COMPLEXCONTENT. LA DEFINIZIONE DEGLI ATTRIBUTI VIENE FATTA USANDO L'ELEMENTO XSD:ATTRIBUTE E PER OGNI ATTRIBUTO È NECESSARIO SPECIFICARE SE È OBBLIGATORIO O OPZIONALE.
NELLA DEFINIZIONE DI UN CONTENUTO COMPLESSO È NECESSARIO SPECIFICARE L'ORDINE IN CUI DEVONO COMPARIRE GLI ELEMENTI ANNIDATI.
XML SCHEMA PROVVEDE 6 MODELLI DI GRUPPO PER DESCRIVERE LA STRUTTURA DEL CONTENUTO DI UN ELEMENTO:

- XSD:SEQUENCE: È UNA SEQUENZA DI ELEMENTI CHE DEVONO PRESENTARSI NELL'ORDINE E CON LA MOLTEPLICITÀ SPECIFICATE.
- XSD:CHOICE: NEL CONTENUTO DELL'ELEMENTO PUÒ COMPARIRE UNO SOLO DEGLI ELEMENTI SPECIFICATI NEL MODELLO.
- XSD:ALL: È UN INSIEME DI ELEMENTI CHE POSSONO COMPARIRE IN UN QUALSIASI ORDINE, MA OGNI ELEMENTO DEVE COMPARIRE UNA SOLA VOLTA.
- XSD:GROUP: REFERENZIA UN MODELLO DI GRUPPO CHE È DEFINITO IN UN ALTRO PUNTO DELLO SCHEMA.

QUESTI MODELLI POSSONO ESSERE A LORO VOLTA ANNIDATI UNO NELL'ALTRO PER CREARE MODELLI PIÙ COMPLESSI.
DOPO LA DEFINIZIONE DEL MODELLO DI GRUPPO SI DEVONO DEFINIRE GLI ATTRIBUTI DEL TIPO. CON L'ATTRIBUTO "USE" DELL'ELEMENTO XSD:ATTRIBUTE È POSSIBILE SPECIFICARE SE L'ATTRIBUTO CHE SI STA DEFINENDO È REQUIRED (OBBLIGATORIO), OPTIONAL (FACOLTATIVO) O PROHIBITED (VIETATO).
GLI ELEMENTI XSD:ELEMENT E XSD:ATTRIBUTE CONSENTONO ANCHE DI FISSARE VALORI DI DEFAULT O VALORI FISSI USANDO GLI ATTRIBUTI DEFAULT E FIXED.

COLLEGAMENTO DI UN'ISTANZA XML CON LO SCHEMA CORRISPONDENTE

UN'ISTANZA DI UN DOCUMENTO XML NON DEVE NECESSARIAMENTE ESSERE COLLEGATA CON UNO SCHEMA XML PERCHÉ LE APPLICAZIONI CHE DEVONO PROCESSARE IL DOCUMENTO POSSONO SCEGLIERE AUTONOMAMENTE QUALE SCHEMA APPLICARE PER VALIDARE IL DOCUMENTO E INTERPRETARNE IL CONTENUTO. NELLA MAGGIOR PARTE DEI CASI, PERÒ, SI PREFERISCE INDICARE ALL'INTERNO DELL'ISTANZA XML LO SCHEMA RISPETTO AL QUALE QUEST'ISTANZA DOVREBBE ESSERE VALUTATA.
PER ASSOCIARE UNO SCHEMA AD UN'ISTANZA XML SI UTILIZZA L'ATTRIBUTO SCHEMALOCATION ALL'INTERNO DELLO START-TAG DELL'ELEMENTO RADICE DEL DOCUMENTO. L'ATTRIBUTO XSI:SCHEMALOCATION CONTIENE UNA COPPIA DI VALORI: IL PRIMO VALORE È L'IDENTIFICATORE DI UN NAMESPACE, IL SECONDO VALORE IDENTIFICA LA POSIZIONE DELLO SCHEMA ASSOCIATO A QUESTO NAMESPACE.

ELEMENTI ED ATTRIBUTI GLOBALI E LOCALI

ALL'INTERNO DI UNO SCHEMA SI DISTINGUE TRA ELEMENTI E ATTRIBUTI GLOBALI, CHE SONO QUELLI DEFINITI COME FIGLI DELL'ELEMENTO XSD:SCHEMA, E TUTTI GLI ALTRI CHE SONO DETTI LOCALI. LA DISTINZIONE È IMPORTANTE PERCHÉ SOLO GLI ELEMENTI GLOBALI POSSONO ESSERE RADICE DI UN DOCUMENTO.

RIUTILIZZABILITÀ DEGLI SCHEMI

IL CONCETTO DI RIUTILIZZABILITÀ È UN ELEMENTO FONDAMENTALE DI XML SCHEMA ED UNO DEI SUOI PUNTI DI FORZA. I MECCANISMI DI RIUTILIZZABILITÀ CONSENTONO DI SFRUTTARE TUTTI GLI SCHEMI GIÀ ESISTENTI PER COSTRUIRE NUOVI MECCANISMI PIÙ COMPLESSI.
L'UTILIZZO DI RIFERIMENTI A ELEMENTI/ATTRIBUTI PREESISTENTI VIENE IMPLEMENTATO ATTRAVERSO L'USO ALL'INTERNO DELL'ELEMENTO XSD: ELEMENT/XSD: ATTRIBUTE DELL'ATTRIBUTO "REF", IL CUI VALORE È IL NOME DI UN ALTRO ELEMENTO/ATTRIBUTO. IL RIFERIMENTO PUÒ ESSERE FATTO ANCHE A GRUPPI DI ELEMENTI O DI ATTRIBUTI, IDENTIFICATI TRAMITE L'ELEMENTO XSD: GROUP.
TUTTAVIA, XML SCHEMA FORNISCE ANCHE DUE MECCANISMI PER RIUTILIZZARE ELEMENTI DEFINITI IN ALTRI DOCUMENTI E COSTRUIRE SCHEMI ESTREMAMENTE COMPLESSI, I DUE MECCANISMI SONO: IMPORT E INCLUDE.

- INCLUDE: È POSSIBILE RIFERIRSI AD UN'ALTRO SCHEMA MENTRE SI LAVORA CON UN SECONDO SCHEMA, SPECIFICANDO IL NOME DELLO SCHEMA A CUI FARE RIFERIMENTO TRAMITE L'ATTRIBUTO "SCHEMALOCATION". DURANTE L'ELABORAZIONE

DEL 2° SCHEMA, IL CONTENUTO DEL FILE "INCLUSO" VIENE INCORPORATO ALL'INTERNO DEL 2° SCHEMA. RICORDIAMO CHE L'ELEMENTO INCLUSO MANTIENE IL SUO NAMESPACE DI ORIGINE.
QUINDI IL MECCANISMO XSD:INCLUDE FUNZIONA SOLO SE LO SCHEMA DA INCLUDERE SI TROVA NELLO STESSO NAMESPACE DELLO SCHEMA IN CUI SI STA INCLUDENDO.

- IMPORT: È POSSIBILE INCLUDERE FRAMMENTI DI SCHEMI DA NAMESPACE DIFFERENTI, IMPORTANDOLI NEL NAMESPACE DEL NUOVO SCHEMA. L'ELEMENTO XSD:IMPORT HA 2 ATTRIBUTI: NAMESPACE (CHE INDICA IL NOME DEL NAMESPACE IN CUI VERRANNO INSERITI GLI ELEMENTI IMPORTATI), E SCHEMALOCATION (CHE INDICA IL NOME DELLO SCHEMA DA IMPORTARE).

ESISTE UNA MANIERA PIÙ SOFISTICATA DI RIUTILIZZARE QUESTI ELEMENTI CHE È QUELLA DI USARLI COME BASE PER DERIVARE NUOVI TIPI DI DATI ATTRAVERSO I MECCANISMI DI ESTENSIONE E RESTRIZIONE. CON IL MECCANISMO DI ESTENSIONE, IL NUOVO TIPO CONTIENE TUTTI I FIGLI E GLI ATTRIBUTI DEL TIPO BASE PIÙ QUELLI AGGIUNTI. UN MODO ALTERNATIVO DI DEFINIRE NUOVI TIPI A PARTIRE DA TIPI INCLUSI DA ALTRI SCHEMI È DI UTILIZZARE IL MECCANISMO DI RIDEFINIZIONE. L'ELEMENTO <XSD:REDEFINE> FUNZIONA IN MANIERA ANALOGA A <XSD:EXTENSION> CON LA DIFFERENZA CHE NON CREA UN NUOVO TIPO DI DATI MA SOSTITUISCE LA DEFINIZIONE DEL TIPO PRECEDENTE.
IL MECCANISMO DI RESTRIZIONE SI RIFERISCE AI SEGUENTI ASPETTI:

- DEFINIZIONE DI VALORI DI DEFAULT
- CANCELLAZIONE DI ELEMENTI OPZIONALI
- DEFINIZIONE DI TIPI PIÙ RISTRETTI PER QUALCHE ELEMENTO.

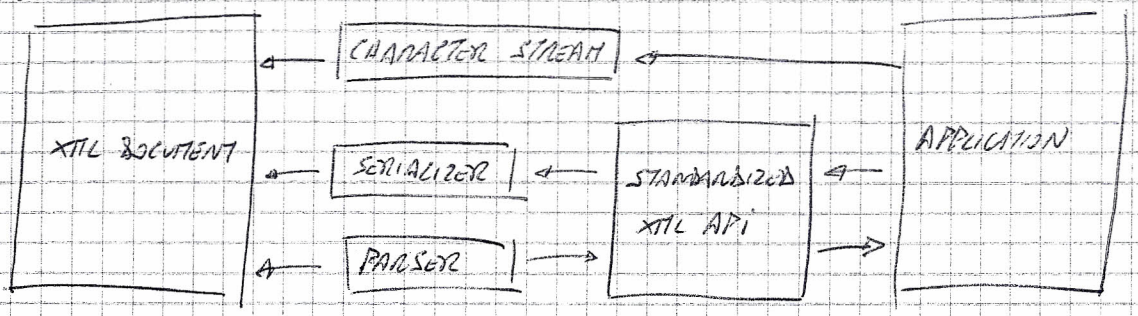
L'ATTRIBUTO XSI:TYPE

CON LE DERIVAZIONI PER ESTENSIONE, IL PROCESSORE DEVE CONOSCERE IL TIPO DI DATI EFFETTIVAMENTE UTILIZZATO PER POTERNE CONVALIDARE LO SCHEMA. È POSSIBILE FORNIRE QUESTA INFORMAZIONE AL PROCESSORE XML UTILIZZANDO L'ATTRIBUTO GLOBALE XSI:TYPE, CHE PUÒ ESSERE APPLICATO A QUALSIASI ELEMENTO DELLO SCHEMA.

MODELLI DI PARSING

XML PROCESSING ORIENTATO ALLA SINTASSI

L'ARCHITETTURA BASE DELL'ELABORAZIONE DI UN DOCUMENTO XML È ARTICOLATA NEL SEGUENTE SCHEMA:



PARSING

CONSIDERIAMO PRIMA IL CASO IN CUI L'APPLICAZIONE DEVE ESTRARRE DELLE INFORMAZIONI DA UN DOCUMENTO XML.

IL PRIMO PASSO PER PROCESSARE UN DOCUMENTO XML CONSISTE NEL FARNE IL PARSING.

FARE IL PARSING DI UN DOCUMENTO XML SIGNIFICA SCOMPORRE IL TESTO DEL DOCUMENTO IN PICCOLE UNITÀ IDENTIFICABILI, DENOMINATE NODI.

QUESTA OPERAZIONE VIENE ESEGUITA DA UN MODULO SOFTWARE DETTO PARSER, CHE SCOMPONE IL DOCUMENTO XML IN UNITÀ START-TAG, END TAG, COPPIE ATTRIBUTO-VALORE, COMMENTI, PROCESSING INSTRUCTIONS... QUESTI NODI VENGONO FORNITI ALL'APPLICAZIONE ATTRAVERSO UNA API-STANDARD CHE IMPLEMENTA UNO SPECIFICO MODELLO DI PARSING.

I MODELLI DI PARSING COMUNEMENTE UTILIZZATI SONO:

- PULL PARSING: IN QUESTO MODELLO, IL PARSING È CONTROLLATO DALL'APPLICAZIONE CHE, QUANDO LO RITIENE OPPORTUNO, CHIEDE AL PARSER DI FORNIRLE LA SUCCESSIVA UNITÀ DI INFORMAZIONE. L'API CHE IMPLEMENTA TALE PARSING, SI CHIAMA STAX (STREAMING API FOR XML).

- PUSH PARSING: IN QUESTO MODELLO, IL PARSING È CONTROLLATO DAL PARSER CHE LEGGE IL DOCUMENTO E, OGNI VOLTA CHE INDIVIDUA UN'UNITÀ DI INFORMAZIONE, INVIA UNA NOTIFICA ALL'APPLICAZIONE. TALE MODELLO È ANCHE DEFINITO COME PARSING EVENT-BASED. L'API CHE IMPLEMENTA TALE PARSING, SI CHIAMA SAX (SIMPLE API FOR XML).

- ONE-STEP PARSING: IN QUESTO MODELLO, IL PARSER LEGGE IN UNA VOLTA L'INTERO DOCUMENTO XML E GENERA UNA STRUTTURA DATI, DETTA PARSE TREE, CHE NE DESCRIVE IL CONTENUTO (ELEMENTI, ATTRIBUTI...)
L'API CHE IMPLEMENTA TALE MODELLO SI CHIAMA DOM (DOCUMENT OBJECT MODEL), CHE SPECIFICA IL TIPO DEGLI OGGETTI CONTENUTI NEL PARSE TREE E LE OPERAZIONI CHE È POSSIBILE ESEGUIRE SU DI ESSI.
LA COMUNITÀ JAVA HA SVILUPPATO UNA SPECIFICA API, CHIAMATA JDOM (JAVA DOM), CHE CONSENTE DI PROGRAMMARE DOCUMENTI XML IN APPLICAZIONI JAVA IN MODO PIÙ SEMPLICE ED EFFICIENTE.
JDOM NON È UNA VERSIONE JAVA DI DOM, MA SEMPLICEMENTE UN'API PIÙ JAVA-ORIENTED PER SEMPLIFICARE LA VITA DEL PROGRAMMATORE.

- HYBRID PARSING: QUESTO APPROCCIO COMBINA LE CARATTERISTICHE DEI 3 MODELLI DI PARSING PER CREARE PARSER EFFICIENTI PER SPECIFICI CONTESTI.
PER ESEMPIO, UN APPROCCIO COMUNE, CONSISTE NEL COMBINARE I MODELLI PULL PARSING E ONE-STEP: L'APPLICAZIONE PENSA DI OPERARE SU UN PARSE TREE CHE RISIEDE INTERAMENTE IN MEMORIA; QUANDO L'APPLICAZIONE RICHIEDE UN CERTO ELEMENTO, IL PARSER INCOMINCIA A LEGGERE IL DOCUMENTO FINCHÉ NON VIENE INDIVIDUATA L'INFORMAZIONE RICHIESTA; LE INFORMAZIONI LETTE VENGONO MEMORIZZATE ALL'INTERNO DI UN PARSE-TREE CORRISPONDENTE ALLA PORZIONE DI DOCUMENTO XML CHE È STATA LETTA DAL PULL PARSER.
I VANTAGGI SONO CHE NON VIENE SPRECATO TEMPO E SPAZIO PER LEGGERE PARTI DEL DOCUMENTO XML NON NECESSARIE, MA OGNI ELEMENTO LETTO VIENE CONSERVATO PER SUCCESSIVE ELABORAZIONI.

I FATTORI DA PRENDERE IN CONSIDERAZIONE PER LA SCELTA DEL MODELLO GIUSTO SONO:
- L'EFFICIENZA (SIA IN TERMINI DI MEMORIA OCCUPATA CHE DI ELABORAZIONE)
- LA SEMPLICITÀ DI UTILIZZO
- IL TIPO DI OPERAZIONI DA EFFETTUARE SUL DOCUMENTO.

LA TABELLA RIASSUNTIVA VANTAGGI E SVANTAGGI DI CIASCUN MODELLO

MODELLO DI PARSING	PRO	CONTRO
PULL PARSING	MOLTO EFFICIENTE IN TERMINI DI TEMPO BUON COMPROMESSO TRA MEMORIA OCCUPATA E FACILITÀ DI ESEGUIRE OPERAZIONI RIPETUTE SUL PARSE-TREE	DIFFICILE DA PROGRAMMARE INFATTI IL PROGRAMMATORE DEVE GESTIRE TUTTE LE ENTRY PRESENTI IN UN DOCUMENTO XML

PUSH PARSING	EFFICIENTE SIA IN TERMINI DI TEMPO CHE DI MEMORIA	DIFFICOLTÀ DI PROGRAMMARE PER OGNI UNITÀ DA RECUPERARE SI DEVE RICHIEDERE IL DOCUMENTO
ONE-STOP	FACILE DA PROGRAMMARE. ATTUALIZZA IL COSTO DI NUMEROSE OPERAZIONI SUL PARSE-TREE. CONSENTE DI NAVIGARE E MANIPOLARE IL PARSE-TREE	INEFFICIENTE IN TERMINI DI MEMORIA E TEMPO

PER RENDERE INDIPENDENTE L'APPLICAZIONE DALLA IMPLEMENTAZIONE DEL PARSER E FAVORIRE LA PORTABILITÀ DEL CODICE, LA COMUNITÀ JAVA HA DEFINITO UN'API STANDAR PER ISTANZIARE PARSER XML E FARE IL PARSING DI DOCUMENTI XML USANDO SAX O DOM. QUESTA API SI CHIAMA JAXP (JAVA API FOR XML PROCESSING)

SERIALIZZAZIONE

IL PARSER SI OCCUPA SOLTANTO DEL PROCESSO DI FORNIRE ALL'APPLICAZIONE INFORMAZIONI ESTRATTE DA UN DOCUMENTO XML. SIA SAX CHE DOM NON SI OCCUPANO INVECE DI PREVEDERE COME L'APPLICAZIONE POSSA SALVARE INFORMAZIONI ALL'INTERNO DI UN DOCUMENTO XML.

QUESTO PROBLEMA PUÒ ESSERE AFFRONTATO A VARI LIVELLI DI ASTRAZIONE:

- UN'APPLICAZIONE PUÒ PROVVEDERE A SCRIVERE DIRETTAMENTE NEL FILE XML, PREOCCUPANDOSI DI GARANTIRE IL RISPETTO DELLA SINTASSI XML. QUESTO APPROCCIO È POCO EFFICACE E SI PRESTA FACILMENTE AD ERRORI.
- UN APPROCCIO PIÙ SEMPLICE È QUELLO DI CREARE UNA RAPPRESENTAZIONE IN MEMORIA DEL DOCUMENTO CHE SI VUOLE CREARE (AD ESEMPIO UN PARSE-TREE) E POI INVOCARE UN OGGETTO SERIALIZZATORE CHE PROVVEDE A NAVIGARE NELLA RAPPRESENTAZIONE IN MEMORIA E GENERARE IL CODICE XML CHE RAPPRESENTA LE SUE INFORMAZIONI.

JAXP FORNISCE UN'INTERFACCIA TRANSFORMER CHE PUÒ ESSERE UTILIZZATA PER GENERARE UN DOCUMENTO XML A PARTIRE DA UN OGGETTO DOM.

XML PROCESSING ORIENTATO AI DATI

TUTTI I MECCANISMI DESCRITTI FINO AD ORA PER FARE IL PARSING O GENERARE XML SONO ORIENTATI ALLA SINTASSI E COSTRINGONO L'APPLICAZIONE AD OPERARE SU ELEMENTI E PEZZI DI TESTO.

LE APPLICAZIONI ORIENTATE AI DATI INTRODUCONO TRA IL PARSER E L'APPLICAZIONE UN LIVELLO DI ASTRAZIONE DEI DATI CHE PRENDE LE UNITÀ DI INFORMAZIONE ESTRATTE DAL DOCUMENTO XML E LE TRASFORMA IN DATI PER L'APPLICAZIONE.

LE APPLICAZIONI ORIENTATE AI DATI OPERANO SECONDO DUE APPROCCI:

- APPROCCIO CENTRATO SULLE OPERAZIONI
- APPROCCIO CENTRATO SUI DATI

APPROCCIO CENTRATO SULLE OPERAZIONI

SECONDO L'APPROCCIO CENTRATO SULLE OPERAZIONI, L'APPLICAZIONE LAVORA SU UNA API COSTRUITA AD-HOC CHE FORNISCE UN METODO PER CIASCUNA DELLE OPERAZIONI DA EFFETTUARE SUL DOCUMENTO. L'INTERAZIONE TRA L'APPLICAZIONE E QUESTA API È BASATA SU DATI NATIVI, MENTRE L'IMPLEMENTAZIONE DELL'API RACCHIUDE TUTTI I DETTAGLI RELATIVI ALL'ELABORAZIONE DEL DOCUMENTO XML. IN QUESTO MODO, L'APPLICAZIONE NON SA NULLA DELLA STRUTTURA DEL DOCUMENTO XML.

APPROCCIO CENTRATO SUI DATI

UN APPROCCIO BASATO SUI DATI, RISOLVE IL PROBLEMA DI LAVORARE CON IL DOCUMENTO XML A CARICO DI MAPPARE GLI ELEMENTI XML IN DATI DELL'APPLICAZIONE E VICEVERSA. QUESTO CONSENTE ALL'APPLICAZIONE DI LAVORARE SU QUESTI DATI IN MANIERA COMPLETAMENTE INDIPENDENTE DALLA LORO PROVENIENZA.

IL PROCESSO DI CONVERTIRE I DATI DELL'APPLICAZIONE IN XML È DETTO MARSHALLING, MENTRE IL PROCESSO DI RAPPRESENTARE I DATI IN XML COME OGGETTI JAVA È DETTO UNMARSHALLING. ESISTONO TECNOLOGIE CHE CONSENTONO DI AUTOMATIZZARE LE OPERAZIONI DI MARSHALLING E UNMARSHALLING, UN ESEMPIO SONO I COMPILATORI DI SCHEMA.

UN COMPILATORE DI SCHEMA È UN TOOL CHE VIENE INVOCATO UNA SOLA VOLTA IN FASE DI PREPROCESSING, ANALIZZA UNO SCHEMA XML E GENERA I MODULI SOFTWARE PER FARE IL MARSHALLING E L'UNMARSHALLING.

I MODULI DI MARSHALLING E UNMARSHALLING, LAVORANO CON STRUTTURE DATI CREATE DALLO SCHEMA XML. È POSSIBILE FORNIRE AL COMPILATORE DELLE ISTRUZIONI PER CUSTOMIZZARE IL BINDING TRA L'XML E LE STRUTTURE DATI DELL'APPLICAZIONE.

I PROTOCOLLI PER CUI È NECESSARIO È CORREDATO DI UNA SPECIFICAZIONE DEI BINDING SONO È:

- L'APPLICAZIONE DEVE UTILIZZARE DELLE STRUTTURE DATI PREDEFINITE. IN QUESTO CASO BISOGNA ADATTARE IL BINDING IN MODO DA MAPPARE L'XML SULLE STRUTTURE DATI DEFINITE DALL'APPLICAZIONE.
- IL BINDING DI DEFAULT PRODUCE DELLE API COMPLESSE.

LA COMUNITÀ JAVA HA DEFINITO UN INSIEME STANDARD DI TOOL E API PER MAPPARE I TIPI DI UNO SCHEMA XML IN STRUTTURE DATI JAVA CHE SONO RACCOLTE SOTTO IL NOME DI JAXB (JAVA ARCHITECTURE FOR XML BINDING).

SAX E DOM

JAXP

JAXP (JAVA API FOR XML PROCESSING) È UN'API JAVA STANDARD CHE FORNISCE ALL'UN'APPLICAZIONE JAVA IL SUPPORTO PER PROCESSARE E MANIPOLARE DOCUMENTI XML.
LA SPECIFICA DI JAXP È BASATA SU ALTRE TECNOLOGIE COME AD ESEMPIO:

- XML
- NAMESPACES IN XML
- XML SCHEMA
- DOM LEVEL 2/LEVEL 3
- SAX
- STAX

L'OBIETTIVO DI JAXP È DI FORNIRE ALL'APPLICAZIONE UN'INTERFACCIA STANDARD ALLE TECNOLOGIE ELENCATE SOPRA CHE SIA INDIPENDENTE DALL'IMPLEMENTAZIONE DA UTILIZZARE. QUESTO OBIETTIVO VIENE PERSEGUITO DA JAXP ATTRAVERSO L'INTRODUZIONE DI UN MECCANISMO DI "PLUGABILITY", CHE CONSENTE A TUTTE LE IMPLEMENTAZIONI CONFORMI ALLE SPECIFICHE SUPPORTATE DA JAXP DI ESSERE UTILIZZATE DA APPLICAZIONI JAVA ATTRAVERSO UN'API STANDARD.
I MECCANISMI DI PLUGABILITY SONO:

- MECCANISMO DI PLUGABILITY PER SAX
- MECCANISMO DI PLUGABILITY PER DOM
- MECCANISMO DI PLUGABILITY PER XSLT (TRASFORMER)

MECCANISMO DI PLUGGABILITY PER SAX

LE CLASSI CHE IMPLEMENTANO IL MECCANISMO DI PLUGGABILITY DI SAX, CONSENTONO AL PROGRAMMATORE DI UN'APPLICAZIONE DI ISTANZIARE UN PARSER SAX, INDIPENDENTEMENTE DA QUALE IMPLEMENTAZIONE SI DECIDE DI UTILIZZARE, FORNENDO UN'IMPLEMENTAZIONE DELL'API ORG.XML.SAX.DEFAULTHANDLER ALL'IMPLEMENTAZIONE DEL SAXPARSER E DI FARE IL PARSING DEL DOCUMENTO XML.

DURANTE IL PROCESSO DI PARSING IL PARSER INVOCHERÀ I METODI DELLA IMPLEMENTAZIONE DEFAULTHANDLER FORNITA.

PER ISTANZIARE UN PARSER SAX, JAXP UTILIZZA IL PATTERN DELLA FACTORY.

PER EFFETTUARE IL PARSING DEL DOCUMENTO XML, L'APPLICAZIONE DOVRÀ ESEGUIRE LE SEGUENTI 3 OPERAZIONI:

- OTTENERE UN'ISTANZA DELLA CLASSE SAXPARSERFACTORY ATTRAVERSO L'INVOCAZIONE DEL METODO STATIC SAXPARSERFACTORY.NEWISTANCE().
- CREARE UN'ISTANZA DI SAXPARSER ATTRAVERSO L'INVOCAZIONE DEL METODO NEWSAXPARSER() DELLA CLASSE SAXPARSERFACTORY.
- ESEGUIRE IL METODO PARSE() SULL'OGGETTO SAXPARSER.

IL METODO STATIC NEWISTANCE() DELLA FACTORY, PERMETTE DI SELEZIONARE LA PARTICOLARE IMPLEMENTAZIONE DEL PARSER SAX CHE SI VUOLE UTILIZZARE.

LA PROCEDURA DI SELEZIONE DELL'IMPLEMENTAZIONE È LA SEGUENTE:

- UTILIZZA LA PROPRIETÀ DI SISTEMA JAVAX.XML.PARSERS.SAXPARSERFACTORY PER INDIVIDUARE L'IMPLEMENTAZIONE DA CARICARE.
- UTILIZZA IL FILE DI PROPRIETÀ LIB/JAXP.PROPERTIES, PRESENTE NELLA DIRECTORY JRE DI JAVA CHE CONTIENE IL NOME QUALIFICATO DELLA CLASSE DI IMPLEMENTAZIONE E LA CHIAVE CHE È IL VALORE DELLA PROPRIETÀ DI SISTEMA JAVAX.XML.PARSERS.SAXPARSERFACTORY.
- UTILIZZA L'API SERVICES, PER LEGGERE NEL FILE META-INF/SERVICES/JAVAX.XML.PARSERS.SAXPARSERFACTORY IL NOME DELLA CLASSE DA ISTANZIARE.

LA SEGUENTE TABELLA ELENCA I METODI PRINCIPALI DELLA CLASSE SAXPARSERFACTORY:

ABSTRACT BOOLEAN GETFEATURE(STRING NAME);

BOOLEAN ISVALIDATING();

STATIC SAXPARSERFACTORY NEWINSTANCE();

ABSTRACT SAXPARSER NEWSAXPARSER();

ABSTRACT VOID SETFEATURE(STRING NAME,
BOOLEAN VALUE);

VOID SETNAMESPACEAWARE(BOOLEAN AWARENESS);

VOID SETVALIDATING(BOOLEAN VALIDATING);

UTILIZZANDO IL METODO CREATO SAXPARSER È POSSIBILE ISTANZIARE UN OGGETTO SAXPARSER SECONDO LA CONFIGURAZIONE FISSATA.
PER EFFETTUARE IL PARSING SI INVOCA IL METODO PARSE SULL'OGGETTO SAXPARSER, PASSANDOGLI IN INPUT IL NOME DEL DOCUMENTO XML DA IL RIFERIMENTO ALL'IMPLEMENTAZIONE DI DEFAULTHANDLER CHE DEVE ESSERE UTILIZZATA.

MECCANISMO DI PLUGABILITY PER DOM

LE CLASSI CHE IMPLEMENTANO IL MECCANISMO DI PLUGABILITY DI DOM, CONSENTONO AL PROGRAMMATORE DI UN'APPLICAZIONE DI ISTANZIARE UN PARSER DOM CHE FACCIA IL PARSING DEL DOCUMENTO XML E PRODUCA UN OGGETTO ORG.W3C.DOM.DOCUMENT, INDIPENDENTEMENTE DA QUALE IMPLEMENTAZIONE SI DECIDE DI UTILIZZARE

ANCHE PER ISTANZIARE UN PARSER DOM, SI UTILIZZA IL PATTERN DELLA FACTORY.

PER EFFETTUARE IL PARSING DEL DOCUMENTO XML, L'APPLICAZIONE DEVE ESEGUIRE LE SEGUENTI 3 OPERAZIONI:

- OTTENERE UN'ISTANZA DELLA CLASSE DOCUMENTBUILDERFACTORY ATTRAVERSO L'INVOCAZIONE DEL METODO STATIC DOCUMENTBUILDERFACTORY.NEWISTANCE().
- CREARE UN'ISTANZA DI DOCUMENTBUILDER ATTRAVERSO L'INVOCAZIONE DEL METODO NEWDOCUMENTBUILDER() DELLA CLASSE DOCUMENTBUILDERFACTORY;
- ESEGUIRE IL METODO PARSE() SULL'OGGETTO DOCUMENBUILDER CHE RESTITUISCE UN OGGETTO ORG.W3C.DOM.DOCUMENT.

LA SELEZIONE DELLA IMPLEMENTAZIONE DEL PARSER DOM CHE LA FACTORY DEVE UTILIZZARE È FATTA USANDO LA STESSA PROCEDURA UTILIZZATA DA SAXPARSERFACTORY.

LA SEGUENTE TABELLA RIASSUME I METODI PRINCIPALI DELLA CLASSE DOCUMENTBUILDERFACTORY:

ABSTACT OBJECT GETATTRIBUTE(STRING NAME);

BOOLEAN ISNAMESPACEAWARE();

BOOLEAN ISVALIDATING();

STATIC DOCUMENTBUILDERFACTORY NEWISTANCE();

ABSTRACT DOCUMENTBUILDER NEWDOCUMENTBUILDER();

ABSTRACT VOID SETATTRIBUTE(STRING NAME, OBJECT VALUE);

ABSTRACT VOID SETNAMESPACEAWARE(BOOLEAN AWARENESS);

VOID SETVALIDATING(BOOLEAN VALIDATING);

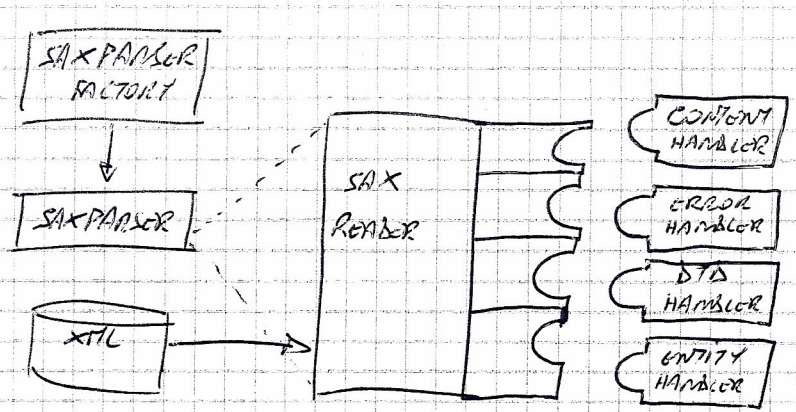
UTILIZZANDO IL METODO NEWDOCUMENTBUILDER È POSSIBILE ISTANZIARE UN OGGETTO DOCUMENTBUILDER SECONDO LA CONFIGURAZIONE PASSATA.
PER ESEGUIRE IL PARSING SI INVOCA IL METODO PARSE SULL'OGGETTO DOCUMENTBUILDER, PASSANDOGLI IN INPUT IL NOME DEL DOCUMENTO XML. QUESTO METODO CREERÀ UN OGGETTO DOCUMENT IN MEMORIA E RESTITUIRÀ UN RIFERIMENTO DI ESSO.
LA CLASSE DOCUMENTBUILDER RIUTILIZZA ALCUNE CLASSI DELLE API SAX, IN PARTICOLARE LE CLASSI PER LA GESTIONE DEGLI ERRORI.

MECCANISMO DI PLUGGABILITY PER XSLT (TRANSFORMER)

LE CLASSI CHE IMPLEMENTANO IL MECCANISMO DI PLUGGABILITY DI XSLT, CONSENTONO AL PROGRAMMATORE DI UN'APPLICAZIONE DI OTTENERE UN OGGETTO TRANSFORMER IL CUI FUNZIONAMENTO È BASATO SULLE REGOLE FORNITE DA UN FOGLIO DI STILE XSLT. ANCHE IN QUESTO CASO, L'ISTANZIAZIONE DI OGGETTI TRANSFORMER È BASATA SUL PATTERN FACTORY: OGGETTI TRANSFORMER SONO ISTANZIATI DA UNA TRANSFORMERFACTORY CHE VIENE ISTANZIATA INVOCANDO IL METODO STATICO NEWINSTANCE().
LA SCELTA DELLA PARTICOLARE IMPLEMENTAZIONE DI TRANSFORMER DA UTILIZZARE È FATTA SECONDO LO STESSO ALGORITMO USATO DA SAXPARSERFACTORY E DOCUMENTBUILDER.
LA CLASSE TRANSFORMER VERRÀ UTILIZZATA PER CREARE DOCUMENTI XML.

SAX

L'ARCHITETTURA DI BASE DI UN'APPLICAZIONE CHE UTILIZZA JAXP PER ISTANZIARE UN PARSER SAX È DATA DALLA SEGUENTE FIGURA:



LA SAXPARSERFACTORY VIENE USATA PER ISTANZIARE UN OGGETTO SAXPARSER. QUESTO OGGETTO INCAPSULA AL SUO INTERNO UN OGGETTO SAXREADER, CHE È L'OGGETTO CHE EFFETTIVAMENTE DEVE FARE IL PARSING DEL DOCUMENTO XML.
IL PARSER SAX È UN PARSER EVENT-DRIVEN: IL PARSER LEGGE SEQUENZIALMENTE IL DOCUMENTO XML ED OGNI VOLTA CHE INDIVIDUA UN'UNITÀ DI INFORMAZIONE LANCIA UN EVENTO CHE IMPLICA L'INVOCAZIONE DI UN METODO CALLBACK NEL CODICE DELL'APPLICAZIONE.
QUESTI METODI SONO RACCOLTI ALL'INTERNO DI 4 INTERFACCE:

- CONTENTHANDLER
- ERRORHANDLER
- DTDHANDLER
- ENTITYRESOLVER

LA CLASSE SAXPARSER FORNISCE DIVERSE VERSIONI DEL METODO PARSE(), OGNUNA DELLE QUALI PRENDE IN INPUT IL DOCUMENTO DA ANALIZZARE ED IL RIFERIMENTO AD UN OGGETTO CHE IMPLEMENTA L'INTERFACCIA DEFAULTHANDLER. QUANDO IL METODO PARSE VIENE INVOCATO IL SAXPARSER INCOMINCIA A LEGGERE IL DOCUMENTO XML ED OGNI VOLTA CHE INDIVIDUA UN EVENTO DI INTERESSE INVOCA UNO DEI METODI CALLBACK DELL'INTERFACCIA CONTENTHANDLER.
I PRINCIPALI METODI DELL'INTERFACCIA CONTENTHANDLER SONO:

- void startDocument();
- void endDocument();
- void startElement();
- void endElement();
- void characters();

L'INTERFACCIA ERRORHANDLER CONTIENE I METODI CALLBACK CHE VENGONO INVOCATI QUANDO IL [redacted] SAXREADER INDIVIDUA UN ERRORE. OGNUNO DI QUESTI METODI PRENDE IN INPUT UN'ECCEZIONE SAXPARSEEXCEPTION.

L'IMPLEMENTAZIONE DI DEFAULTHANDLER INTERAGISCE SOLO CON ERRORI DI LIVELLO FORTISSIMO (ERRORI FATALI) CHE IGNORA OGNI SUO VALIDAZIONE.

I METODI DELL'INTERFACCIA ERRORHANDLER SONO:

- VOID ERROR():
- VOID FATALERROR():
- VOID WARNING():

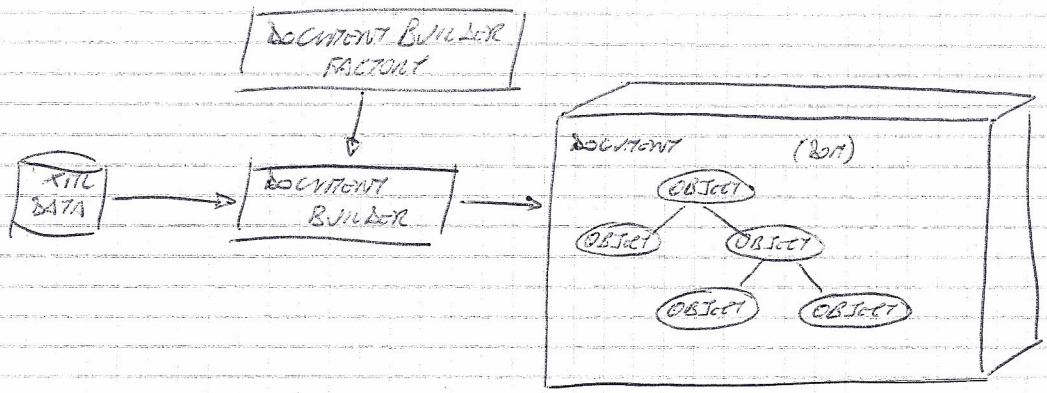
L'INTERFACCIA DTDHANDLER CONTIENE METODI CHE HANNO A CHE FARE CON L'ELABORAZIONE DELLE DTD.

L'INTERFACCIA ENTITYRESOLVER, CONTIENE METODI CHE SERVONO PER INDIVIDUARE LA POSIZIONE DI ENTITÀ ESTERNE CHE SONO REFERENZIATE ALL'INTERNO DEL DOCUMENTO XML.

UN'APPLICAZIONE CHE VUOLE UTILIZZARE UN PARSER SAX DEVE FORNIRE UNA PROPRIA IMPLEMENTAZIONE DELLE INTERFACCE UTILIZZATE (IN PARTICOLARE CONTENTHANDLER) LA CLASSE DEFAULTHANDLER FORNISCE UN'IMPLEMENTAZIONE DI DEFAULT I CUI METODI NON FANNO NULLA. PER PROCESSARE IL DOCUMENTO SECONDO LA LOGICA DELL'APPLICAZIONE BISOGNA ESTENDERE LA CLASSE DEFAULTHANDLER FORNENDO UN'IMPLEMENTAZIONE DEI METODI CORRISPONDENTI AGLI EVENTI CHE SI VUOLE GESTIRE.

DOM

L'ARCHITETTURA BASE DI UN'APPLICAZIONE CHE UTILIZZA JAXP PER ISTANZIARE UN PARSER DOM È DATA NELLA SEGUENTE FIGURA:



LA CLASSE DOCUMENTBUILDERFACTORY VIENE USATA PER ISTANZIARE UN OGGETTO DOCUMENTBUILDER. QUESTO OGGETTO VIENE UTILIZZATO PER FARE IL PARSING DEL DOCUMENTO XML E CREARE UN OGGETTO DOCUMENT SECONDO LE SPECIFICHE DI DOM.

DOM DEFINISCE DELLE API CHE PERMETTONO ALL'APPLICAZIONE DI NAVIGARE ALL'INTERNO DELL'OGGETTO DOCUMENT, MANIPOLARE IL CONTENUTO ED EVENTUALMENTE MODIFICARNE LA STRUTTURA.

LA SPECIFICA DI DOM HA DIVERSI LIVELLI:

- DOM LEVEL 1: FORNISCE SOLO LE FUNZIONALITÀ BASE PER LA NAVIGAZIONE E LA MANIPOLAZIONE DI DOCUMENTI XML.
- DOM LEVEL 2: È COSTITUITA DA VARIE PARTI, LA PRINCIPALE È DOM LEVEL 2 CORE CHE RISOLVE ALCUNI PROBLEMI DI DOM LEVEL 1 E DEFINISCE NUOVI ADDITIVI PER NAVIGARE IN DOCUMENTI XML E MANIPOLARE IL CONTENUTO E LA STRUTTURA DEL DOCUMENTO. DOM LEVEL 2 SUPPORTA COMPLETAMENTE I NAMESPACE XML.
- DOM LEVEL 3: RECENTEMENTE PUBBLICATO, IL CUI SUPPORTO PERÒ, È ANCORA LIMITATO.

DOM È STATA LA PRIMA SPECIFICA CHE È STATA FORNITA PER IL PARSING. L'IDEA DI BASE DI DOM (E QUINDI DEI PARSER ONE-STEP) È DI CREARE IN MEMORIA UN OGGETTO DOCUMENT CHE RAPPRESENTI IL DOCUMENTO XML, RIFLETTENDONE IL PIÙ POSSIBILE LA STRUTTURA. LE INTERFACCE DI DOM OPERANO SU ELEMENTI, ATTRIBUTI, E [redacted] DESCRIVONO LA SINTASSI XML A BASSO LIVELLO. PER QUESTO MOTIVO, DOM NON È ADATTO PER PROCESSARE DOCUMENTI DATA-ORIENTED (QUESTO È IL MOTIVO PRINCIPALE DELLA NASCITA [redacted] DI JDOM).

LE INTERFACCE DI PROGRAMMAZIONE DOM SONO: NODELIST, CHARACTERDATA, TEXT, NODE, DOCUMENT, NAMEDNODEMAP, ELEMENT, ATTR.

L'INTERFACCIA PRINCIPALE È NODE. UN NODE RAPPRESENTA IL TIPO BASE DI OGNI ELEMENTO DEL DOCUMENT, DA CUI SONO DERIVATE TUTTE LE INTERFACCE CHE DESCRIVONO I VARI TIPI DI ELEMENTI PRESENTI IN UN DOCUMENTO XML.
TRA LE PRINCIPALI OPERAZIONI FORNITE DA NODE CI SONO:

- RECUPERARE IL NOME ED IL TIPO DI UN NODO, IL SUO NAMESPACE E LA LISTA DEI SUOI ATTRIBUTI (GETNODENAME(), GETNODETYPE(), GETNAMESPACEURI(), GETNODEATTRIBUTES())
- LEGGERE E MODIFICARE IL VALORE DI UN NODO (GETNODEVALUE(), SETNODEVALUE()).
- NAVIGARE TRA I FIGLI, I FRATELLI ED IL PADRE DI UN NODO (GETFIRSTCHILD(), GETLASTCHILD(), GETNEXTSIBLING(), GETPREVIOUSSIBLING(), GETCHILDNODES(), GETPARENTNODE()).
- AGGIUNGERE, RIMUOVERE E SOSTITUIRE FIGLI (APPENDCHILD(), REMOVECHILD(), REPLACECHILD()).

A PARTIRE DA NODE SONO DERIVATE LE INTERFACCE ATTR, [redacted], CHARACTERDATA, DOCUMENT, ELEMENT, TEXT.

L'INTERFACCIA DOCUMENT RAPPRESENTA L'INTERO DOCUMENTO. LE OPERAZIONI PIÙ IMPORTANTI SONO:

- LA CREAZIONE DI NODI DI OGNI TIPO
- L'IMPORTAZIONE NEL DOCUMENT DI NODI DA UN ALTRO DOCUMENT
- IL RECUPERO DELL'OGGETTO DOMIMPLEMENTATION CHE DEFINISCE L'IMPLEMENTAZIONE DELLE VARIE INTERFACCE DI DOM.
- IL RECUPERO DEL NODO RADICE DEL DOCUMENT (GETDOCUMENTELEMENT()).
- LA RICERCA DI ELEMENTI NEL DOCUMENT IN BASE AL NOME / ALL'ID (GETELEMENTSBYTAGNAME(), GETELEMENTSBYTAGNAMENS(), GETELEMENTBYID()).

IN PARTICOLARE IL METODO GETELEMENTSBYTAGNAME() È ESTREMAMENTE UTILE PERCHÉ CONSENTE DI INDIVIDUARE TUTTI I NODI DELL'ALBERO CON UN DATO NOME SENZA DOVER NAVIGARE ALL'INTERNO DEL DOCUMENTO.

L'INTERFACCIA ELEMENT RAPPRESENTA UN ELEMENTO DEL DOCUMENTO E FORNISCE I METODI PER OPERARE SULLE INFORMAZIONI CONTENUTE ALL'INTERNO DELLO START-TAG DELL'ELEMENTO. IN PARTICOLARE QUESTA INTERFACCIA FORNISCE I SEGUENTI METODI:

- VERIFICARE SE ESISTE UN ATTRIBUTO CON UN DATO NOME, RECUPERARLO, RIMUOVERLO O AGGIUNGERLO (HASATTRIBUTE(), GETATTRIBUTE(), REMOVEATTRIBUTE(), SETATTRIBUTE()).
- RECUPERARE IL NOME DELL'ELEMENTO (GETTAGNAME()).
- RECUPERARE LA LISTA DEI FIGLI DI UN ELEMENTO CON UN DATO NOME (GETELEMENTSBYTAGNAME()).

L'INTERFACCIA ATTR, RAPPRESENTA UN ATTRIBUTO DI UN ELEMENTO E FORNISCE I METODI PER OTTENERE E DARE IL VALORE (GETVALUE(), SETVALUE()).
LE INTERFACCE NODELIST E NAMEDNODEMAP, CONSENTONO DI OPERARE CON COLLEZIONI DI NODI (COME QUELLE RESTITUITE DAL METODO GETELEMENTSBYTAGNAME()).

UTILIZZO DI PARSER VALIDANTI

LA VALIDAZIONE DEL DOCUMENTO XML VIENE EFFETTUATA AUTOMATICAMENTE DAL PARSER XML. SE SI VUOLE AVERE UN PARSER VALIDANTE, BISOGNA CONFIGURARE LA FACTORY IN MODO CHE ISTANZI UN PARSER VALIDANTE; FORNIRE AL PARSER LO SCHEMA XML DA UTILIZZARE; SPECIFICARE L'OPERAZIONE CHE DEVE ESSERE EFFETTUATA IN CASO DI ERRORE DI VALIDAZIONE.
PER CONFIGURARE LA FACTORY È NECESSARIO SETTARE A TRUE LA PROPRIETÀ CORRISPONDENTE. QUESTO PUÒ ESSERE FATTO UTILIZZANDO IL METODO SETVALIDATING() DI DOCUMENTBUILDERFACTORY, PASSANDOGLI IN INPUT IL VALORE TRUE.
PER SPECIFICARE IL NOME DELLO SCHEMA DA UTILIZZARE PER LA VALIDAZIONE SI DEVE USARE IL METODO SETPROPERTY() DI DOCUMENTBUILDER PER SETTARE DUE PROPRIETÀ:
- SETTARE LA PROPRIETÀ "...../SCHEMALANGUAGE", IL CUI VALORE DEVE ESSERE UNA URI CHE IDENTIFICA IL LINGUAGGIO UTILIZZATO.
- SETTARE LA PROPRIETÀ "...../SCHEMASOURCE", IL CUI VALORE DEVE IDENTIFICARE LO SCHEMA DA UTILIZZARE.

LA PROCEDURA UTILIZZATA DA JAXP PER LOCALIZZARE LO SCHEMA È LA SEGUENTE:
- SE LA PROPRIETÀ "...../SCHEMASOURCE" È STATA SETTATA ED IL SUO VALORE È UGUALE A QUELLO DELL'ATTRIBUTO SCHEMALOCATION PRESENTE NELL'ISTANZA, ALLORA LO SCHEMA INDICATO DALLA PROPRIETÀ VERRÀ EFFETTIVAMENTE CARICATO.
- SE IL VALORE DELLA PROPRIETÀ "...../SCHEMASOURCE" È DIVERSO DA QUELLO DELL'ATTRIBUTO SCHEMALOCATION PRESENTE NELL'ISTANZA, VIENE CARICATO LO SCHEMA INDICATO NELL'ISTANZA.

CREAZIONE DI DOCUMENTI XML

IL PARSER SI OCCUPA SOLTANTO DI ESTRARRE UNITÀ DI INFORMAZIONE DA UN DOCUMENTO XML E FORNIRLE TALI DATI AD UN'APPLICAZIONE. SIA SAX CHE DOM, QUINDI, NON FORNISCONO API PER CONSENTIRE LA CREAZIONE O LA MODIFICA DI UN DOCUMENTO XML. IN PARTICOLARE, LE INTERFACCE DI DOM PERMETTONO DI MANIPOLARE STRUTTURA E CONTENUTO DELL'OGGETTO DOCUMENT IN MEMORIA, MA NON FORNISCONO STRUMENTI PER SERIALIZZARE QUESTO OGGETTO IN UN FILE XML.

JAXP FORNISCE UN'API JAVA STANDARD PER CREARE DOCUMENTI XML. LA SOLUZIONE FORNITA DA JAXP È BASATA SUL SUO MECCANISMO DI PLUGGABILITY DI TASTE, CHE CONSENTE AL PROGRAMMATORE DI UN'APPLICAZIONE DI DEFINIRE UN OGGETTO TRANSFORMER.

PER LAVORARE CON JAXP E TRANSFORMER, SI UTILIZZA IL SEGUENTE PACKAGE:

- JAVAX.XML.TRANSFORM (DEFINISCE LE CLASSI TRANSFORMFACTORY E TRANSFORMER).

I PRINCIPALI METODI DELLA CLASSE TRANSFORMER SONO:

- CLEARPARAMETERS()
- GETERRORLISTENER()
- GETOUTPUTPROPERTIES()
- GETPARAMETER()
- GETURIRESOLVER()
- SETERRORLISTENER()
- SETOUTPUTPROPERTIES()
- SETOUTPUTPROPERTY()
- SETPARAMETER()
- SETURIRESOLVER()
- TRANSFORM()

IL METODO PRINCIPALE È TRANSFORM, CHE PRENDE IN INPUT UN OGGETTO SOURCE E RESTITUISCE UN OGGETTO RESULT. L'INTERFACCIA SOURCE CONTIENE I METODI NECESSARI AD OPERARE SU UN SOURCE TREE CHE RAPPRESENTA UN DOCUMENTO XML O HTML.
LE CLASSI CHE IMPLEMENTANO L'INTERFACCIA SOURCE SONO:
- DOMSOURCE (RAPPRESENTA IL SOURCE-TREE COME UN OGGETTO DOCUMENT)
- SAXSOURCE (RAPPRESENTA IL SOURCE-TREE MEDIO STILE DI SAX)
- STREAMSOURCE (RAPPRESENTA IL SOURCE-TREE COME UNO STREAM DI NARRATORI XML)

L'INTERFACCIA RESULT, CONTIENE LE INFORMAZIONI NECESSARIE A PRODURRE L'ALBERO FRUTTO DELLA TRASFORMAZIONE.
LE CLASSI CHE IMPLEMENTANO L'INTERFACCIA RESULT SONO:
- SAXRESULT (PRODUCE UNA SERIE DI EVENTI CHE VENGONO CATTURATI DAI METODI CALLBACK DI DEFAULTHANDLER)
- DOMRESULT (PRODUCE UN OGGETTO DOCUMENT)
- STREAMRESULT (PRODUCE UNO STREAM DI NARRATORI XML)

CI SONO VARI MODI DI UTILIZZARE L'INTERFACCIA TRANSFORMER PER CREARE UN DOCUMENTO XML. UNA SOLUZIONE SEMPLICE CONSISTE NEL CREARE IN MEMORIA UN OGGETTO DOCUMENT E QUINDI INVOCARE IL METODO TRANSFORM PASSANDOGLI IN INPUT IL DOCUMENTO COME UN DOMSOURCE E IL FILE DA CREARE COME UNO STREAMRESULT.
PER LA CREAZIONE DEL DOCUMENTO POSSIAMO UTILIZZARE LE API DI DOM CHE CONSENTONO DI MANIPOLARE STRUTTURA E CONTENUTI DEI NODI DEL DOCUMENTO.

JDOM

JDOM È UNA LIBRERIA OPEN SOURCE PER IL PROCESSING DI DOCUMENTI XML OTTIMIZZATA PER JAVA. JDOM NON È UNA VERSIONE DI DOM. SI POTREBBE DIRE CHE LE SOMIGLIANZE SI LIMITANO AL FATTO CHE ANCHE JDOM UTILIZZA UNA CLASSE DOCUMENT PER RAPPRESENTARE IN MEMORIA IL DOCUMENTO XML, MA LA STRUTTURA DI QUESTA CLASSE È MOLTO DIVERSA DA QUELLA DI DOM E LA SUA CREAZIONE NON PRESUPPONE L'UTILIZZO DI UN PARSER DOM.

LE INTERFACCE DI JDOM SONO STATE COSTRUITE PER ESSERE UTILIZZATE ALL'INTERNO DI UN'APPLICAZIONE JAVA E SFRUTTANO TUTTE LE FUNZIONALITÀ DI QUESTO LINGUAGGIO, AL CONTRARIO DI DOM CHE È STATO PROGETTATO PER ESSERE NEUTRALE RISPETTO AL LINGUAGGIO.

LA LIBRERIA JDOM CONSISTE DI 6 PACKAGES:

- ORG.JDOM
- ORG.JDOM.INPUT (CONTIENE LE CLASSI BUILDER PER COSTRUIRE DOCUMENTI XML)
- ORG.JDOM.OUTPUT (CONTIENE LE CLASSI OUTPUTTER PER PRODURRE IN OUTPUT DOCUMENTI XML)
- ORG.JDOM.TRANSFORM
- ORG.JDOM.XPATH
- ORG.JDOM.ADAPTERS

LE CLASSI BUILDER, PERMETTONO DI COSTRUIRE UN OGGETTO DOCUMENT A PARTIRE DAI RISULTATI PRODOTTI DA UN PARSER. IN PARTICOLARE, SAXBUILDER COSTRUISCE IL DOCUMENT A PARTIRE DAGLI EVENTI PRODOTTI DA UN PARSER SAX.

SAXBUILDER, FORNISCE UN'IMPLEMENTAZIONE DEI METODI CALLBACK CHE IMPLEMENTA LA CREAZIONE DEL DOCUMENT.

DOMBUILDER, COSTRUISCE UN DOCUMENT DI JDOM A PARTIRE DA UN DOM TREE.

POICHÉ L'UTILIZZO DEL BUILDER È TOTALMENTE TRASPARENTE ALL'UTENTE, L'APPROCCIO GENERALMENTE SEGUITO È:

- UTILIZZA SAXBUILDER SE IL DOCUMENT DEVE ESSERE COSTRUITO A PARTIRE DA UNO STREAM (PER ES. UN FILE).
- UTILIZZA DOMBUILDER SE IL DOCUMENT DEVE ESSERE COSTRUITO A PARTIRE DA UN DOM TREE PREESISTENTE.

RESULTBUILDER, È UN BUILDER MOLTO POPOLARE CHE CONSENTE DI COSTRUIRE UN DOCUMENT A PARTIRE DAI RISULTATI DI UNA QUERY SQL.

LE CLASSI OUTPUTTER, CONVERTONO I DOCUMENTI DI JDOM IN ALTRI OGGETTI. IN PARTICOLARE,

XMLOUTPUTTER, PRODUCE UNA SERIE DI INDICAZIONI XML CHE POI POSSONO INVIATO IN OUTPUT AD UN FILE, SOCKET ED OGNI ALTRO TIPO DI STREAM.

SAXOUTPUTTER, DATO IL CONTENUTO DEL DOCUMENTO E GENERA EVENTI CHE POSSONO ESSERE RICEVUTI DA UN PARSER SAX.

DOMOUTPUTTER, PRODUCE UN DOM TREE.

UN OGGETTO DOCUMENT CREATO, HA UNA STRUTTURA GERARCHICA CHE RIFLETTE FEDELMENTE LA STRUTTURA DEL DOCUMENTO XML, SENZA LO STRATTO DI DOM.

PER COSTRUIRE UN DOCUMENTO DA ZERO È SUFFICIENTE RICHIAMARE UN OGGETTO DOCUMENT E AGGIUNGERGLI VIA VIA I VARI ELEMENTI.

SE INVECE, SI VUOLE COSTRUIRE IL DOCUMENTO A PARTIRE DA UN FILE ESISTENTE, SI DOVE UTILIZZARE UN SAXBUILDER.

COSTRUZIONE DI UN DOCUMENTO XML

PER CREARE UN DOCUMENTO XML BISOGNA USARE LA CLASSE XMLOUTPUTTER. PER DEFAULT, QUESTA CLASSE COSTRUISCE UN DOCUMENTO XML CHE RIPRODUCE LA SUA RAPPRESENTAZIONE IN MEMORIA. XMLOUTPUTTER HA ANCHE UNA SERIE DI METODI CHE CONSENTONO DI DEFINIRE LA FORMATTAZIONE DEL FILE CREATO.

NAVIGAZIONE ALL'INTERNO DEL DOCUMENTO

JDOM CONSENTE UNA NAVIGAZIONE ALL'INTERNO DEL DOCUMENTO PIÙ AGEVOLE DI QUANTO FA DOM, ANCHE GRAZIE ALLA DIVERSA STRUTTURA DEI DOCUMENTI, INOLTRE JDOM SUPPORTA TOTALMENTE I NAMESPACES XML.

JDOM FORNISCE ANCHE SUPPORTO A TRASFORMAZIONI XSLT. LE CLASSI FONDAMENTALI SONO:

- JDOMSOURCE (FORNISCE UN DOCUMENT JDOM AL TRANSFORMER)
- JDOMRESULT (PRENDE IL RISULTATO DELLA TRASFORMAZIONE E LO RAPPRESENTA COME UN DOCUMENT JDOM)

STRUTTURA DEI MESSAGGI SOAP E CONVENZIONE RPC

PROTOCOLLO SOAP E SUO UTILIZZO PER LO SVILUPPO DI WEB SERVICES

DEFINIZIONE DI WEB SERVICE:
UN WEBSERVICES È UN SISTEMA SOFTWARE PROGETTATO PER SUPPORTARE INTERAZIONI INTEROPERABILI TRA APPLICAZIONI DIFFERENTI ATTRAVERSO LA RETE.
IL WEB SERVICE HA UN'INTERFACCIA DESCRITTA IN UN FORMATO PROCESSABILE DA UN'APPLICAZIONE. GLI ALTRI SISTEMI INTERAGISCONO CON IL WEB SERVICE NELLA MANIERA PRESCRITTA DA QUESTA DESCRIZIONE UTILIZZANDO MESSAGGI SOAP, TIPICAMENTE TRASFERITI USANDO HTTP CON UNA SERIALIZZAZIONE XML INSIEME CON ALTRI STANDARD COLLEGATI AL WEB.

QUINDI SOAP È UN ELEMENTO FONDAMENTALE DELLA TECNOLOGIA DEI WEB SERVICES. SOAP È UN PROTOCOLLO MOLTO SEMPLICE MA ESTREMAMENTE FLESSIBILE ED ESTENDIBILE E PER QUESTO MOTIVO PUÒ ESSERE ADATTATO A QUALSIASI ESIGENZA. POICHÉ È BASATO SU XML, SOAP È NEUTRALE RISPETTO AL LINGUAGGIO DI PROGRAMMAZIONE ED ALLA PIATTAFORMA HARDWARE/SOFTWARE.

FUNZIONALITÀ DI SOAP

SOAP È DIVENTATO UN PROTOCOLLO DI IMPORTANZA FONDAMENTALE PERCHÉ ATTUALMENTE È IL MIGLIOR SFORZO PRODOTTO DALL'INDUSTRIA PER STANDARDIZZARE LA TECNOLOGIA PER COSTRUIRE UN'INFRASTRUTTURA DI PROGRAMMAZIONE DISTRIBUITA BASATA SU XML.
SOAP È UN PROTOCOLLO SEMPLICE CHE DESCRIVE I PRINCIPALI ASPETTI DI UN MODELLO DI PROGRAMMAZIONE DISTRIBUITA.
IL PROTOCOLLO FORNISCE:
- UN MECCANISMO PER DEFINIRE IL FORMATO DELL'UNITÀ DI COMUNICAZIONE TRA APPLICAZIONI "LOOSELY-COUPLED".
- UN MODELLO DI ELABORAZIONE CHE FORNISCE L'INSIEME DI REGOLE CHE DEVE SEGUIRE IL MODULO SOFTWARE CHE DEVE GESTIRE I MESSAGGI SOAP.
- UN MECCANISMO PER LA GESTIONE DEGLI ERRORI CHE CONSENTA DI INDIVIDUARE LA CAUSA DELL'ERRORE E PERMETTA ALLE APPLICAZIONI DI SCAMBIARSI INFORMAZIONI PER LA DIAGNOSTICA DELL'ERRORE.
- UN MODELLO DI ESTENDIBILITÀ CHE PERMETTA DI AGGIUNGERE NUOVE FUNZIONALITÀ A QUELLE PREVISTE DAL PROTOCOLLO BASE
- UN MODELLO FLESSIBILE PER LA RAPPRESENTAZIONE DI DATI
- UNA CONVENZIONE PER RAPPRESENTARE INVOCAZIONI E RISPOSTE RPC TRAMITE MESSAGGI SOAP.

UN FRAMEWORK PER MAPPARE SOAP AD UN PROTOCOLLO DI TRASPORTO CHE SPECIFICA COME I MESSAGGI SOAP VENGONO INCAPSULATI NEI PACCHETTI DEL PROTOCOLLO DI TRASPORTO

STRUTTURA DELLA SPECIFICA E DIFFERENZE TRA LE VARIE VERSIONI

LA SPECIFICA DI SOAP 1.2 È DIVISA IN 3 PARTI:

- PARTE 0: È UN PRIMER, CIOÈ CON UN APPROCCIO NON FORMALE FORNISCE UN TUTORIAL FACILITANTE COMPRENSIBILE ALLE CARATTERISTICHE DELLA SPECIFICA E UTILIZZANDO DEGLI ESEMPI ANCHE DI DESCRIVENE DEGLI SCENARI DI UTILIZZO DELLE VARIE FUNZIONALITÀ.
- PARTE 1: CONTIENE LA DESCRIZIONE FORMALE DEL NUCLEO DELLA SPECIFICA E DESCRIVE LA STRUTTURA DEL MESSAGGIO SOAP, IL MODELLO DI ELABORAZIONE E QUELLO DI ESTENDIBILITÀ.
- PARTE 2: CONTIENE UNA SERIE DI AGGIUNTE AL NUCLEO DELLA PARTE 1, COME UN MODELLO DATI ED IL BINDING DEI MESSAGGI SOAP SUL PROTOCOLLO HTTP.

SOAP È FONDAMENTALMENTE UN PARADIGMA PER LO SCAMBIO DI MESSAGGI UNIDIREZIONALI E SENZA STATO TRA DUE NODI SOAP: IL SOAP SENDER ED IL SOAP RECEIVER. LE APPLICAZIONI, POI, POSSONO CREARE SCAMBI DI INTERAZIONE PIÙ COMPLESSI, PER ESEMPIO RICHIESTA-RISPOSTA O RICHIESTA-VARIE RISPOSTE, COMBINANDO VARI MESSAGGI. LA CREAZIONE DI QUESTI SCAMBI COMPLESSI È BASATA SUI MECCANISMI DI ESTENDIBILITÀ DI SOAP CHE CONSENTONO DI DEFINIRE DELLE CARATTERISTICHE SPECIFICHE PER UNA DATA APPLICAZIONE.

STRUTTURA DI UN MESSAGGIO SOAP

L'ELEMENTO RADICE DI UN MESSAGGIO SOAP È SOAPENV:ENVELOPE, NEL NAMESPACE "…/SOAP-ENVELOPE", TIPICAMENTE IDENTIFICATO DAL PREFISSO SOAPENV.
QUANDO IL NODO SOAP RICEVE IL MESSAGGIO, RICONOSCE IL NAMESPACE ED INTERPRETA NELLA MANIERA CORRETTA IL SUO CONTENUTO.
L'ELEMENTO SOAPENV:ENVELOPE CONTIENE DUE ELEMENTI: SOAPENV:BODY E SOAPENV:HEADER.
L'ELEMENTO HEADER È OPZIONALE, MA PUÒ ESSERE INSERITO PER INCLUDERE INFORMAZIONI CHE SPIEGANO DELLE CARATTERISTICHE AGGIUNTIVE DEL PROTOCOLLO. QUESTO ELEMENTO È IL MECCANISMO DI ESTENDIBILITÀ FORNITO DA SOAP PER PASSARE INFORMAZIONI CHE NON SONO CONTENUTE NEL MESSAGGIO. I FIGLI DELL'ELEMENTO HEADER SONO DETTI HEADER BLOCKS, OGNI HEADER BLOCK RAPPRESENTA UN RAGGRUPPAMENTO LOGICO DI DATI CHE È DIRETTO AD UN NODO SOAP E CHE IMPLEMENTA UNA DETERMINATA CARATTERISTICA DEL PROTOCOLLO.
L'ELEMENTO SOAPENV:BODY È OBBLIGATORIO E CONTIENE IL CONTENUTO DEL MESSAGGIO DIRETTO AL SOAP RECEIVER. L'ELEMENTO BODY PUÒ CONTENERE QUALSIASI SOTTOINSIEME XML.

GESTIONE DEGLI ERRORI

SOAP GESTISCE GLI ERRORI IN MANIERA SIMILE A COME FA JAVA, UTILIZZANDO IL CONCETTO DI SOAP FAULT CHE È SIMILE A QUELLO DELLE ECCEZIONI.
UN SOAP FAULT È UN NORMALE MESSAGGIO SOAP CHE CONTIENE ALL'INTERNO DEL BODY UN SOLO ELEMENTO DI NOME SOAP-ENV:FAULT.
IL CONTENUTO DELL'ELEMENTO FAULT È DEFINITO NEI SEGUENTI MODO:

- FAULT CODE: È UN CODICE, RAPPRESENTATO DA UN QNAME, CHE IDENTIFICA LA TIPOLOGIA DI ERRORE.
  SOAP DEFINISCE 4 TIPOLOGIE DI ERRORI:
  - SENDER: IDENTIFICA CHE IL PROBLEMA È STATO CAUSATO DA UNA SCORRETTA COSTRUZIONE DEL MESSAGGIO.
  - RECEIVER: IDENTIFICA CHE IL PROBLEMA SI È VERIFICATO DURANTE L'ELABORAZIONE DEL MESSAGGIO MA NON È ATTRIBUIBILE AL CONTENUTO DEL MESSAGGIO STESSO
  - MUSTUNDERSTAND: IDENTIFICA LA SITUAZIONE IN CUI UN MESSAGGIO CONTIENE UN HEADER BLOCK CON ATTRIBUTO DI MUSTUNDERSTAND=TRUE, MA IL NODO SOAP CHE HA RICEVUTO IL MESSAGGIO NON È IN GRADO DI COMPRENDERLO
  - VERSIONMISMATCH: QUESTO ERRORE VIENE GENERATO QUANDO IL NODO SOAP RICEVE UN MESSAGGIO SOAP CONFORME AD UNA VERSIONE CHE LUI NON È IN GRADO DI SUPPORTARE.
- SUBCODE: SOAP CONSENTE DI SPECIFICARE UNA GERARCHIA ARBITRARIA DI SOTTOCODICI DI ERRORE CHE POSSONO SERVIRE A FORNIRE MAGGIORI INFORMAZIONI SUL TIPO DI ERRORE CHE SI È VERIFICATO.
- REASON: QUESTO ELEMENTO È OBBLIGATORIO E CONTIENE UNA DESCRIZIONE TESTUALE DEL TIPO DI ERRORE CHE SI È VERIFICATO.
- NODE E ROLE: ELEMENTI FACOLTATIVI CHE SPECIFICANO IL NOME DEL NODO CHE HA GENERATO IL MESSAGGIO D'ERRORE ED IL SUO RUOLO.
- DETAILS: QUESTO ELEMENTO SERVE PER INSERIRE NEL MESSAGGIO FAULT INFORMAZIONI PIÙ COMPLESSE DEL SEMPLICE CODICE DI ERRORE O DELLA BREVE DESCRIZIONE TESTUALE CONTENUTI IN CODE E REASON.

LA STRUTTURA DI UN MESSAGE FAULT IN SOAP È LEGGERMENTE DIVERSA. LE PRINCIPALI DIFFERENZE SONO:

- IN SOAP NON EXISTE L'ELEMENTO SUBCODE MA IL VALORE DI CODE È UN QNAME FORMATO DA UNA CONCATENATA DI NOMI SEPARATI DA PUNTI.
- L'ELEMENTO REASON IN SOAP FAULT PUÒ CONTENERE UNA SOLA STRINGA.
- L'ELEMENTO DETAIL PUÒ CONTENERE SOLO INFORMAZIONI SUGLI ERRORI VERIFICATESI NELL'ELABORAZIONE DEL BODY DEL MESSAGGIO.

LA CONVENZIONE RPC DI SOAP

UNO DEGLI OBIETTIVI DEL PROGETTO SOAP ERA DI INCAPSULARE LE FUNZIONALITÀ DI RPC USANDO LA FLESSIBILITÀ E L'ESTENSIBILITÀ DI XML. LA CONVENZIONE RPC DI SOAP, DEFINISCE UNA RAPPRESENTAZIONE UNIFORME PER I MESSAGGI RICHIESTA DI RISPOSTA COINVOLTI IN UN'INTERAZIONE RPC. QUESTA RAPPRESENTAZIONE È BASATA SULLA DEFINIZIONE DELLA STRUTTURA DEL CONTENUTO DEL BODY QUANDO IL MESSAGGIO SOAP DEVE ESSERE UTILIZZATO PER IMPLEMENTARE UNA INTERAZIONE DI TIPO RPC.

PER INVOCARE UN SERVIZIO RPC TRAMITE SOAP, SONO NECESSARIE LE SEGUENTI INFORMAZIONI:

1) L'INDIRIZZO DEL NODO SOAP CHE FORNISCE IL SERVIZIO
2) IL NOME DEL METODO O DELLA PROCEDURA
3) I VALORI DEGLI ARGOMENTI CHE DEVONO ESSERE PASSATI ALLA PROCEDURA INSIEME CON EVENTUALI PARAMETRI OUTPUT ED IL VALORE DI RITORNO.
4) UNA CHIARA SEPARAZIONE DEGLI ARGOMENTI UTILIZZATI PER IDENTIFICARE LA RISORSA WEB, OBIETTIVO DELLA RICHIESTA RPC DA QUELLI CHE TRASPORTANO DATI O INFORMAZIONI DI CONTROLLO UTILIZZATI PER PROCESSARE LA CHIAMATA NEL NODO DESTINAZIONE.
5) LO SCHEMA DI SCAMBIO DI MESSAGGI UTILIZZATO PER IMPLEMENTARE RPC
6) EVENTUALI INFORMAZIONI ACCESSORIE CHE DEVONO ESSERE TRASPORTATE ALL'INTERNO DI HEADER BLOCKS

PER QUANTO RIGUARDA IL 1° PUNTO, SOAP IDENTIFICA IL NODO CHE DEVE PROCESSARE LA RICHIESTA RPC CON IL RUOLO DI ULTIMATE RECEIVER.

IL NODO ULTIMATE RECEIVER IDENTIFICA IL METODO O LA PROCEDURA DA INVOCARE ESAMINANDO LA SUA URI.

LE INFORMAZIONI RELATIVE AI PUNTI 2 & 3 SONO INSERITE NEL MESSAGGIO SOAP SECONDO UNA SCHEMA DEFINITO DALLA CONVENZIONE RPC DI SOAP.

IL CONTENUTO DEL BODY DI UN MESSAGGIO SOAP CONTENENTE UNA RICHIESTA RPC, DEVE SOTTOSTARE AI SEGUENTI VINCOLI:

- IL BODY CONTIENE UN UNICO FIGLIO IL CUI NOME COINCIDE CON IL NOME DEL METODO O DELLA PROCEDURA REMOTA CHE DEVE ESSERE INVOCATA.
- UN ELENCO DI [redacted] TANTI FIGLI QUANTI SONO GLI ARGOMENTI DA PASSARE AL METODO.

IL MESSAGGIO RPC DI RISPOSTA È COSTRUITO SECONDO I SEGUENTI VINCOLI:

- IL NOME È DATO DAL NOME DEL METODO INVOCATO SEGUITO DAL SUFFISSO RESPONSE.
- IL PRIMO FIGLIO SI CHIAMA RESULT E CONTIENE IL VALORE RESTITUITO DAL METODO REMOTO.
- GLI ALTRI FIGLI CONTENGONO I VALORI DI ALTRI PARAMETRI RESTITUITI DAL METODO.

PASSAGGIO DEI PARAMETRI

SOAP RPC DISTINGUE TRA PASSAGGIO DI PARAMETRI PER VALORE (IN CUI L'APPLICAZIONE INVOCANTE PASSA UN ARGOMENTO AL METODO INVOCATO), E PER RIFERIMENTO (DOVE L'APPLICAZIONE CHIAMANTE PASSA AL METODO L'INDIRIZZO DI UNA VARIABILE CHE IL METODO PROVVEDE A MODIFICARE).
RICORDIAMOCI CHE JAVA, NON SUPPORTA IL PASSAGGIO PER RIFERIMENTO.
SOAP DISTINGUE 3 TIPOLOGIE DI PARAMETRI:

- PARAMETRI IN: SONO PARAMETRI CHE VENGONO PASSATI DAL CLIENT AL [redacted] SERVER E COMPAIONO SOLO NEL MESSAGGIO SOAP DI RICHIESTA.
- PARAMETRI OUT: SONO PARAMETRI CHE VENGONO PASSATI AL SERVER AL CLIENT E COMPAIONO SOLO NEL MESSAGGIO DI RISPOSTA.
- PARAMETRI IN/OUT: SONO PARAMETRI CHE VENGONO PASSATI IN ENTRAMBI I VERSI. PRIMA IL CLIENT PASSA UN VALORE AL SERVER CHE LO MODIFICA E LO RESTITUISCE MODIFICATO. QUESTI PARAMETRI COMPAIONO SIA NEL MESSAGGIO DI RICHIESTA CHE IN QUELLO DI RISPOSTA.

PER DEFAULT I PARAMETRI SONO DI TIPO IN.

MODELLI DI ELABORAZIONE E DI ESTENSIONE DI SOAP

MODELLO DI ESTENSIONE DI SOAP

LA SPECIFICA SOAP HA PREVISTO DEI MECCANISMI ATTRAVERSO I QUALI È POSSIBILE ESTENDERE LE FUNZIONALITÀ DEL PROTOCOLLO PER ADEGUARLO ALLE ESIGENZE DI OGNI APPLICAZIONE.
DISTINGUIAMO 2 DIVERSI TIPI DI ESTENSIONE:

- ESTENSIONE VERTICALE: INTENDIAMO L'ESTENSIONE DELLE FUNZIONALITÀ DEL NODO SOAP CHE DEVE PROCESSARE IL MESSAGGIO (EX: FUNZIONALITÀ DI LOGGING O DI AUTENTICAZIONE).

- ESTENSIONE ORIZZONTALE: INTENDIAMO L'ESTENSIONE DELLE FUNZIONALITÀ DEI NODI SOAP CHE SI TROVANO LUNGO IL PERCORSO SEGUITO DAL MESSAGGIO PER CONSENTIRE AI SOAP RECEIVER I NODI LUNGO IL PERCORSO SONO DETTI INTERMEDIARI E POSSONO LIMITARSI A INSTRADARE PASSIVAMENTE I MESSAGGI O SVOLGERE UN RUOLO PIÙ ATTIVO PROCESSANDO I MESSAGGI. (EX: FUNZIONI DI FIRMA E AUTENTICAZIONE CHE I NODI POSSONO ESEGUIRE QUANDO IL MESSAGGIO ATTRAVERSA I CONFINI DI UN TRUST DOMAIN)

PER IMPLEMENTARE QUESTI MECCANISMI DI ESTENSIONE, CI SONO DI RISOLVERE 2 PROBLEMI:

- COME QUESTI ELEMENTI POSSONO ESSERE INSERITI ALL'INTERNO DI UN MESSAGGIO SOAP SENZA VIOLARE LA VALIDITÀ?
- COME QUESTI ELEMENTI DEVONO ESSERE PROCESSATI DA UN NODO SOAP?

I PROGETTISTI DI SOAP, HANNO RISOLTO IL 1° PROBLEMA DEFINENDO UN PUNTO BEN PRECISO DEL MESSAGGIO IN CUI È POSSIBILE INSERIRE GLI ELEMENTI DI ESTENSIBILITÀ.
PER IL 2° PROBLEMA, I PROGETTISTI DI SOAP HANNO SCELTO UN APPROCCIO CHE È STATO QUELLO DI DEFINIRE UN QUADRO DI RIFERIMENTO STANDARD [redacted] ALL'INTERNO DEL QUALE È POSSIBILE IMPLEMENTARE QUALSIASI TIPO DI ESTENSIONE IN UNA MANIERA STANDARD.
IN PARTICOLARE, L'ELEMENTO ALL'INTERNO DEL MESSAGGIO SOAP IN CUI VANNO INSERITE LE INFORMAZIONI PER ESTENDERE LE FUNZIONALITÀ DEL PROTOCOLLO È L'ELEMENTO OPZIONALE SOAP HEADER. L'HEADER È UN ELEMENTO XML CONTENUTO ALL'INTERNO DELL'ELEMENTO SOAPENV:ENVELOPE ED È IDENTIFICATO DAI TAG <SOAPENV:HEADER> </SOAPENV:HEADER>. IL CONTENUTO DELL'HEADER È ARBITRARIO. SECONDO LA SPECIFICA DI SOAP, OGNI ELEMENTO CONTENUTO IN HEADER È UN HEADER BLOCK ED IDENTIFICA UN ELEMENTO DI ESTENSIBILITÀ.

ESTENSIONE VERTICALE

L'UTILIZZO PRINCIPALE DEGLI HEADER È PER AGGIUNGERE NUOVE FUNZIONALITÀ AI MESSAGGI. NELLA MAGGIOR PARTE DEI CASI GLI HEADER VENGONO UTILIZZATI PER DUE SCOPI:

- ESTENSIONE DELL'INFRASTRUTTURA DI MESSAGING: GLI HEADER DI INFRASTRUTTURA POSSONO CONTENERE INFORMAZIONI QUALI CREDENZIALI DI SICUREZZA, DA UTILIZZARE PER IMPLEMENTARE UNO SCAMBIO DI MESSAGGI AFFIDABILE.

- DEFINIRE DATI ORTOGONALI: QUESTI HEADER VENGONO UTILIZZATI PER AGGIUNGERE CONTENUTI A UNO SCHEMA NON MODIFICABILE.
  QUESTI HEADER SONO DEFINITI DALL'APPLICAZIONE E CONTENGONO DATI CHE SONO ORTOGONALI AL CONTENUTO DEL BODY DEL MESSAGGIO, MA SONO DESTINATI ALL'APPLICAZIONE CHE DEVE PROCESSARE IL MESSAGGIO.

MODULI SOAP

QUANDO SI VUOLE IMPLEMENTARE UNA ESTENSIONE DEL PROTOCOLLO TRAMITE HEADER SOAP È NECESSARIO FORNIRE TUTTE LE ISTRUZIONI NECESSARIE AGLI ALTRI NODI PER POTER UTILIZZARE TALE ESTENSIONE. PER QUESTO MOTIVO, L'IMPLEMENTATORE SCRIVE UNA SPECIFICA CON I DETTAGLI RELATIVI ALLO SCOPO, IL FORMATO DEI DATI ECC... QUESTE SPECIFICHE SONO DETTE MODULI SOAP.

ESTENSIONE ORIZZONTALE

L'ESTENSIONE ORIZZONTALE RICHIEDE LA CAPACITÀ DI INVIARE DIVERSE PARTI DI UN MESSAGGIO SOAP A DIVERSI NODI.
IL PERCORSO SEGUITO DAL MESSAGGIO È DETTO SOAP MESSAGE PATH. IL PROTOCOLLO SOAP NON HA NESSUN MECCANISMO PER DEFINIRE IL PERCORSO CHE IL MESSAGGIO DEVE SEGUIRE.
LA SPECIFICA SOAP DISTINGUE DUE CATEGORIE DI INTERMEDIARI:

- GLI FORWARDING INTERMEDIARIES: SONO NODI CHE BASANDOSI SULLA SEMANTICA ASSOCIATA AD UN HEADER BLOCK DEL MESSAGGIO RICEVUTO, POSSONO INSTRADARE IL MESSAGGIO SOAP AD UN ALTRO NODO, EVENTUALMENTE AGGIUNTO O MODIFICATO LA STRUTTURA DEL MESSAGGIO.

- GLI ACTIVE INTERMEDIARIES: POSSONO FARE DELLE ELABORAZIONI SUL MESSAGGIO SOAP RICEVUTO PRIMA DI RISPEDIRLO AD UN ALTRO NODO, INDIPENDENTEMENTE DAL CONTENUTO DEL MESSAGGIO. TALI ELABORAZIONI, INFLUENZANO

L'INTERPRETAZIONE DEL MESSAGGIO SOAP DA PARTE DEI SUCCESSIVI NODI

LA DIFFERENZA CHIAVE TRA I DUE TIPI DI INTERMEDIARI, STA NELLE INFORMAZIONI CHE IL NODO SOAP SENDER HA SU DI LORO:

- LE OPERAZIONI ESEGUITE DAI FORWARDING INTERMEDIARIES SONO NOTE AL SENDER, CHE IN QUALCHE MODO LE RICHIEDE INSERENDO NEL MESSAGGIO DELL'HEADER BLOCK O ATTIVANDO UN PARTICOLARE TIPO DI INTERAZIONE
- LE OPERAZIONI ESEGUITE DAGLI ACTIVE INTERMEDIARIES NON SONO SOLLECITATE DAL SENDER

GLI ACTIVE INTERMEDIARIES POSSONO ESSERE:

- TRASPARENTI: IL SENDER NON HA CONOSCENZA DELLA PRESENZA DELL'INTERMEDIARIO
- ESPLICITI: IL SENDER SA CHE IL MESSAGGIO PASSERÀ PER L'INTERMEDIARIO, CHE POTRÀ INSERIRE QUINDI DELLE INFORMAZIONI DIRETTE ALL'INTERMEDIARIO

STRUTTURA DI HEADER BLOCK

SECONDO LA SPECIFICA SOAP, UN ELEMENTO HEADER APPARTIENE AL NAMESPACE "---/SOAP-ENVELOPE" E PUÒ CONTENERE UN NUMERO ARBITRARIO DI ELEMENTI FIGLI, DETTI HEADER BLOCK.

OGNI HEADER BLOCK DEVE SODDISFARE GLI SEGUENTI CARATTERISTICHE:

- DEVE AVERE UN NOME QUALIFICATO
- IL SUO CONTENUTO È ARBITRARIO
- PUÒ AVERE 0 O PIÙ ATTRIBUTI

GLI ATTRIBUTI CHE POSSONO ESSERE ASSEGNATI AD HEADER BLOCK SONO:

- SOAPENV: ENCODINGSTYLE
- SOAPENV: ROLE
- SOAPENV: MUSTUNDERSTAND
- SOAPENV: RELAY

L'ATTRIBUTO SOAPENV:ROLE → VIENE UTILIZZATO PER INDICARE IL NODO SOAP A CUI È DIRETTO L'HEADER BLOCK. IL VALORE DI ROLE PUÒ ESSERE SPECIFICATO DAL SERVIZIO. IN QUESTO MODO, SE UN NODO SOAP SUPPORTA UNA CERTA FUNZIONALITÀ ALLORA È IN GRADO DI RICONOSCERE IL RUOLO CORRISPONDENTE E QUINDI DI PROCESSARE IL CONTENUTO DELL'HEADER BLOCK.
SOAP PREVEDE IN AGGIUNTA 3 RUOLI CHE SONO RICONOSCIUTI DA TUTTI I NODI SOAP, E CHE SONO:

- "---/ROLE/NEXT": INDICA CHE L'HEADER BLOCK DEVE ESSERE PROCESSATO DAL PROSSIMO NODO SOAP LUNGO IL PERCORSO

- "----/role/ultimateReceiver" → SI RIFERISCE ALL'ULTIMO NODO SOAP LUNGO IL PERCORSO DEL MESSAGGIO CHE È QUELLO CHE DEVE PROCESSARE IL BODY DEL MESSAGGIO.

- "--/role/none" → È UN RUOLO SPECIALE CHE NESSUN NODO SOAP PUÒ ASSUMERE. UN HEADER BLOCK CON QUESTO RUOLO NON È DIRETTO A NESSUN NODO SOAP. QUESTO SIGNIFICA CHE NESSUN NODO PUÒ PROCESSARLO/RIMUOVERLO DAL MESSAGGIO. VIENE UTILIZZATO PER CONTENERE DATI CHE TUTTI I NODI SOAP LUNGO IL PERCORSO DEVONO LEGGERE.

IN SOAP, L'ATTRIBUTO CHE SERVE AD IDENTIFICARE IL DESTINATARIO DELL'HEADER BLOCK SI CHIAMA ACTOR.

L'ATTRIBUTO MUSTUNDERSTAND → VIENE UTILIZZATO PER INDICARE SE L'ELABORAZIONE DI UN HEADER BLOCK È OBBLIGATORIA O FACOLTATIVA.
VIENE UTILIZZATO ANCHE PER IDENTIFICARE GLI HEADER BLOCK CHE CONTENGONO INFORMAZIONI CRITICHE PER L'IMPLEMENTAZIONE DEL SERVIZIO. (ESEMPIO INFORMAZIONI UTILI A DECRIPTARE IL BODY DEL MESSAGGIO).

L'ATTRIBUTO RELAY → È UTILIZZATO PER INDICARE SE UN HEADER BLOCK DIRETTO AD UN CERTO NODO E CHE NON È STATO PROCESSATO, DEVE ESSERE TRASMESSO AL PROSSIMO NODO O RICEVUTO.

MODELLO DI ELABORAZIONE DI SOAP (Processing model)

IL MODELLO DI ELABORAZIONE DI SOAP SPECIFICA LE REGOLE CHE UN NODO SOAP DEVE SEGUIRE QUANDO RICEVE UN MESSAGGIO.

SECONDO LA SPECIFICA SOAP, L'ELABORAZIONE DI UN MESSAGGIO SOAP DEVE AVVENIRE SECONDO I SEGUENTI PASSI:

1) DETERMINARE L'INSIEME DEI RUOLI ASSUNTI DAL NODO
2) IDENTIFICARE TUTTI GLI HEADER BLOCK DIRETTI AI RUOLI DEL NODO E CHE SONO OBBLIGATORI, HANNO CIOÈ L'ATTRIBUTO CHE: MUSTUNDERSTAND = TRUE
3) SE IL NODO NON È IN GRADO DI "CAPIRE" QUALCUNO DEGLI HEADER BLOCK IDENTIFICATI AL PASSO PRECEDENTE, DEVE INTERROMPERE L'ELABORAZIONE E GENERARE UN SOAP FAULT DI TIPO MUSTUNDERSTAND
4) PROCESSARE TUTTI GLI HEADER BLOCK OBBLIGATORI DIRETTI AL NODO, E NEL CASO DI ULTIMARE ROLE=NEXT, ANCHE IL BODY.

GLI ERRORI CHE SI VERIFICANO DURANTE L'ELABORAZIONE DI UN HEADER BLOCK GENERANO UN UNICO MESSAGGIO SOAP FAULT. QUESTO MESSAGGIO, PUÒ CONTENERE AL SUO INTERNO DEGLI HEADER BLOCK. IN PARTICOLARE, CONTERRÀ DEGLI HEADER BLOCK RELATIVI AGLI ERRORI RELATIVI ALL'ELABORAZIONE DEGLI HEADER BLOCK.

SONO PREVISTI 2 HEADER BLOCK:

- NOTUNDERSTOOD HEADER: AGGIUNTO AD UN SOAP FAULT DI TIPO MUSTUNDERSTAND PER IDENTIFICARE IL NOME DELL'HEADER OBBLIGATORIO CHE NON È STATO COMPRESO
- UPGRADE HEADER: AGGIUNTO AD UN SOAP FAULT DI TIPO VERSIONMISMATCH PER INDICARE LA VERSIONE DI SOAP CHE È SUPPORTATA

ARCHITETTURA DI AXIS E DESCRIZIONE DEL PROCESSO DI ELABORAZIONE DEI MESSAGGI SOAP

AXIS È UN MOTORE SOAP E QUINDI SI OCCUPA DI PROCESSARE MESSAGGI SOAP.
AXIS È STATO PROGETTATO IN MANIERA MODULARE: OGNI COMPONENTE PUÒ ESSERE PROGETTATO IN MANIERA INDIPENDENTE DAGLI ALTRI E SVOLGE UNA PARTICOLARE ELABORAZIONE SUI MESSAGGI SOAP PROCESSATI DAL MOTORE.
L'ELABORAZIONE DI UN MESSAGGIO SOAP DA PARTE DI AXIS SI RIDUCE ALL'ESECUZIONE DI UNA SEQUENZA DI COMPONENTI, CIASCUNO DEI QUALI SVOLGE UNA FUNZIONE.
QUESTI COMPONENTI ATTI, HANDLER, POSSONO ESSERE FACILMENTE COMPOSTI IN MODO DA IMPLEMENTARE ELABORAZIONI COMPLESSE DA ADATTARSI ALLE ESIGENZE DELLE VARIE APPLICAZIONI.
OGNI HANDLER È UNA CLASSE JAVA CHE IMPLEMENTA L'INTERFACCIA HANDLER, CARATTERIZZATA DAL SOLO METODO: VOID INVOKE (MESSAGE CONTEXT)
QUANDO UN HANDLER VIENE INVOCATO ESEGUE LA LOGICA INCAPSULATA ALL'INTERNO DEL SUO METODO INVOKE E PUÒ SVOLGERE QUALSIASI TIPO DI ELABORAZIONE SU UN MESSAGGIO SOAP: LEGGERE/MODIFICARE PARTI DEL MESSAGGIO, MEMORIZZARE DELLE INFORMAZIONI IN UN FILE DI LOG O IN UN DATABASE, VERIFICARE DELLE CREDENZIALI DI AUTENTICAZIONE O QUALSIASI ALTRA COSA SI POSSA IMMAGINARE.
IL METODO INVOKE DI UN HANDLER PRENDE IN INPUT UN RIFERIMENTO AD UN OGGETTO MESSAGECONTEXT.
UN OGGETTO MESSAGECONTEXT VIENE PASSATO A CIASCUNO DEGLI HANDLER INVOCATI PER IMPLEMENTARE UN CERTO WEB SERVICE. CIASCUNO DEGLI HANDLER RICEVE IN INPUT IL MESSAGECONTEXT MODIFICATO DALL'HANDLER PRECEDENTE OPERA SU DI ESSO, EVENTUALMENTE MODIFICANDOLO E LO PASSA ALL'HANDLER SUCCESSIVO. QUINDI IL MESSAGECONTEXT RAPPRESENTA UNA SORTA DI MEMORIA CONDIVISA ATTRAVERSO IL QUALE I DIVERSI HANDLER POSSONO INTERAGIRE TRA DI LORO.
GLI HANDLER POSSONO ESSERE RACCOLTI IN CHAINS.
AXIS UTILIZZA 2 TIPI DI CHAINS:
- SIMPLE CHAINS: È UNA SEQUENZA DI HANDLER CHE DEVONO ESSERE INVOCATI NELL'ORDINE SPECIFICATO
- TARGETED CHAINS: È COSTITUITA DA 3 HANDLER, CHE DEVONO ESSERE INVOCATI IN SUCCESSIONE:
  - L'HANDLER DI RICHIESTA
  - IL PIVOT
  - L'HANDLER DI RISPOSTA

LE TARGETED CHAIN SONO ~~CREATE~~ COSTRUITE SULL'IDEA CHE IL LAVORO PRINCIPALE DEVE ESSERE SVOLTO DAL PIVOT, MENTRE GLI HANDLER DI RICHIESTA E DI RISPOSTA DEVONO ESEGUIRE OPERAZIONI DI PREPROCESSING E POSTPROCESSING.

UNIVERSITA' DEGLI STUDI DI SALERNO

Corso di Laurea di Informatica
Insegnamento di Programmazione su Reti

L.O. 5: Aspetti Avanzati di Web Services
U.D. 1: Modelli di elaborazione e di estensione di SOAP

Prof. Vincenzo Auletta
email: auletta@dia.unisa.it

5/5/2008

Modelli di elaborazione e di estensione di SOAP

Modello di estensione di SOAP

La specifica di SOAP ha definito un formato dei messaggi volutamente semplice e minimale. La specifica stessa, però, he previsto dei meccanismi attraverso i quali è possibile estendere le funzionalità del protocollo nelle maniere più disparate per adeguarlo alle esigenze di ogni applicazione.

Nel parlare di estensioni delle funzioni base del protocollo distinguiamo tra due diversi tipi di estensione:
Estensione verticale – con questo termine intendiamo l'estensione delle funzionalità del nodo SOAP che deve processare il messaggio. Tipicamente queste funzionalità sono ortogonali rispetto al servizio che si sta implementando ma fanno comunque parte del modello di elaborazione disegnato dal progettista. Esempi di estensioni di questo genere sono funzionalità di logging o di autenticazione che devono essere eseguite su tutte le interazioni con i servizi offerti da un'azienda, indipendentemente dalla loro logica.
Estensione orizzontale – con questo termine intendiamo l'estensione delle funzionalità dei nodi SOAP che si trovano lungo il percorso seguito dal messaggio per giungere al SOAP Receiver. I nodi lungo il percorso sono detti intermediari e possono limitarsi a istradare passivamente i messaggi o svolgere un ruolo più attivo processando i messaggi. Un esempio di estensione di qesto genere possono essere le funzionalità di firma e autenticazione che i nodi possono eseguire quando il messaggio attraversa i confini di un Trust Domain.

Per implementare questi meccanismi di estensione c'è bisogno di risolvere due problemi: come questi elementi possono essere inseriti all'interno di un messaggio SOAP senza violarne la validità e come questi elementi devono essere processati da un nodo SOAP.

Per consentire di aggiungere elementi alla struttura di un messaggio SOAP si potrebbe modificare lo schema XML che definisce la struttura dei messaggi SOAP in modo da consentire di aggiungere nuovi elementi in qualsiasi punto del messaggio. Anche se, in linea teorica, è possibile costruire uno schema di SOAP di questo tipo, questo approccio è da scartare perchè renderebbe il processo di validazione molto complesso. I progettisti di SOAP, invece, hanno risolto questo problema definendo un punto ben preciso del messaggio in cui è possibile inserire gli elementi di estendibilità.
Anche il problema di definire le regole di elaborazione degli elementi aggiunti al messaggio di SOAP è molto delicato. Infatti, le situazioni in cui si potrebbe dover processare questo tipo di informazioni sono le più disparate ed è praticamente impossibile trovare delle regole che vadano bene in ogni situazione. L'approccio dei progettisti di SOAP è stato quello di definire un quadro di riferimento standard all'interno del quale è possibile implementare qualsiasi tipo di estensione in una maniera standard.

In particolare, l'elemento all'interno del messaggio SOAP in cui vanno inserite le informazioni per estendere le funzionalità del protocollo è l'elemento opzionale SOAP Header. L'Header è un elemento XML contenuto all'interno dell'elemento soapenv:Envelope (prima dell'elemento

soapenv:Body) ed è identificato dai tag <soapenv:Header> </soapenv:Header>. Il contenuto dell'Header è arbitrario. Secondo la specifica SOAP 1.2 ogni elemento contenuto in Header è un Header block ed identifica un elemento di estendibilità ma molto spesso nel linguaggio corrente viene detto semplicemente header.

Vediamo un esempio di messaggio SOAP che contiene un ordine di acquisto per il Web Service della SkatesTown di sottomissione ordini. Le funzionalità del servizio sono estese con un controllo di autenticazione dell'utente che ha sottomesso l'ordine. Le informazioni relative all'autenticazione sono inserite all'interno di un header block di nome notary:token e verranno processate da una classe diversa da quella che processa l'ordine di acquisto (nella prossima unità didattica vedremo i dettagli del modello di elaborazione dei messaggi SOAP all'interno di AXIS Server). Questo consente di aggiungere delle informazioni ausiliarie necessarie al modello d'elaborazione del servizio senza dovere modificare la struttura del documento XML contenuto nel Body (in questo caso, l'ordine di acquisto).

```
<soapenv:Envelope
   xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
      <soapenv:Header>
         <notary:token xmlns:notary="http://notaries-r-us.com">
            XQ34Z-4G5
         </notary:token>
      </soapenv:Header>
      <soapenv:Body>
         <PO>
            <!-- inserire l'ordine di acquisto -->
         </PO>
      </soapenv:Body>
</soapenv:Envelope>
```

Ogni singolo header block rappresenta un elemento di estendibilità ed ogni messaggio SOAP può contenere un numero arbitrario di header block. I progettisti di SOAP hanno preso spunto da altri protocolli, come http, che sfruttano il concetto di header definiti dall'utente per estendere le funzionalità base del protocollo. Negli altri protocolli, però, il contenuto di questi header è semplicemente una stringa, mentre in SOAP ogni header block è basato su XML e quindi può essere utilizzato per codificare strutture dati molto più articolate e definire elaborazioni molto più flessibili e potenti.

Estensione Verticale

L'utilizzo principale degli header è per aggiungere nuove funzionalità ai messaggi. Questo tipo di utilizzo è quello che in precedenza abbiamo definito estensione verticale. Nonostante gli header possono contenere qualsiasi tipo di dati, nella maggior parte dei casi essi vengono utilizzati per due scopi:

- Estendere l'infrastruttura di messaging – questo tipo di header sono tipicamente processati dal middleware. L'applicazione non vede gli header ma può subire gli effetti della loro elaborazione. Gli header di infrastruttura possono contenere informazioni come credenziali di sicurezza, ID utilizzati per implementare uno scambio affidabile di messaggi, informazioni per controllare l'instradamento del messaggio o qualsiasi altro tipo di informazioni che possa fornire servizi all'applicazione.

- Definire dati ortogonali — Questi header sono definiti dall'applicazione e contengono dati che sono ortogonali al contenuto del body del messaggio ma sono comunque destinati all'applicazione che deve processare il messaggio. Per esempio, questi header potrebbero essere utilizzati per aggiungere elementi ad uno schema non modificabile. Invece, di definire una nuova versione del servizio e dover gestire l'interoperabilità tra le varie versioni, gli elementi aggiuntivi della nuova versione vengono inseriti nell'header e processati solo da quelle applicazioni che supportano questa nuova versione del servizio.

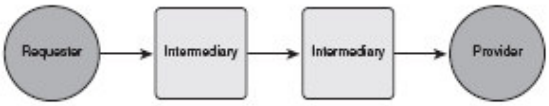
Le informazioni contenute negli header block possono essere anche dirette a nodi differenti che vengono incontrati lungo il percorso dal sender al receiver. Tali intermediari SOAP possono utilizzare i dati contenuti negli header block per fornire servizi a valore aggiunto, implementando quella che abbiamo definito estensione orizzontale. La scelta di quali dati inserire in un header block e quali inserire nel body deve essere presa al momento del progetto dell'applicazione.

Moduli SOAP

Quando si vuole implementare una estenzione del protocollo tramite header SOAP è necessario fornire tutte le istruzioni necessarie agli altri nodi per poter utilizzare tale estensione. Per questo motivo, l'implementatore scrive una specifica con i dettagli relativi alle regole, le precondizioni, il formato dei dati, la semantica dell'operazione, ecc. Queste specifiche sono dette moduli SOAP e sono identificate tramite delle URI. Quando un nodo SOAP vuole utilizzare una certa estensione fa riferimento all'URI del modulo SOAP corrispondente.

Estensione Orizzontale

Mentre l'estendibilità verticale riguarda la capacità di introdurre nuovi pezzi di informazione all'interno di un messaggio SOAP, l'estendibilità orizzontale riguarda la capacità di inviare diverse parti di un messaggio SOAP a diversi nodi. L'estendibilità orizzontale viene fornita tramite i nodi intermediari che sono applicazioni in grado di processare messaggi SOAP lungo il percorso seguito dal messaggio SOAP tra la sorgente e la destinazione finale. Il modello di SOAP prevede che non ci sia necessariamente comunicazione diretta tra il nodo SOAP Sender ed il nodo SOAP Ultimate Raceiver; il messaggio lungo la sua strada può passare per un numero arbitrario di nodi SOAP.



Il percorso seguito dal messaggio è detto SOAP message path. Il protocollo SOAP non ha nessun meccanismo per definire il percorso che il messaggio deve seguire.

La specifica di SOAP 1.2 distingue due categorie di intermediari: i forwarding intermediaries sono nodi che basandosi sula semantica associata ad un header block del messaggio ricevuto possono istradare il messaggio SOAP ad un altro nodo, eventualmente dopo aver processato alcuni header block ed, eventualmente aggiunto o modificato la struttura del messaggio; gli active intermediaries, invece, possono fare delle elaborazioni sul messaggio SOAP ricevuto prima di rispedirlo ad un altro nodo in maniera indipendente dal contenuto del messaggio stesso. Le elaborazioni eseguite da un intermediario attivo, in genere, influenzano il modo in cui i nodi successivi sul percorso interpreteranno il messaggio SOAP. La differenza chiave tra i due tipi di intermediari sta nelle informazioni che il nodo SOAP Sender ha su di loro: le operazioni eseguite dai forwarding intermediaries sono note al Sender che in qualche modo le richiede inserendo nel messaggio degli header block o attivando un particolare tipo di interazione; le operazioni eseguite dagli active

intermediaries non sono sollecitate dal Sender. Gli active intermediaries possono essere trasparenti o espliciti. Nel primo caso il Sender non ha cognizione della presenza dell'intermediario; nel secondo caso, invece, il Sender sa che il messaggio passerà per l'intermediario ed, in genere, inserisce informazioni dirette all'intermediario.

Il protocollo SOAP è stato progettato tenendo presente della presenza degli intermediari ed ha un meccanismo semplice ma flessibile che consente di passare informazioni ad un intermediario, specificando l'intermediario in questione e definendo come tale intermediario si deve comportare. Le informazioni dirette ad un intermediario vengono inserite all'interno di un header block e l'intermediario a cui sono dirette viene specificato tramite l'attributo role. Il modo in cui un nodo SOAP assume un certo ruolo esula dai compiti del protocollo SOAP e dipende dall'architettura del particolare motore SOAP utilizzato (nella prossima unità didattica vedremo come AXIS permette di specificare i ruoli). Il modo in cui l'intermediario si deve comportare, invece, è definito dal modello di elaborazione di SOAP.

Struttura di Header Block

Secondo la specifica SOAP 1.2 un elemento Header appartiene al namespace "http://www.w3.org/2003/05/soap-envelope" e può contenere un numero arbitrario di elementi figli, detti header block. Ogni header block deve soddisfare le seguenti caratteristiche:

- deve avere un nome qualificato;
- il suo contenuto è arbitrario e può contenere elementi figli sia qualificati che non qualificati;
- può avere zero o più attributi.

Tra gli attributi che possono essere assegnati ad header block la specifica di SOAP ne definisce quattro che hanno un significato particolare e sono compresi da qualsiasi nodo SOAP.

- soapenv:encodingStyle*;*
- soapenv:role*;*
- soapenv:mustUnderstand;
- soapenv:relay.

L'attributo soapenv:role

L'attributo soapenv:role viene utilizzato per indicare il nodo SOAP a cui è diretto l'header block. Il valore di questo attributo è un'URI e identifica una categoria di nodi SOAP e non necessariamente uno specifico nodo. Il valore di role può essere specificato dal servizio. In questo modo, se un nodo SOAP supporta una certa funzionalità allora è in grado di riconoscere il ruolo corrispondente e quindi di processare il contenuto dell'header block. SOAP 1.2 prevede in aggiunta ai ruoli definiti dall'applicazione anche tre ruoli che sono riconosciuti da tutti i nodi SOAP e che sono:

- **http://www.w3.org/2003/05/soap-envelope/role/next** — indica che l'header block deve essere processato dal prossimo nodo SOAP lungo il percorso. Questo ruolo viene utilizzato per elaborazioni che devono essere effettuate hop-by-hop (es. tracciatura del percorso seguito dal messaggio).
- **http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver** — si riferisce all'ultimo nodo SOAP lungo il percorso del messaggio che è quello che deve processare il body del messaggio. Per default, se non viene specificato nessun valore per l'attributo role, allora un header block è diretto a ultimateReceiver.

- **http://www.w3.org/2003/05/soap-envelope/role/none** — è un ruolo speciale che nessun nodo SOAP può assumere. Un header block con questo ruolo non è diretto a nessun nodo SOAP. Questo significa che nessun nodo può processarlo e rimuoverlo dal messaggio. In genere, viene utilizzato per contenere dati che tutti i nodi SOAP lungo il percorso devono leggere.

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
   <p:oneBlock xmlns:p="http://example.com"
        env:role="http://example.com/Log">
   ...
   </p:oneBlock>
   <q:anotherBlock xmlns:q="http://example.com"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   ...
   </q:anotherBlock>
   <r:aThirdBlock xmlns:r="http://example.com">
   ...
   </r:aThirdBlock>
  </env:Header>
  <env:Body >
   ...
  </env:Body>
 </env:Envelope>
```

Nell'esempio, il messaggio SOAP ha tre header blocks: il primo è diretto al prossimo nodo SOAP che riceverà il messaggio; il secondo è diretto ad un nodo SOAP che riveste il ruolo identificato dall'URI http://example.com/Log; il terzo non ha specificato nessun valore dell'attributo role e, quindi, per default è diretto all'Ultimate Receiver.

In SOAP 1.1 l'attributo che serve ad identificare il destinatario dell'header block si chiama actor e l'unico valore definito all'interno della specifica e riconosciuto da tutti i nodi SOAP è http://schemas.xmlsoap.org/soap/actor/next.

L'attributo mustUnderstand

L'attributo mustUnderstand viene utilizzato per indicare se l'elaborazione di un header block è obbligatoria o facoltativa. Il suo valore è Booleano e può essere "true" "1", "false" e "0". In SOAP 1.1, invece, gli unici valori ammissibili sono "0" e "1". Se questo attributo non viene specificato si intende che l'header block corrispondente è opzionale.

L'attributo mustUnderstand viene utilizzato per identificare gli header block che contengono informazioni critiche per limplementazione del servizio. Si immagini, per esempio, un messaggio SOAP il cui body per motivi di sicurezza è stato crittato perchè contiene informazioni private (esempio, il numero di carta di credito). L'applicazione aggiunge un header contenente le informazioni per descrivere come decrittare il messaggio. In questo caso, se il nodo SOPA che ha ricevuto il messaggio non è in grado di processare l'header e comprendere tali istruzioni non può processare correttamente il contenuto del messaggio SOAP.

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
   <p:oneBlock xmlns:p="http://example.com"
        env:role=”http://example.com/Log”
        env:mustUnderstand=”true”>
   ...
   </p:oneBlock>
   <q:anotherBlock xmlns:q="http://example.com"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   ...
   </q:anotherBlock>
   <r:aThirdBlock xmlns:r="http://example.com">
   ...
   </r:aThirdBlock>
  </env:Header>
  <env:Body >
   ...
  </env:Body>
 </env:Envelope>
```

Nell’esempio il primo header block è obbligatorio e quindi deve necessariamente essere processato da qualsiasi nodo SOAP che riveste il ruolo identificato dall’URI http://example.com/Log; gli altri due header block, invece, sono facoltativi.

L’attributo relay

L’attributo relay è utilizzabile solo con la versione SOAP 1.2 e viene utilizzato per indicare se un header block diretto ad un certo nodo e che non è stato processato deve essere trasmesso al prossimo nodo o rimosso. Il suo valore è Booleano e può essere “true” “1”, “false” e “0”. Se questo attributo non viene specificato si intende che l’header block corrispondente non deve essere ritrasmesso.

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
   <p:oneBlock xmlns:p="http://example.com"
        env:role=”http://example.com/Log”
        env:mustUnderstand=”true”>
   ...
   </p:oneBlock>
   <q:anotherBlock xmlns:q="http://example.com"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    env:relay=”true”>
   ...
   </q:anotherBlock>
   <r:aThirdBlock xmlns:r="http://example.com">
   ...
   </r:aThirdBlock>
  </env:Header>
  <env:Body >
```

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

Nell'esempio, il secondo header block è diretto al prossimo nodo SOAP sul percorso ma non è obbligatorio. Quindi, il nodo che lo riceve può decidere di processarlo, ed in questo caso cancellarlo dal messaggio SOAP che spedirà, oppure non processarlo e lasciarlo nel messaggio che spedirà.

E' evidente che ha senso specificare l'attributo env:relay solo nel caso di header block facoltativi e che non siano diretti al ruolo none.

Modello di elaborazione di SOAP (Processing Model)

SOAP definisce un modello di elaborazione distribuita che assume che un messaggio SOAP generato ad un nodo inziale (SOAP sender) è inviato ad un nodo destinazione (SOAP ultimate receiver) passando per zero o più nodi SOAP intermedi (SOAP intermediaries). Il modello di elaborazione di SOAP specifica le regole che un nodo SOAP deve seguire quando riceve un messaggio SOAP. Si noti che tali regole si riferiscono ad ogni singolo messaggio SOAP ricevuto. Il nodo SOAP processa ogni messaggio senza mantenere memoria dei messaggi che ha già visto passare. Eventuali informazioni di stato o correlazioni tra vari messaggi SOAP sono al di fuori degli scopi della specifica di SOAP e devono essere implementate attraverso estensioni del protocollo (SOAP features).

Secondo la specifica SOAP 1.2 l'elaborazione di un messaggio SOAP deve avvenire secondo i seguenti passi:

1. Determinare l'insieme dei ruoli rivestiti dal nodo. La specifica non prevede in che modo il nodo conosce quali sono i suoi ruoli. Può apprendere queste informazioni sia staticamente, hard-coded o lette da un file di configurazione, che dinamicamente, leggendo il contenuto del messaggio SOAP (sia body che header) prima dell'elaborazione e decidendo in funzione del suo contenuto. Questo significa che il nodo SOAP può cambiare i suoi ruoli tra l'elaborazione di un messaggio e quella del successivo; non può, invece, cambiare i suoi ruoli durante l'elaborazione di un messaggio.
2. Identificare tutti gli header block diretti ai ruoli del nodo e che sono obbligatori (mandatory), hanno cioè l'attributo env:mustUnderstand=true.
3. Se il nodo non è in grado di "capire" qualcuno degli header block identificati al passo precedente deve interrompere l'elaborazione e generare su SOAP Fault di tipo mustUnderstand. La definizione "capire" è da intendersi come la capacità del nodo SOAP di essere conforme ed implementare la semantica specificata dal modulo SOAP indicato dall'URI associata al nome dell'header block.
4. Processare tutti gli header block obbligatori diretti al nodo e, nel caso di ultimateReceiver, anche il body. Il nodo SOAP può scegliere di processare anche qualcuno degli header block diretti a lui e non obbligatori.
5. nel caso di un intermediario SOAP un nuovo messaggio SOAP deve essere generato. Il messaggio prodotto deve essere costruito secondo le seguenti regole: gli header block che sono stati processati devono essere rimossi; gli header block diretti al nodo che non sono stati processati e che hanno l'attributo env:relay=true debbono essere reinseriti nel messaggio SOAP uscente. Se, invece, il valore dell'attributo env:relay è false il nodo può decidere di comportarsi come vuole, anche se di norma decide di rimuoverli. Tutti gli header block non diretti al nodo non sono considerati.

In ogni caso un intermediario SOAP può sempre decidere di inserire nuovi header block nel messaggio uscente; in particolare, può reinserire qualcuno degli header block che ha processato e che ha dovuto rimuovere dal messaggio. La seguente tabella riassume le condizioni che definiscono se un header block deve essere reinserito nel messaggio uscente.

ruolo	assunto	compreso e processato	Inoltrato
next	sì	sì	Solo se reinserito
		no	Solo se relay ="true"
user-defined	sì	sì	Solo se reinserito
		no	Solo se relay ="true"
	no	n/a	Sì
ultimateReceiver	sì	sì	n/a
		no	n/a
None	no	n/a	Sì

E' bene precisare che un intermediario SOAP è comunque libero di leggere e modificare il contenuto di ogni parte del messaggio, anche quella che non appartiene a header block diretti ai suoi ruoli, senza con questo implicare che ha processato quell'elemento. Infatti, se un nodo decide di processare un header block lo deve fare in conformità alla semantica specificata dal modulo SOAP indicato dall'URI associata al nome dell'header block. La specifica determina le circostanze in cui una tale elaborazione dovrebbe produrre un fault.

Gli errori che si verificano durante l'elaborazione di un header block generano un unico messaggio SOAP Fault. Questo messaggio, come tutti gli altri messaggi SOAP, può contenere al suo interno degli header block. In particolare, conterrà degli header block relativi ad errori relativi all'elaborazione degli header block. Due header block previsti dalla specifica SOAP 1.2 sono:

- **NotUnderstood Header** – aggiunto ad un SOAP fault di tipo mustUnderstand per identificare il nome dell'header obbligatorio che non è stato compreso. L'header block contiene il nome qualificato dell'header block che ha generato l'errore. Eventualmente l'header può contenere anche il nome di un header block alternativo che il nodo è in grado di comprendere. Questo meccanismo consente di negoziare alcune caratteristiche di QoS.
- **Upgrade Header** – aggiunto ad un SOAP Fault di tipo VersionMismatch per indicare la versione di SOAP che è supportata.

E' importante sottolineare che un solo messaggio SOAP Fault può essere generato per ogni messaggio SOAP elaborato. Quindi, se il nodo SOAP ha individuato più errori deve decidere quale di questi segnalare. Normalmente, quest'informazione è definita dalla semantica associata agli header blocks.

Utilizzo dell'API Message per creare messaggi con header

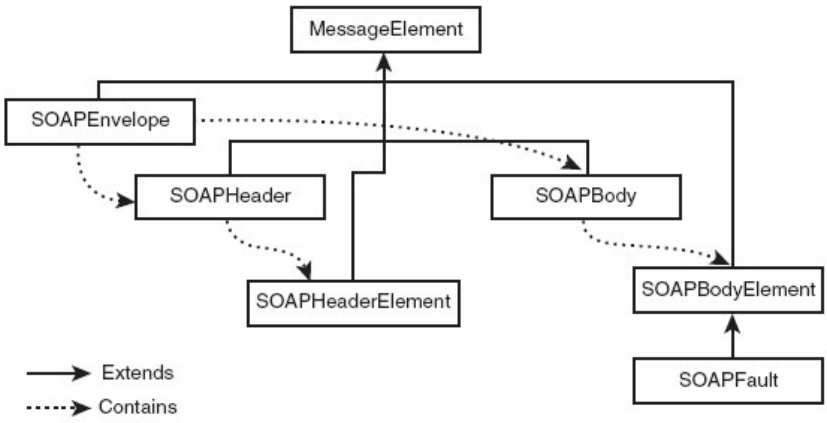
Fino ad ora abbiamo visto in astratto come un nodo SOAP deve elaborare messaggi SOAP. Vediamo ora come un'applicazione Java può rappresentare e manipolare messaggi SOAP. In particolare, faremo riferimento alle API utilizzate da AXIS per rappresentare i messaggi SOAP. Tali API sono conformi alla specifica SAAJ (SOAP with Attachments API for Java). SAAJ faceva parte di un progetto più ampio, denominato JAXM (Java API for XML Messaging). Quando i progettisti Java si sono resi conto che servivano delle classi Java per implementare i messaggi che

dovevano essere utilizzati in JAX-RPC, allora hanno scorporato la parte relativa alla rappresentazione dei messaggi e ne hanno fatto una specifica separata che adesso è arrivata alla versione 1.2. SAAJ permette ad un programmatore JAVA di produrre e manipolare messaggi SOAP 1.1.

Le interfacce di SAAJ sono contenute nel package javax.xml.soap. La classe base di tale specifica è la classe SOAPMessage, che è la classe radice per tutti i messaggi SOAP. Gli oggetti SOAPMessage vengono creati dalla classe MessageFactory. Un oggetto SOAPMessage può contenere un elemento SOAPPart, che contiene codice XML, ed un numero arbitrario di elementi AttachmentParts, che possono contenere qualsiasi tipo di dati. La codifica del messaggio sulla rete dipende dalla presenza di AttachmentParts: se ci sono Attachment l'oggetto SOAPMessage è codificato come un messaggio MIME, altrimenti è codificato come un semplice messaggio XML.

Gli AttachmentPart contengono dati definiti dall'applicazione e i corrispondenti MIMEHeaders, che sono coppie nome/valore (vedremo in maggiore dettaglio l'utilizzo degli AttachmentPart più avanti quando parleremo della trasmissione di dati binari all'interno di messaggi SOAP). L'oggetto SOAPPart, invece, contiene un oggetto SOAPEnvelope e consente di accedere alle varie parti del messaggio: SOAPBody e SOAPHeader. I contenuti di ciascuna delle due parti sono definiti come interfacce che estendono l'interfaccia comune MessageElement.

La figura mostra la relazione tra le varie classi Java che rappresentano i componenti di un messaggio SOAP.



Tutte queste classi implementano anche l'interfaccia Element di DOM e quindi è possibile vedere l'elemento SOAPEnvelope come un oggetto Document e navigarlo e manipolarlo usando le API di DOM. Non è possibile,però, passare indiscriminatamente dalla rappresentazione come SOAPMessage a quella come Document e viceversa. In particolare, dopo che il Document è stato (implicitamente) trasformato in un elemento di un SOAPMessage non è più possibile navigarlo usando le API di DOM.

L'architettura di SAAJ è abbastanza complessa e la creazione e l'elaborazione di messaggi SOAP richiede numerosi passi. Vediamo per esempio cosa si dovrebbe fare per creare un messaggio SOAP contenente una richiesta RPC:

1. istanziare la classe MessageFactory tramite il metodo static MessageFactory.newInstance();
2. istanziare un oggetto SOAPMessage tramite il metodo createMessage();
3. accedere all'oggetto SOAPPart tramite il metodo getSOAPPart();
4. accedere all'oggettto SOAPEnvelope tramite il metodo getEnvelope();

5. accedere all'oggetto SOAPBody tramite il metodo getSOAPBody();
6. creare un oggetto SOAPBodyElement ed aggiungerlo al SOAPBody tramite il metodo addBodyElement();
7. aggiungere i contenuti del SOAPBodyElement.

Per snellire l'interazione con i messaggi SOAP, i progettisti di AXIS hanno fornito una implementazione di SAAJ che, pur supportando completamente la specifica 1.2, estende quest'ultima in maniera significativa e rende l'elaborazione dei messagi molto più semplice. L'implementazione fornita da AXIS di SAAJ è basata sulla classe org.apache.axis.Message ed è contenuta nel package org.apache.axis.message. E' bene ricordare, però, che questa implementazione non è standard e può essere utilizzata solo su nodi SOAP basati su AXIS.

La novità principale dell'implementazione di SAAJ fornita da AXIS rispetto alla specifica originale è che ciascuna classe rappresentante un elemento del messaggio SOAP ha un costruttore e può essere creata autonomamente. Per esempio è possibile creare direttamente un SOAPBody senza dover creare un SOAPMessage e poi da questo recuperare il SOAPPart, il SOAPEnvelope ed il SOAPBody. Inoltre, nel package sono contenute anche delle nuove classi che estendono l'interfaccia MessageElement e permettono di rappresentare delle chiamate RPC (RPCElement e RPCParam).

Di seguito elenchiamo le principali classi del package org.apache.axis.message, fornendo per ciascuna di esse una piccola descrizione.

- MessageElement — Tutte le classi Java che rappresentano elementi del messaggio SOAP sono derivate dalla classe org.apache.axis.message.MessageElement che implementa l'interfaccia SOAPElement di SAAJ ed aggiunge una serie di altri metodi di utilità che però non sono standard.
- SOAPEnvelope — questa classe rappresenta l'elemento SOAP envelope e contiene al suo interno un elemento SOAPBody ed eventualmente un elemento SOAPHeader. L'implementazione di AXIS di SOAPEnvelope consente di aggiungere direttamente all'Envelope elementi senza dover prima ottenere un riferimento all'elemento SOAPBody.
- SOAPBody — questa classe rappresenta l'elemento Body del messaggio SOAP e funge da contenitore per elementi SOAPBodyElement.
- SOAPBodyElement — Questi elementi rappresentano gli elementi contenuti nel Body e possono essere chiamate RPC, SOAP Fault o qualsiasi altra cosa possa essere contenuta nel Body.
- SOAPHeader — questa classe rappresenta l'elemento Header del messaggio SOAP e funge da contenitore per elementi SOAPHeaderElement.
- SOAPHeaderElement — Questi elementi rappresenta un Header block. L'interfaccia SOAPHeaderElement fornisce I metodi necessary per leggere e scrivere i vari attributi di un header block. Inoltre, ogni header block ha un flag che può essere settato per specificare se l'header block è stato processato o meno.
- SOAPFault — Questa classe estende SOAPBodyElement e rappresenta un SOAP fault.
- RPCElement – Questa classe non fa parte della specifica SAAJ ed è specifica delle API di AXIS. L'elemento rappresenta una chianata RPC costruita secondo la convenzione RPC di SOAP. La classe fornisce dei metodi per ottenere e modificare il nome dell'operazione da invocare e i parametri da passare.
- RPCParam – Questa classe non fa parte della specifica SAAJ ed è specifica delle API di AXIS. L'elemento rappresenta un parametro di una chianata RPC.

A titolo esemplificativo, vediamo cosa si dovrebbe fare per creare un messaggio SOAP contenente una richiesta RPC usando il package org.apache.axis.message:

1. istanziare un oggettto SOAPEnvelope;
2. istanziare un oggetto RPCElement, passando al costruttore della classe il contenuto dell'elemento;
3. aggiungere l'oggetto RPCElement al SOAPEnvelope usando il metodo addBodyElement().

Per interagire con Web Services che utilizzano elementi di estendibilità è necessario utilizzare lato client le API di SAAJ per creare un elemento SOAPEnvelope e poi passarlo al metodo invoke(SOAPEnvelope) dell'oggetto Call. In questo caso, però, il metodo invoke non restituirà direttamente il valore di ritorno dell'operazione RPC ma un oggetto SOAPEnvelope che rappresenta il messaggio di risposta di RPC e bisogna nuovamente utilizzare le API di SAAJ per accedere ai valori contenuti al suo interno.

Esercitazione

Proviamo ora a scrivere un'applicazione Java che utilizza le API di AXIS per manipolare messaggi SOAP. In particolare, riscriveremo l'applicazione client che interagisce con il Web Service InventoryCheck implementato nella unità didattica LO4.2 in modo che il messaggio SOAP spedito contenga due header block: il primo, chiamato "logHeader", è diretto ad ultimateReceiver e contiene l'indirizzo del client che deve essere utilizzato dal servizio per memorizzarlo in un file di log; il secondo, chaiamato "emailHeader", contiene un indirizzo di posta elettronica a cui deve essere inviata una risposta sulla disponibilità della merce.

```
import java.util.Iterator;
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import org.apache.axis.soap.SOAPConstants;
import org.apache.axis.message.SOAPEnvelope;
import org.apache.axis.message.SOAPHeaderElement;
import org.apache.axis.message.RPCElement;
import org.apache.axis.message.RPCParam;

/*
 * Inventory check web service client
 */
public class InventoryCheckerClient {
        /**
         * URL del servizio
         */
        String url;
        /**
         * Indirizzo email a cui mandare la conferma
         */
        String email;
        /**
         * Inizializza l'URL del servizio a cui puntare e l'indirizzo di posta elettronica
         * a cui ricevere la conferma.
         */
        public InventoryCheckerClient(String url, String email) {
```

```
                this.url = url;
                this.email = email;
        }

        /**
         * invoca il servizio di verifica della disponibilità
         */
        public boolean doCheck(String sku, int quantity) throws Exception {

                // costruisce l'header Email
                SOAPHeaderElement emailHeader = new
                        SOAPHeaderElement("http://www.skatestown.com", "Email");
                emailHeader.setRole("http://www.w3.org/2003/05/soap-
envelope/role/ultimateReceiver");
                //emailHeader.setMustUnderstand(true);
                emailHeader.setValue(email);

                //costruisce l'header Log
                SOAPHeaderElement logHeader = new
                        SOAPHeaderElement("http://www.skatestown.com", "Log");
                logHeader.setRole("http://skatestown.com/roles/logger");
                logHeader.setValue(email);

                // costruisce il messaggio SOAP con la richiesta RPC
                SOAPEnvelope reqEnv = new
                        SOAPEnvelope(SOAPConstants.SOAP12_CONSTANTS);
                reqEnv.addHeader(emailHeader);
                reqEnv.addHeader(logHeader);
                Object [] params = new Object [] { sku, new Integer(quantity), };
                reqEnv.addBodyElement(new RPCElement("", "doCheck", params));

                Service service = new Service();
                Call call = (Call) service.createCall();
                call.setTargetEndpointAddress( new java.net.URL(url));
                call.setSOAPVersion(SOAPConstants.SOAP12_CONSTANTS);
                // effettua la chiamata
                SOAPEnvelope respEnv = call.invoke (reqEnv);

                // recupera il risultato dal messaggio SOAP di risposta
                RPCElement respRPC = (RPCElement)respEnv.getFirstBody();
                RPCParam result = (RPCParam)respRPC.getParams().get(0);
                return ((Boolean)result.getObjectValue()).booleanValue();
        }
}
```

Architettura di AXIS e descrizione del processo di elaborazione dei messaggi SOAP

Nelle sezioni precedenti abbiamo visto come sia possibile costruire messaggi SOAP utilizzando le API di AXIS Client. Adesso vedremo come i messaggi SOAP possono essere letti e processati da

AXIS. Prima di fare ciò, però, dobbiamo esaminare l'architettura di AXIS e discutere il modello di elaborazione dei messaggi adottato da questo motore SOAP.

AXIS è un motore SOAP e quindi si occupa semplicemente di processare messaggi SOAP. Il motore SOAP può essere utilizzato in diversi ambienti e nei contesti più svariati implementando modelli di elaborazione non prevedibili al momento della sua progettazione. Per questo motivo, AXIS è stato progettato in maniera estremamente modulare: ogni componente può essere progettato in maniera indipendente dagli altri e svolge una particolare elaborazione sui messaggi SOAP processati dal motore. L'elaborazione di un messaggio SOAP da parte di AXIS si riduce all'esecuzione di una sequenza di componenti, ciascuno dei quali svolge una data funzione. Questi componenti, detti handler, possono essere facilmente composti in modo da implementare elaborazioni complesse ed adattarsi alle esigenze delle varie applicazioni.

Ogni handler è una classe Java che implementa l'interfaccia org.apache.axis.Handler, costituita dal solo metodo

void invoke(MessageContext context) throws AxisFault.

Quando un handler viene invocato esegue la logica incapsulata all'interno del suo metodo invoke e può svolgere qualsiasi tipo di elaborazione su un messaggio SOAP: leggere o modificare parti del messaggio, memorizzare delle informazioni in un file di log o in un database, verificare delle credenziali di autenticazione o qualsiasi altra cosa si possa immaginare. Axis fornisce una serie di handler precostituiti che si occupano della gestione delle sessioni e delle autorizzazioni ma gli sviluppatori possono facilmente implementare altri handlers e aggiungerli all'interno del motore SOAP.

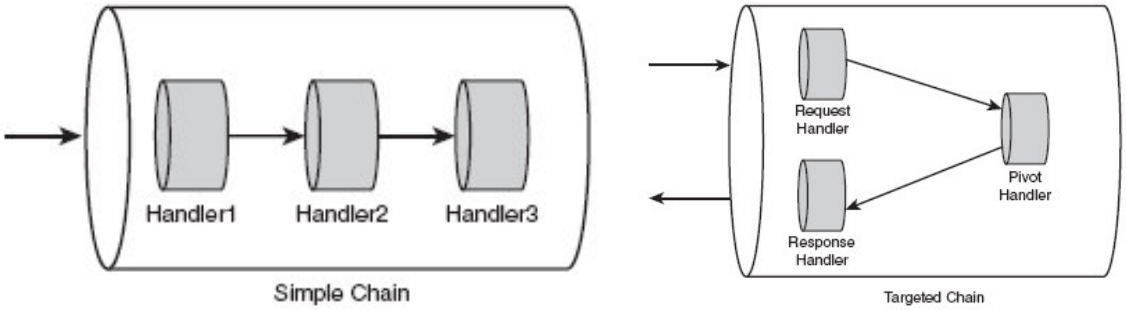
Il metodo invoke di un handler prende in input un riferimento ad un oggetto MessageContext. Questa classe riveste un ruolo cruciale nell'architettura di AXIS perché contiene tutte le informazioni che possono essere rilevanti per implementare un'interazione SOAP:

- messaggi SOAP di richiesta e di risposta;
- una serie di altre proprietà che permettono di controllare il comportamento del sistema;
- alcuni campi speciali.

Un oggetto MessageContext viene passato a ciascuno degli handler invocati per implementare un certo Web Service. Ciascun handler riceve in input il MessageContext modificato dall'handler precedente opera su di esso, eventualmente modificandolo e lo passa all'handler successivo. Quindi, il MessageContext rappresenta una sorta di memoria condivisa attraverso il quale i diversi handler possono interagire tra di loro.

Gli handler possono essere raggruppati in chains, che a loro volta implementano l'interfaccia Handler. Il sistema vede una chain come se fosse un handler ed invoca il suo metodo invoke; l'implementazione di questo metodo invoca in successione il metodo invoke di ciascun handler della chain (che potrebbe a sua volta essere una chain), passando ad ognuno di essi l'oggettoMessageContext restituito dall'handler precedente.

Axis utilizza due tipi di chains: simple chains e targeted chains.
Una simple chain è una sequenza di handler che devono essere invocati nell'ordine specificato. Una targeted chain, invece, è costituita da tre handler (che possono essere delle chain) che devono essere invocate in successione: l'handler di richiesta, il pivot e l'handler di risposta.



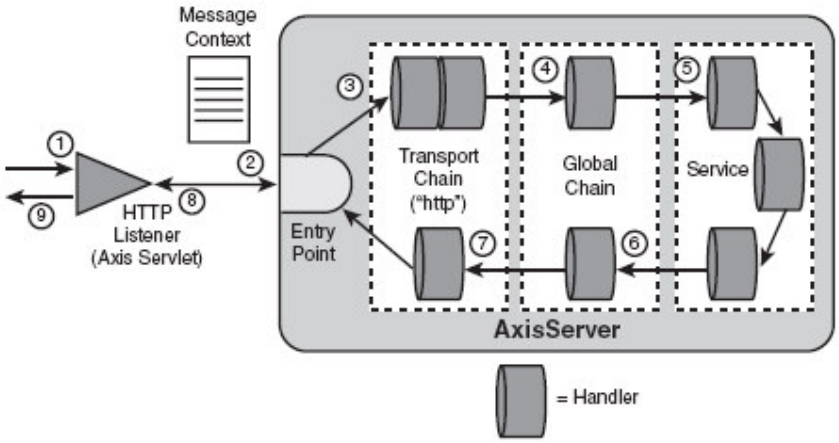
Le targeted chain sono costruite con l'idea che il lavoro principale deve essere svolto dal pivot mentre gli handler di richiesta e di risposta devono eseguire operazioni di preprocessing e postprocessing. I Web Services pubblicati su AXIS sono implementati come delle targeted chain ed il pivot handler chiama la classe Java che implementa il servizio. Questo significa che, lato server, l'handler di richiesta opera sul messaggio SOAP di richiesta prima che il servizio venga eseguito e l'handler di risposta opera sul messaggio SOAP di risposta creato dal pivot, eventualmente leggendo i valori contenuti nel messaggio SOAP di richiesta.

Elaborazione dei messaggi lato server

Esaminiamo ora come AXIS, quando opera come SOAP server, processa i messaggi SOAP. Gli handler che devono processare i messaggi SOAP che implementano un'interazione sono organizzati in chain di tre diversi livelli:

- transport chain;
- global chain;
- service chain.

La seguente figura mostra il flusso di elaborazione di un messaggio ricevuto tramite il protocollo http.



Quando un messaggio SOAP viene ricevuto dal servlet container che ospita AXIS le seguenti operazioni vengono eseguite:

1. il pacchetto http viene passato ad un Transport Listener. Con questo termine si intende un modulo software che è in grado di prendere pacchetti in input e tradurli in un formato

comprensibile per AXIS. HTTPListener fa parte della distribuzione di AXIS ed è implementato come un servlet. Esso riceve un pacchetto http e crea un oggetto MessageContext che contiene un oggetto Message che rappresenta il messaggio SOAP di richiesta, contenente le informazioni presenti nel pacchetto http, e varie altre proprietà utili per l'elaborazione del messaggio, come per esempio il nome del protocollo di trasporto utilizzato dal messaggio. Tra queste proprietà ci sono anche informazioni specifiche del protocollo di trasporto, come per esempio il valore di vatti campi dell'header http, in un formato comprensibile alla servlet.

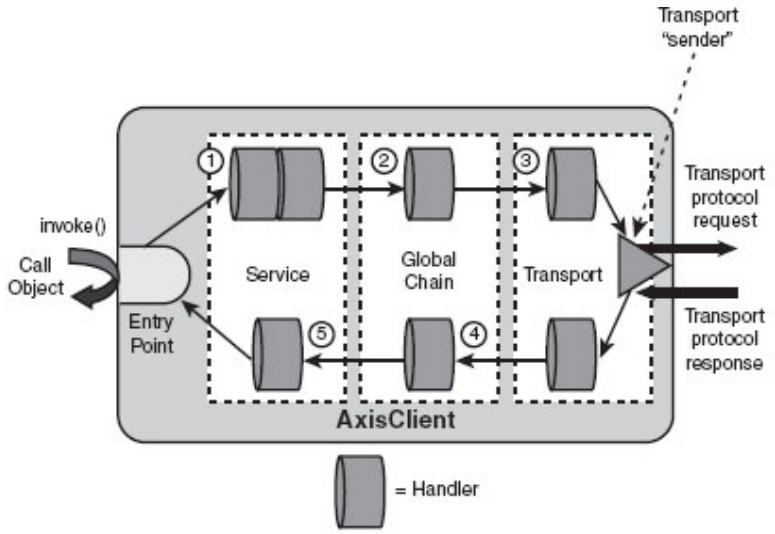
2. Il MessageContext creato dal Listener viene passato ad AXIS Server.
3. La prima cosa che AXISServer fa quando riceve il MessageContext e controllare se esiste una Transport chain (targeted chain) con lo stesso nome specificato nella proprietà del MessageContext contenente il nome del protocollo di trasporto. Se esiste, invoca il metodo invoke dell'handler di richiesta e gli passa il MessageContext prodotto dal Listener. Questo consente di implementare delle elaborazioni che dipendono dalle informazioni contenute nell'header del protocollo di traposrto e sono specifiche di quel tipo di trasporto. Per esempio, un handler potrebbe leggere il valore contenuto nel campo SOAPAction del pacchetto http (che specifica il nome del servizio da invocare) e renderlo disponibile agli handler successivi. Un altro esempio di handler di livello Transport è un handler che controlla le credenziali di autenticazione rappresentate nella maniera specificata dal protocollo di trasporto.
4. Il MessageContext prodotto dall'handler di richiesta della Transport chain viene passato all'handler di richiesta della Global chain. Questa chain contiene handler che devono processare ogni messaggio ricevuto dal sistema, indipendentemente dal protocollo di trasporto utilizzato. Per esempio, nella Global chain vengono inseriti gli handler che implementano le politiche di sicurezza o quelli che implementano sistemi di logging.
5. Dopo che tutti gli handler della global chain di richiesta sono stati invocati AXIS server deve invocare un Service handler che deve svolgere operazioni specifiche del servizio. Ci sono vari modi in cui il motore SOAP può identificare la Service chain da invocare per processare il messaggio. Una possibilità è che il nome della Service chain sia contenuta in uno dei campi del MessageContext. Il Service handler è un tipo particolare di handler, chiamato SOAPService (org.apache.axis.handlers.soap.SOAPService), che è esso stesso una targeted chain con al suo interno una chain di richiesta che esegue operazioni di preprocessing ed una di risposta che esegue operazioni di postprocessing. All'inerno di SOAPService vi è uno speciale handler, detto Provider, che funge da pivot della targeted chain e separa il flusso di richiesta da quello di risposta. Il Provider ha il compito di svolgere il vero lavoro del Web Service, inclusa la istanziazione della classe Java che implement ail servizio e l'invocazione dei suoi metodi. In particolare, è compito del Provider fare il parsing del messaggio SOAP per riconoscere il nome del metodo da invocare e gli argomenti da passare e, successivamente, prendere il valore restituito dalla classe Java e serializzare il messaggio SOAP di risposta. Il MessageContext prodotto dal SOAPService, eventualmente contenente il messaggio SOAP di risposta creato dal Provider e modificato dagli handler della response chain, viene passato alla Global response chain.
6. Gli handler della Global chain di risposta eseguono operazioni su tutti i messaggi di risposta spediti dal sistema (es. jogging). Alla fine il MessageContext prodotto viene passato alla Transport chain di risposta.
7. Gli handler della Transport chain di risposta di nuovo elaborano il messaggio di risposta in modalità specifiche del protocollo di trasporto. Per esempio, vanno a settare delle proprietà contenenti valori che devono essere inseriti nell'header dei pacchetti del protocollo di trasporto.
8. Il MessageContext finale prodotto dalla Transport chain di risposta viene restituito ad AXIS Server che lo invia all'HTTPListener.

9. HTTPlistener, sulla base delle informazioni contenute nel MessageContext, costruisce i pacchetti htto che devono essere rispediti al client.

Elaborazione dei messaggi lato client

La classe AxisClient è l'equivalente di AxisServer, lato client. La sua architettura modulare è simile a quella di AxisServer ma differisce per l'ordine con cui vengono invocate le chain. Le chain sono invocate nell'ordine:

- service chain;
- global chain;
- transport chain.



La classe Call svolge il ruolo di Listener e, sulla base delle informazioni ottenute dall'applicazione costruisce il MessageContext e lo passa ad AxisClient. Quest'operazione viene effettuata quando l'applicazione invoca il metodo invoke della Call. L'applicazione non interagisce mai direttamente con AxisClient o con i suoi handler.

La Transport chain lato client è un po' diversa da quella utilizzata sul server. In questo caso, infatti, c'è un handler pivot che è il transport sender, e d è quello che si occupa di prendere il messaggio SOAP di richiesta dal MessageContext e spedirlo secondo le modalità specifiche del protocollo di trasporto e di recuperare il messaggio di risposta inserirlo nel MessageContext ed inviarlo sulla catena di risposta fino a farlo arrivare all'oggetto Call. La Call è responsabile di fare il parsine del messaggio di risposta ed estrarre i dati da passare all'applicazione.

Utilizzi del MessageContext

Abbiamo visto che gli oggetti MessageContext svolgono un ruolo fondamentale nell'architettura di AXIS. Essi, inoltre, consentono di definire un'architettura "loosely coupled" per i componenti del processo di elaborazione dei messaggi che sia al tempo stesso flessibile e potente. Un esempio di ciò è l'implementazione di un modello di elaborazione che prevede l'interazione tra due handler: il primo handler deve passare dele informazioni che influenzano le operazioni del secondo handler. Invece di prevedere un meccanismo di comunicazione diretta tra i due handler, che a quel punto

sarebbero tightly coupled, AXIS utilizza il MessageContext come una sorta di memoria condivisa: il primo handler setta delle proprietà del MessageContext che verranno lette dal secondo handler.

Definizione della catena di elaborazione dei messaggi in AXIS

Dopo aver essaminato in dettaglio l'arcitettura di AXIS, ora dobbiamo vedere come specificare al motore AXIS quail handler devono essere utilizzati per processare un messaggio ed in che ordine devono essere invocati. Lo strumento principale utilizzato dallo sviluppatore del servizio per configurare Axis Server è il file WSDD utilizzato come descrittore di deploy di cui abbiamo già visto degli esempi di utilizzo nella LO4.

WSDD è uno schema XML che AXIS utilizza per memorizzare informazioni di configurazione e di deployment. Axis Server mantiene queste informazioni all'interno del file server-config.wsdd contenuto nella directory WEB-INF, mentre AxisClient mantiene le informazioni di configurazione nel file client-config.wsdd presente nella direcotry in cui è eseguita l'applicazione che utilizza il servizio. Entrambi i file hanno una versione di default che fa parte di axis.jar e che viene utilizzata in assenza di una versione presente nella directory di axis.

L'elemento radice di un documento WSDD e l'elemento <deployment>. Tra i suoi figli c'è un elemento <globalConfiguration> che contiene delle opzioni per il motore AXIS (sia client che server) e le definizioni delle global chain di richiesta e di risposta.

```
<globalConfiguration>
        <parameter name="defaultSOAPVersion" value="1.2"/>
        <requestFlow>
                <handler type="java:org.apache.axis.handlers.LogHandler"/>
        </requestFlow>
        <responseFlow>
        </responseFlow>
</globalConfiguration>
```

Questo esempio mostra una <globalConfiguration> che specifica una global chain di risposta vuota ed una di richiesta contenente un handler che effettua un servizio di logging. La dichiarazione <parameter> serve a specificare il valore dell'opzione defaultSOAP che, in questo caso, è SOAP 1.2. Per default AXIS utilizza SOAP 1.1.Per passare a SOAP 1.2 bisogna specificare l'opzione defaultSOAPVersion nell'elemento <globalConfiguration>. Lato client, invece, è possible specificare dinamicamente la versione di SOAP usando il metodo setSOAPConstants della classe Call.

WSDD consente anche di dichiarare degli handler e configurarli con delle opzioni. Le dichiarazioni di handler possono apparire come figli di <deployment> oppure all'interno di elementi <chain>, <requestFlow> e <responseFlow>. Il seguente è un esempio di dichiarazione di handler.

```
<handler [name="name"] type="type">
        <parameter name="name" value="value"/>
</handler>
```

L'attributo type specifica il tipo di handler. Il suo valore è o un QName in uno speciale namespace Java (xmlns:java="http://xml.apache.org/axis/wsdd/providers/java") che specifica il nome della classe Java che implementa l'handler oppure il nome di un handler/chain precedentemente definita. In questo caso, nella precedente dichiarazione si deve essere specificato l'attributo name. Gli

elementi <parameter> vengono utilizzati per settare delle opzioni. Il seguente esempio mostra la definizione di un Handler che effettua un servizio di jogging che utilizza l'opzione logFile per specificare il nome del file di log.

```
<handler name="log" type="java:org.apache.axis.handlers.LogHandler">
        <parameter name="logFile" value="myLog.txt"/>
</handler>
```

L'insieme di opzioni supportate da un handler dovrebbe essere specificato nella documentazione dell'handler. L'interfaccia Handler fornisce i metodi getOption() and setOption() che permettono di accedere e modificare i valori di queste opzioni. L'attributo locked="true" conente di specificare che il valore di una certa opzione non deve essere modificato al runtime.

La dichiarazione di una chain specific ail nome della chain e l'elenco degli handler che la compongono. Gli handler verranno invocati nell'ordine in cui compaiono all'interno della dichiarazione della chain. Per esempio, la seguente dichiarazione specifica la chain "logAndNotify" costituita dai due handler LogHandler e NotificationHandler che devono essere invocati in quest'ordine.

```
<chain name="logAndNotify">
        <handler type="java:org.apache.axis.handlers.LogHandler"/>
        <handler type="java:myPackage.NotificationHandler">
                <parameter name="email" value="admin@skatestown.com"/>
        </handler>
</chain>
```

Una volta definite le chain, queste possono essere inserite all'interno degli elementi <requestFlow> e <responseFlow>.

Infine, l'elemento <transport> definisce la Transport chain. Il seguente è l'elemento <transport> contenuto nella versione di default di server-config.wsdd.

```
<handler type="java:org.apache.axis.handlers.http.URLMapper" name="URLMapper"/>
<transport name="http">
        <requestFlow>
                <handler type="URLMapper"/>
                <handler type="java:org.apache.axis.handlers.http.HTTPAuthHandler"/>
        </requestFlow>
</transport>
```

Questo elemento contiene i due handler contenuti per default nella HTTP Transport chain: URLMapper legge l'http URL e setta il nome del servizio Axis da invocare come una proprietà del MessageContext; HTTPAuthHandler preleva username e password dall'HTTP Basic authentication header e li inserisce all'interno del MessageContext in una forma indipendente dal trasporto.

AXIS supporta alcune feature di sicurezza e sfruttando il modello di estendibilità di SOAP consente di aggiungerne di nuovi. In particolare, Axis fornisce all'interno del package org.apache.axis.handlers degli handler che possono essere usati per controllare l'accesso ai servizi in base alla username o in base ai ruoli. Questi controlli possono essere fatti sia a livello globale, nel qual coaso l'handler deve essere inserito nella global chain di richiesta, o di singolo servizio. I due handler sono:

- SimpleAuthenticationHandler che serve a controllare al runtime se l'utente associate alla richiesta corrente è autorizzato a dusare il servizio;
- SimpleAuthorizationHandler che consente di gestire liste di accesso per particolari servizi.

Una volta fatto il deploy dell'handler è possible specificare gli utenti autorizzati ad usare il servizio tramite il parametro allowedRoles. Il seguente frammento del file WSDD mostra la descrizione del servizio MySecureService che può essere utilizzato solo dall'utente pippo e dal gruppo di utenti identificati dal ruolo manager.

```
<service name="MySecureService">
...
        <parameter name="allowedRoles" value="manager,pippo"/>
</service>
```

Il SimpleAuthenticationHandler utilizza l'interfaccia SecurityProvider, contenuta nel package org.apache.axis.security per accedere al sistema di sicurezza in modo indipendente dai dettagli relativi alla sua implementazione. AXIS fornisce due implementazioni dell'interfaccia SecurityProvider:

- SimpleSecurityProvider usa un file di testo per moemorizzare username e password;
- ServletSecurityProvider usa I meccanismi di sicurezza del motore di servlet all'interno del quale gira AXIS.

Per utilizzare il securityProvider è necessario attivarlo nel file di configurazione della servlet (web.xml).

Definizione di un handler

Un handler è una classe Java che implementa l'interfaccia org.apache.axis.Handler. L'elemento principale di questa interfaccia è il metodo invoke che viene invocato dal motore AXIS e prende in input un MessageContext. Inoltre, l'interfaccia contiene metodi per gestire alcune opzioni ed ottenere dei meta-dati relative alle opzioni contenute nel descrittore di deployment dell'handler.

Tipicamente, un handler ha a che fare con l'estensione verticale delle funzionalità del nodo SOAP e quindi deve gestire degli header. L'handler recupera l'oggetto Message che rappresenta il messaggio SOAP dal MessageContext ed utilizza le API di SAAJ per accedere e manipolarne le varie parti. L'utilizzo di queste API è stato giù illustrato nella precedente Unità Didattica quando abbiamo analizzato come costruire messaggi SOAP lato client.

Nell'esaminare il contenuto del messaggio l'handler può dover fare il parsing dello stesso. Poiché il parser in caso di errori solleva una SAXException il metodo invoke deve essere scritto in modo da catturare tali eccezioni. A sua volta, l'handler deve segnalare l'errore lanciando un'eccezione di tipo AxisFault, il cui utilizzo verrà illustrato più avanti.

Esercitazione

Proviamo, ora, ad estendere il servizio InventoryCheck che abbiamo implementato nella LO4 in modo da fornire un servizio di logging e di notifica. Il servizio deve interagire con l'applicazione

client che abbiamo scritto nella precedente Unità Didattica. I due servizi vengono forniti tramite due handler: l'handler di logging è inserito all'interno della catena di richiesta di SOAPService; l'handler di notifica, invece, è inserito all'interno della catena di risposta di SOAPService. L'handler di jogging legge le informazioni all'interno dell'header Logger e li copia nel file di logging. L'handler di notifica legge l'indirizzo di posta elettronica contenuto nell'header Mail ed invia un messaggio di notifica all'indirizzo contenente il codice del prodotto richiesto ed il risultato della verifica. Per semplicità, simuliamo la spedizione della mail con la scrittura in un file.

Il codice dell'applicazione è contenuto nella cartella Esempi LO5 (inventoryCheckMail.zip).

Riferimenti

1. S. Graham e alt., Building Web Services in Java II ed., SAMS, 2005, cap. 3 pag. 121-140, Cap. 4 pag. 233-246
2. Simple Object Access Protocol (SOAP) 1.1, Documento W3C, (Maggio 2000)
3. Simple Object Access Protocol (SOAP) 1.2 Part 0: Primer, Documento W3C, (Giugno 2003)
4. Simple Object Access Protocol (SOAP) 1.2 Part 1: Messaging Framework, Documento W3C, (Giugno 2003)
5. Simple Object Access Protocol (SOAP) 1.2 Part 2: Adjuncts, Documento W3C, (Giugno 2003)
6. SOAP with Attachments API for Java (SAAJ) 1.2, Sun Microsystems, (Ottobre 2003)

UNIVERSITA' DEGLI STUDI DI SALERNO

Corso di Laurea di Informatica
Insegnamento di Programmazione su Reti

L.O. 6: WSDL e UDDI
U.D. 1: Descrizione di Servizi

Prof. Vincenzo Auletta
email: auletta@dia.unisa.it

22/5/2008

Descrizione di Servizi

Ruolo della descrizione dei servizi in un'architettura SOA

Un'architettura SOA definisce un modello di riferimento per sviluppare applicazioni distribuite "loosely couplet". In quest'architettura un'entità detta Requestor utilizza un servizio messo a disposizione dall'entità Provider in maniera indipendente dalla piattaforma hardware/software su cui opera il servizio. Il Requestor non deve necessariamente avere una conoscenza pregressa del servizio ma può scoprirlo interrogando un Registry, su cui sono registrati tutti i servizi messi a disposizione dai vari Provider. Ovviamente, l'architettura garantisce che il Requestor possa usare il servizio senza restrizioni dovute alla piattaforma hardware/software, ma in ogni caso ci sono alcune informazioni, funzionali e non funzionali, che il Provider deve rendere disponibili e che il Requestor deve acquisire per poter utilizzare il servizio. Tra le altre cose il Requestor ha bisogno di sapere:

- a quale URL è disponibile il servizio;
- che tipo di interazione viene offerta (RPC o altro);
- che tipo di messaggi devono essere utilizzati per l'interazione (SOAP o altro);
- come devono essere strutturati i messaggi da utilizzare per interagire con il servizio (se si usa SOAP bisogna capire cosa deve essere inserito nel body, se ci devono essere degli header block, che regole di codifica utilizzare per la serializzazione del messaggio, che versione di SOAP usare, ecc.);
- quale protocollo di trasporto utilizzare per inviare questi messaggi.

Questa lista è assolutamente incompleta ed a queste informazioni se ne potrebbero aggiungere dele altre in specifiche situazioni, come per esempio l'informazione su eventuali restrizioni nell'utilizzo del servizio (es. acceso consentito solo gli utenti registrati) o sulle modalità di fornitura del servizio (es. servizio a pagamento).

La maniera più semplice di gestire il passaggio di queste informazioni dal Provider ai Requestor è quella di immaginare che il Provider prepari un documento esplicativo delle modalità di utilizzo del servizio e lo distribuisca ai suoi potenziali clienti o lo renda disponibile sul un sito Web. In questa maniera i Requestor che hanno recuperato questo documento possono leggere queste istruzioni e comportarsi di conseguenza. Questo approccio ha almeno due problemi:

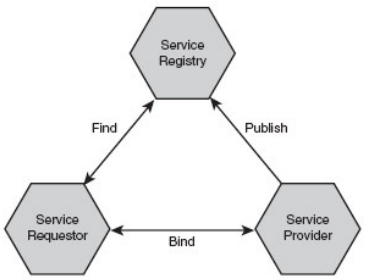
1. presuppone che il Provider conosca i suoi Requestor (nel caso in cui il documento deve essere distribuito) o che i Requestor conoscano il Provider (nel caso in cui il documento viene pubblicato su un sito Web);
2. una descrizione testuale si presta ad essere interpretata in vari modi e potrebbero sorgere problemi di interoperabilità dovuti a errate interpretazioni delle istruzioni;
3. anche se la descrizione è molto precisa deve essere interpretata da una persona che deve scrivere del codice secondo le istruzioni indicate nel documento.

Un approccio molto più efficace sarebbe definire una struttura formale del documento di descrizione del servizio che viene reso disponibile ad un certo URL ben conosciuto. In questo modo, i Requestor possono scoprire i servizi senza dover necessariamente conoscere in anticipo il loro nome o chi li ha resi disponibili, il documento non è ambiguo e può anche essere processato da

un'applicazione software che può generare il codice necessario per interagire con il servizio senza nessun intervento umano. Sul mercato sono disponibili diversi tool che rendono estremamente semplice lo sviluppo di applicazioni che utilizzano Web Services attraverso l'utilizzo di codice generato a partire da documenti di descrizione dei servizi strutturati secondo degli standard.

Questo documento di descrizione del servizio svolge un ruolo fondamentale nell'ambito di un'architettura SOA ed è coinvolto in ognua delle tre operazioni fondamentali:

- **publish** – con questa operazione il Provider pubblica sul Registry il documento di descrizione del servizio che ha messo a disposizione;
- **find** – con questa operazione il Requestor cerca nel Registry documenti di descrizione di servizi che corrispondano alle sue richieste;
- **bind** – con questa operazione il Requestor interagisce con il servizio fornito dal Provider secondo le modalità specificate nel documento di descrizione del servizio.



Descrizione dei Servizi

Una descrizione completa del servizio dovrebbe contenere tutte le informazioni che servono al Requestor per poter scegliere quale servizio utilizzare e interagire con esso. Potremmo riassumere queste informazioni in cinque categorie:

1. **chi** – chi può usare il servizio
2. **cosa** – cosa offre il servizio
3. **dove** – dove è disponibile il servizio
4. **perché** – perché si dovrebbe usare il servizio
5. **come** – come viene fornito il servizio

La descrizione di un servizio consiste di due parti:

- la descrizione funzionale fornisce i dettagli operativi su come il Web Service deve essere invocato (es. URL, formato dei messaggi, protocollo di trasporto, ecc.);
- la descrizione non funzionale contiene dettagli che possono essere utili al Requestor per decidere se utilizzare il servizio o meno ed in che modo utilizzarlo ma non sono strettamente legate alle modalità di interazione (es. politiche di sicurezza).

Descrizione Funzionale

La descrizione funzionale di un Web Service specifica tutto ciò che il Requestor deve fare per poter invocare un dato servizio. In particolare, esso specifica: le operazioni che il Web Service mette a disposizione (cosa), la sintassi dei messaggi da utilizzare per invocare il servizio (come), l'URL dove è possible invocare il servizio (dove). Questa descrizione viene espressa utilizzando un linguaggio IDL (*Interface Definition Language*). I linguaggi IDL sono stati utilizzati anche in altri ambiti. Nel caso specifico dei Web Services, viene utilizzato WSDL che è un linguaggio IDL basato su XML ampiamente supportato e adottato dalla W3C come base per il processo di standardizzazione della descrizione di servizi.

Spesso in WSDL, per comodità, la descrizione del Web Service viene divisa in una definizione dell'implementazione, che specifica dove il servizio è disponibile, ed una definizione dell'interfaccia del servizio, che specifica cosa il servizio offre e come lo si deve utilizzare.

Descrizione nonfunzionale

La descrizione funzionale è fondamentale per poter invocare un Web Service ma in molti casi può non essere sufficiente per decidere se invocarlo o meno e dovrebbe essere integrata da una descrizione non funzionale. Non esiste una maniera precisa di specificare cosa c'è in una descrizione non funzionale. In gener contiene tutto quello che non può stare in una descrizione funzionale. Per esempio, una descrizione non funzionale si occupa di specificare che tipo di funzione viene svolta dal Web Service e come si integra del in modelli di business più ampi (perchè), di dare dettagli su chi fornisce il servizio (chi) e sulle modalità di offerta del servizio, come per esempio quali meccanismi sono attivati per garantire la privatezza e l'autenticità dei dati o quali livelli di QoS sono supportati (come). Si noti che le descrizioni non funzionali potrebbero anche modificare la struttura dei messaggi SOAP, per esempio aggiungendo dei security-header, ma comunque non riguardano il contenuto principale presente nel body del messaggio.

La tabella seguente riassume come le informazioni che devono essere presenti all'iterno di un documento di descrizione di un Web Service sono divise tra la descrizione funzionale e quella non funzionale.

Chi	Descrizione non funzionale
Cosa	Descrizione funzionale
Dove	Descrizione funzionale
Perchè	Descrizione non funzionale
Come	Sia descrizione funzionale che non funzionale

Linguaggi IDL

Ilinguaggi IDL sono stati introdotti insieme ai primi modelli di computazione distribuita per descrivere le interfacce dei vari componenti del sistema.
Il primo esempio importante di utilizzo di un linguaggio IDL risale al 1994 alla specifica di RPC della Open Software Foundation's Distributed Computing Environment (DCE). Questo importantissimo concetto fu poi ripreso in tutti I successive progetti di computazione distribuita, come l'IDL definito dalla OMG (Object Management Group's) per CORBA e i COM IDL e COM ODL (Object Definition Language) definiti da Microsoft. Come succede spesso, tutti questi linguaggi IDL sono spesso focalizzati su determinati aspetti delle tecnologie per cui sono stati definiti e sono più o meno compatibili tra loro. Da questo punto di vista WSDL rappresenta una ltro grosso passo avanti concettuale nell'evoluzione dei linguaggi IDL. Infatti, essendo WSDL basato su XML, è totalmente indipendente dalla tecnologia utilizzata per fornire i Web services e dal linguaggio di programmazione in cui i servizi verranno implementati. In particolare, WSDL può essere utilizzato per descrivere servizi che non sono implementati utilizzando SOAP.
I linguaggi IDL sono molto utili nello sviluppo di applicazioni distribuite "loosely coupled" perché forniscono un valido meccanismo per gestire l'eterogeneità del sistema. La strategia universalmente adottata per gestire l'eterogeneità è stata quella di definire una maniera standard di comunicazione tra applicazioni che sia indipendente dall'implementazione delle applicazioni. Tale strategia deve consistere di due parti:

- definire un modo standard per fare un'invocazione, includendo il modo in cui viene individuato il nome della funzione da invocare, come si passano gli argomenti e come questi

vengono codificati, come si gestiscono gli errori e cosi via (questo è il lavoro svolto da DCE, COM, Corba, SOAP);
- specificare cosa si deve invocare, includendo il nome delle operazioni, le loro firme, i tipi dei valori restituiti, le eventuali eccezioni e così via (questo è il lavoro fatto dall'IDL).

In tutte le architetture per la computazione distribuita l'utilizzo di un linguaggio IDL è accompagnato dalla presenta di un compilatore IDL che combina la descrizione delle interfacce contenute nel file IDL con le convenzioni definite dal middleware per generare del codice che permette ad una applicazione di interagire con una applicazione remota senza doversi occupare di tutti i dettagli relativi all'interazione tramite la rete. Tipicamente, il compilatore IDL genera un modulo software detto stub (o client proxy) che espone la stessa inerfaccia del servizio remoto che si vuole invocare; l'applicazione client interagisce con il servizio remoto semplicemente invocando localmente I metodi esposti dallo stub.

Analogamente, il fornitore del servizio include la sua implementazione all'interno di un modulo software detto skeleton (o server stub) che è generato automaticamente dal compilatore IDL. L'interazione tramite la rete avviene tra lo stub e lo skeleton in maniera completamente trasparente sia all'applicazione client che a quella server.

Originariamente DCE IDL consentiva di specificare solo interface di funzioni. Successivamente, CORBA IDL ha introdotto numerose estensioni, quali la definizione di interface di oggetti ed implementando anche meccanismi di derivazione delle interface, divenendo il linguaggio di riferimento per sistemi distribuiti ad oggetti. La sintassi di CORBA IDL ricorda molto quella del C++. Il linguaggio ODL di Microsoft's ODL non è compatibile con quello di CORBA nè al livello di sintassi e neanche a livello di impostazione. Nonostante questo, però, gli oggetti COM vengono utilizzati in maniera molto simile agli oggetti CORBA

I linguaggi di programmazione moderni hanno in parte eliminato la necessità di utilizzare IDL perchè ogni componente compilato include in se una serie di metadata quail la sua classe padre, le proprietà ed i metodi, le interface supportate. Questo rende possibile, per esempio, utilizzare direttamente RMI per invocare metodi remoti senza dover creare prima file IDL. Tuttavia, quando ci troviamo a dover gestire applicazioni eterogenee, con pià componenti scritte in diversi linguaggi di programmazione l'approccio più valido è quello di separare l'interfaccia dalla sua implementazione.

WSDL

WSDL (Web Service Description Language) è un linguaggio IDL che viene utilizzato per fornire la descrizione funzionale di un Web Service. Esso, infatti, consente di descrivere la sintassi dei messaggi associati all'invocazione ed alla risposta di un Web Service.

In particolare, un documento WSDL descrive tre proprietà fondamentali di un Web Service:
- cosa fa il servizio — fornisce l'elenco delle operazioni fornite dal Web Service, gli argomenti che prendono in input ed i valori che restituiscono;
- come si deve invocare il servizio — specifica il formato dei dati ed i protocolli che devono essere utilizzati per invocare il servizio;
- dove è disponibile il servizio — fornisce dettagli sull'indirizzo di rete dove il servizio è disponibile, per esempio l'URL.

La versione 1.1 di WSDL è stata presentata alla W3C come proposta di standard da un gruppo di aziende nel 2001 ed è ampiamente accettata e supportata. Recentemente la W3C ha pubblicato la

specifica 2.0 di WSDL che modifica profondamente la precedente specifica e rende molto più semplice e allo stesso tempo flessibile l'utilizzo di WSDL.

Una descrizione deI servizio WSDL non è altro che un documento XML conforme allo schema di WSDL.

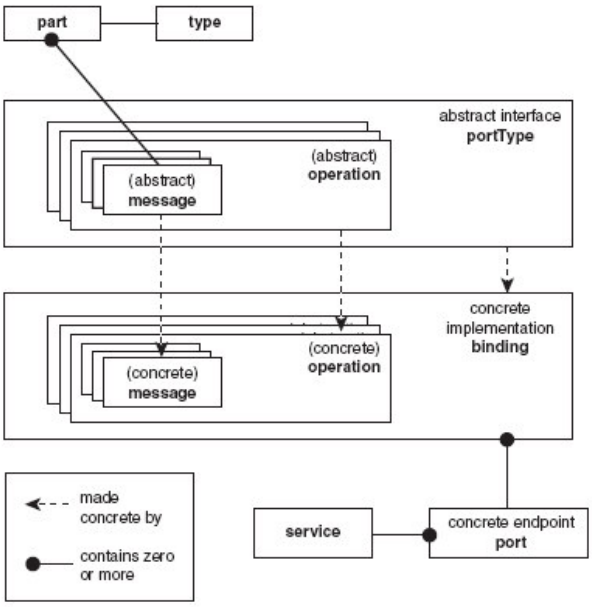
L'elemento radice dello schema è l'elemento <definitions>: all'interno di questo elemento sono definiti gli elementi che compongono una descrizione di un Web service. Gli elementi principali sono:

- portType
- message
- types
- binding
- service

Diamo ora una descrizione sommaria di ciascuno di questi elementi.
portType -- rappresenta l'interfaccia di un Web Service (in maniera simile ad un'inerfaccia Java) e contiene al suo interno una sequenza di elementi operation che rappresentano le operazioni fornite dal Web Service (in maniera simile ai metodi di un'interfaccia Java). WSDL vede le operation come manipolatrici di elementi message: per ogni operation viene specificato una sorte di firma costituita dal nome dell'operazione, dall'elemento message che prende in input e dall'elemento message che fornisce in output.
message – definisce il formato di un messaggio, costituito da uno o più argomenti passati in input o in output ad una operazione. Ogni elemento message può a sua volta essere ripartito in elementi part.
types – contiene le definizioni delle elementi e degli attributi e le dichiarazioni dei tipi di dati XML che vengono referenziate all'interno del documento WSDL.
binding – contiene i dettagli relativi a come gli elementi dell'interfaccia astratta (gli elementi operation contenute nell'elemento portType) sono convertiti in una rappresentazione concreta.
port – espirme l'indirizzo di un endpoint dove l'implementazione descritta da un elemento binding è disponibile.
service – è semplicemente una collezione di elementi port.

Da questa breve descrizione si può dedurre che l'elemento portType descrive cosa il Web Service fornisce, l'elemento binding descrive come il Web Service fornisce i suoi servizi e gli elementi port e service descrivono dove il servizio è disponibile.

Nella figura sono mostrate le relazioni tra i vari elementi dello schema WSDL.



Sebbene la struttura di un documento WSDL possa sembrare molto complessa non bisogna spaventarsi. La struttura di questi documenti è rigorosamente definita da uno schema XML e oramai ci sono tantissimi tool sul mercato che permettono di generare automaticamente il file WSDL che descrive un Web Service a partire dall'interfaccia della classe Java che lo implementa o, viceversa, a generare lo stub e lo skeleton a partire dal file WSDL. Tuttavia, una certa conoscenza dello schema di WSDL ò necessari perché a volte è necessario intervenire sulle descrizioni generate automaticamente da questi tool per adttarle a specifiche esigenze.

Elementi di una descrizione WSDL

Descriveremo la sintassi dei vari elementi di un documento WSDL utilizzando come caso di studio la descrizione di un Web Serivces della società Skatestown che fornisce un'unica operazione: consente di controllare il prezzo di un prodotto fornendo in input il suo codice identificativo.

Un documento WSDL deve essere conforme allo schema XML definito per il linguaggio WSDL e identificato dall'URI "http http://schemas.xmlsoap.org/wsdl/". Secondo questo schema, l'elemento radice di un documento WSDL è l'elemento <definitions> e contiene tutti gli altri elementi. All'interno di un elemento definitions possono comparire:

- Zero o più elementi <documentation>;
- Zero o più elementi <import>;
- Un elemento <types> opzionale;
- Zero o più elementi <message>;
- Zero o più elementi <portType> (normalmente ce ne è uno solo);
- Zero o più elementi <binding> (normalmente ce ne è uno solo corrispondete all'elemento portType);
- Zero o più elementi <service> (normalmente ce ne è uno solo corrispondete all'elemento binding);

L'elemento definitions contiene un attributo name ed un attributo targetNamespace che identifica il namespace in cui vengono inserite tutte le definizione contenute nel documento WSDL. Quando all'interno di un elemento del documento WSDL si vuole fare riferimento ad una ltro elemento del documento bisogna fare attenzione a qualificare il suo nome con il nome del suo namespace (che è

quello specificato come targetNamespace). Infine, l'elemento definitions contiene la definizione dei namespace utilizzati nel resto del documento.

```
<?xml version="1.0"?>
<definitions name="PriceCheck"
targetNamespace="http://www.skatestown.com/services/PriceCheck"
xmlns:pc="http://www.skatestown.com/services/PriceCheck"
xmlns:avail="http://www.skatestown.com/ns/availability"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
. . .
</definitions>
```

L'elemento portType è il punto di partenza ideale per capire la struttura di un documento WSDL. L'elemento portType descrive l'interfaccia di un Web Service e, sulla falsa riga delle interfacce Java, contiene un elenco di firme astratte di metodi. Ogni elemento portType ha un attributo name che deve essere univoco all'interno del documento WSDL. In generale, si consiglia di avere documenti WSDL con un unico elemento portType per facilitare il riutilizzo di queste descrizioni e la loro composizione per costruire descrizioni di servizi più complessi.

All'interno di un elemento portType compare una sequenza di elementi operation, ognuno dei quali rappresenta una operazione fornita dal servizio. Un'elemento operation è l'equivalente della firma di un metodo all'interno di un'interfaccia Java che specifica il nome del metodo il tipo dei parametri ed il tipo del valore restituito. Gli elementi operation definiti all'interno del portType sono identificati da un attributo name il cui valore deve essere univoco all'interno dell'elemento portType a cui appartengono (in due portType differenti ci possono essere operazioni con lo stesso nome).
L'operazione è vista come un manipolatore di messaggi: un'elemento operation può ricevere al più un messaggio in input e restituire al più un messaggio in output. I messaggi ricevuti in input e forniti in output fanno riferimento alle definizioni di elementi message che sono definiti in un altro punto del documento WSDL e devono essere referenziati con il loro nome qualificato. La presenza o meno degli elementi input e output ed il loro ordine all'interno dell'elemento operation è utilizzato da WSDL per capire di che tipo di operazione si tratta (questo aspetto verrà trattato più avanti). Nel caso in cui ad un elemento input, output o fault non viene associato un nome allora lo schema gli assegna un nome di default dipendente dal tipo di primitiva di trasmissione. Ad esempio, per l'operazione XXX di tipo Request-Response gli elementi di input e output si chiamano XXXRequest e XXXResponse. Invece, gli elementi fault devono avere necessariamente un nome. Inoltre, per le operazioni che vengono implementate RPC-style è possibile specificare anche l'attributo parameterOrder, che elenca in che ordine i parametri devono essere inseriti nella chiamata RPC. L'utilizzo di parameterOrder, però, introduce all'interno della descrizione astratta del servizio un'informazione relativa alla sua implementazione.

Nel nostro esempio, il portType prevede una sola operazione checkPrice, che definisce un messaggio di input ed un messaggio di output.

```
<!-- Port type definitions -->
<portType name="PriceCheckPortType">
        <operation name="checkPrice">
                <input message="pc:PriceCheckRequest"/>
                <output message="pc:PriceCheckResponse"/>
```

```
        </operation>
</portType>
```

L'elemento message è il formato astratto di un qualsiasi messaggio di input, output o di errore. Il messaggio può essere visto o come un documento XML (document-centric) o come la lista dei parametri nella chiamata ad un metodo remoto (RPC-style messaging). Un documento WSDL può contenere un numero arbitrario di elementi message (anche zero). Ogni message ha un attributo name che lo identifica univocamente all'interno del documento WSDL, e contiene una sequenza di elementi part.

Il meccanismo delle part consente di scomporre il messaggio in unità più piccole che possono essere trattate separatamente in funzione del particolare protocollo utilizzato per implementare il servizio. Per esempio, nel caso si utilizzasse SOAP per implementare il Web Service, una parte del messaggio potrebbe essere inserita nel Body e la'ltra all'interno di un Headerblock. Questa corrispondenza tra le varie part e i protocolli utilizzati per implementare il servizio viene definite più avanti nell'elemento binding.

Ogni elemento part ha due attributi che ne definiscono nome e tipo. L'attributo name lo identifica univocamente all'interno dell'elemento operation a cui appartiene. Il secondo attributo può essere element o type: element fa riferimento al nome di un elemento globale che è definito all'interno del documento WSDL; type fa riferimento al nome di un tipo XML Schema o ad una altro tipo di dati definito all'interno dell'elemento types.

Nel nostro esempio sono definiti i due message PriceCheckRequest e PriceCheckResponse che sono forniti in input ed in output all'operazione definiti a prima. Ciascun messaggio è costituito da una sola part e fa riferimento al nome di un elemento appartenente al namespace avail.

```
<message name="PriceCheckRequest">
        <part name="sku" element="avail:sku"/>
</message>

<message name="PriceCheckResponse">
        <part name="result" element="avail:StockAvailability"/>
</message>
```

La scelta di usare type o element per identificare il tipo di una part è molto importante. Quando si utilizza element si sta specificando che il contenuto del messaggio che verrà spedito deve essere esattamente l'elemento XML indicato (nell'esempio sopra il messaggio PriceCheckRequest deve contenere l'elemento XML sku appartenente al namespace http://www.skatestown.com/ns/availability). L'elemento attribute dovrebbe essere usato quando si sta descrivendo Web Services document-oriented. Quando si utilizza type, invece, si sta specificando che il contenuto del messaggio deve essere del tipo indicato ma la rappresentazione di questo tipo sarà definita dal binding. Per capire la differenza possiamo vedere come cambierebbe la rappresentazione del messaggio PriceCheckResponse nel caso in cui si utilizzasso element e type.

```
<!-- rappresentazione di PriceCheckResponse se viene usato l'attributo element -->
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
        <StockAvailability xmlns="http://www.skatestown.com/ns/availability">
                <sku>123</sku>
                <price xmlns="">100.0</price>
                <quantityAvailable xmlns="">12</quantityAvailable>
        </StockAvailability>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<!-- rappresentazione di PriceCheckResponse se viene usato l'attributo type -->
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
        <ns1:checkPriceResponse
           soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           xmlns:ns1="http://www.skatestown.com/services/PriceCheck">
              <result href="#id0"/>
        </ns1:checkPriceResponse>
        <multiRef id="id0" soapenc:root="0"
           soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           xsi:type="ns2:availabilityType"
           xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
           xmlns:ns2="http://www.skatestown.com/ns/availability">
                <sku xsi:type="xsd:string">123</sku>
                <price xsi:type="xsd:double">100.0</price>
                <quantityAvailable xsi:type="xsd:integer">12</quantityAvailable>
        </multiRef>
   </soapenv:Body>
</soapenv:Envelope>
```

L'attributo element dovrebbe essere usato quando si sta descrivendo Web Services document-oriented; l'attributo type dovrebbe essere usato quando si sta descrivendo un Web Services RPC-style.

Abbiamo visto che nella definizione delle part di un messaggio è possibile far riferimento ai nomi di elementi globali o di tipi di dati. Il sistema di tipi di default di WSDL è XML Schema. Tuttavia, è possibile definire all'interno del documento elementi globali o altri tipi di dati usando i meccanismi di XML Schema. Le definizioni degli elementi e dei tipi di dati vanno inserite all'interno dell'elemento types. Ogni documento WSDL può contenere un solo elemento types che può contenere al suo interno più frammenti di schemi. L'elemento schema contenuto in types avrà un suo targetNamespace differente da quello del documento WSDL. Utilizzando il meccanismo di import di XML Schema è possibile importare all'interno dell'elemento types schemi XML definiti in altri documenti. Spesso, i documenti WSDL hanno un elemento types che contiene semplicemente l'importazione di schemi da altri documenti.

L'esempio mostra un elemento types in cui vengono definiti l'elemento globale avail:sku ed il tipo complesso availabilityType.

```
<types>
  <xsd:schema
     targetNamespace=”http://www.skatestown.com/ns/availability” >
  <xsd:element name=”sku” type=”xsd:string” />
  <xsd:complexType name=”availabilityType”>
          <xsd:sequence>
              <xsd:element ref=”avail:sku”/>
              <xsd:element name=”price” type=”xsd:double”/>
              <xsd:element name=”quantityAvailable” type=”xsd:integer”/>
          </xsd:sequence>
  </xsd:complexType>

  <xsd:element name=”StockAvailability” type=”avail:availabilityType” />
</xsd:schema>
</types>
```

Nella sezione types è possibile definire dei tipi di dati array. XML Schema non prevede array e quindi questi tipi vengono definiti a partire dal tipo soap:array. La definizione di un array, però, ha una sintassi particolare come mostrato in quest’esempio.

```
<wsdl:types>
   <xsd:schema targetNamespace="http://my_example.org/types"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
         <xsd:complexType name="arrayOfItem">
              <xsd:complexContent>
                   <xsd:restriction base="soapenc:Array">
                        <xsd:attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"/>
                   </xsd:restriction>
              </xsd:complexContent>
         </xsd:complexType>
    </xsd:schema>
</wsdl:types>
```

Fino ad ora abbiamo visto come il portType permetta di definire un’interfaccia astratta del Web Service specificando le operazioni fornite e indicando anche quali elementi o tipi XML queste operazioni prendono in input o restituiscono in output. Adesso dobbiamo analizzare quale deve essere il formato dei messaggi che devono essere utilizzati per invocare questi servizi. Ovviamente, questo dipende dal particolare tipo di protocollo che decidiamo di utilizzare per implementare il servizio (es. SOAP o semplicemente http post con i documenti XML incapsulati nel corpo dei pacchetti), o se utilizziamo un’interazione RPC-style o document-centric.
Questi dettagli vengono specificati all’interno dell’elemento binding e nel documento WSDL ci possono essere tanti elementi binding quante sono le possibili implementazioni di un dato portType.

Ogni elemento binding ha un attributo name ce lo identifica univocamente all’interno del documento WSDL ed un attributo type che specifica il nome qualificato dell’elemento portType di cui questo binding specifica l’implementazione. Convenzionalmente, il nome del binding è la concatenazione del portType implementato e del protocollo utilizzato per l’implementazione. Tipicamente, un documento WSDL contiene un solo elemento binding per favorire il riutilizzo dei documenti WSDL. L’elemento binding contiene al suo interno una sequenza di elementi operation, che replicano le operation del portType che si vuole implementare. Per ogni operation si dovrebbero

specificare le istruzioni necessarie a implementare quell'operazione. In genere, queste istruzioni dipendono fortemente dal protocollo che si decide di utilizzare. Per poter supportare una vasta gamma di protocolli la specifica di WSDL ha previsto una convenzione di estendibilità che permette di estendere il binding con elementi provenienti da altri namespace XML consentendo di definire binding per tutti i possibili protocolli. La specifica di WSDL prevede esplicitamente tre estensioni: SOAP/HTTP, HTTP GET/POST, e SOAP with MIME attachments.

L'estensione per SOAP/HTTP, identificata dal namespace soap:, prevede diversi elementi:
soap:binding – permette di definire lo stile di interazione ed il tipo di protocollo di trasporto utilizzato nel binding. Queste istruzioni si applicano all'intero binding e non ad una singola operazione. Lo stile di interazione è definito dal valore dell'attributo style mentre il protocollo di trasporto è indicato dall'attributo transport. Le scelte possibili per transport sono: http, ftp e smtp (ma è possibile definire altri binding). Le scelte possibili per lo stile di interazione sono due: document e rpc. Uno stile di interazione document-centric significa che il body del messaggio SOAP deve essere interpretato come semplice XML. Uno stile di interazione rpc-based indica che il body del messaggio SOAP deve essere interpretato secondo le regole della convenzione RPC di SOAP. In genere si dovrebbe utilizzare lo stile document-centric per portType in cui le part dei messaggi sono state definite usando l'attributo element, mentre si dovrebbe usare lo stile RPC-based se si è utilizzato l'attributo type.
Per esempio, l'elemento

<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

specifica che il binding è implementato tramite SOAP su HTTP e lo stile di interazione è document-centric.

soap:operation – contenuto all'interno di un elemento operation, permette di definire il formato specifico dei messaggi SOAP che implementano quella operazione. L'attributo soapAction specifica il valore che dovrebbe essere copiato nel campo SOAPAction dell'header http.

soap:body – contenuto all'interno di elementi input o output di un'operation, specifica il contenuto del body del messaggio SOAP che implementa il corrispondente message definito nel portType. L'elemento fa riferimento ad una part del messaggio. L'attributo use specifica il modo in cui il contenuto della part deve essere serializzato nel messaggio SOAP. Se il valore dell'attributo è literal allora il contenuto deve essere copiato cosi com'è; se, invece, il valore è encoded, allora il contenuto del body viene serializzato secondo le regole di codifica specificate dall'attributo encodingStyle. La regola da seguire è la seguente:

- se lo stile di interazione è document-centric il valore dell'attributo use è literal per rappresentare delle part che sono stati definite utilizzando l'attributo element;
- se lo stile di interazione è rpc-style il valore dell'attributo use è encoded per rappresentare delle part che sono state definite utilizzando l'attributo type.

soap:header – contenuto all'interno di elementi input o output di un'operation, specifica il contenuto del header del messaggio SOAP che implementa il corrispondente message definito nel portType. L'elemento fa riferimento ad una part del messaggio. L'utilizzo dell'attributo use è analogo al caso descritto in precedenza. In questo caso, però, l'attributo use dovrebbe essere sempre uguale a literal.

L'esempio mostra il contenuto dell'elemento binding che implementa PriceCheckPortType utilizzando SOAP su http con stile di interazione rpc-style. I messaggi sono codificati usando le regole di codifica di SOAP 1.1

```
<binding name="PriceCheckSOAPBinding" type="pc:PriceCheckPortType">
   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="checkPrice">
        <soap:operation/>
                <input>
                   <soap:body use="encoded"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://www.skatestown.com/services/PriceCheck"/>
                </input>
                <output>
                   <soap:body use="encoded"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                        namespace="http://www.skatestown.com/services/PriceCheck"/>
                </output>
  </operation>
</binding>
```

L'elemento Service contiene al suo termine una collezione di elementi Port.
L'elemento Port è molto semplice. Il suo unico scopo è di specificare l'indirizzo dell'endpoint dove la particolare implementazione di un Web Serivce fornita da un elemento binding è disponibile. Per questo motivo ogni elemento Port ha un nome che lo identifica univocamente all'interno del documento WSDL e un attributo binding che fa riferimento all'elemento binding di cui si fornisce l'indirizzo. L'estensione per SOAP/http permette di specificare questo indirizzo tramite un URL.

Esercitazione

Scrivere il documento WSDL che descrive il Web Service CheckPrice utilizzando uno stile RPC-style. Il Web Service fornisce soltanto il metodo priceCheck che prende in input un elemento di tipo string che rappresenta il codice di un prodotto e restituisce un elemento di tipo Availability che contiene il codice ed il prezzo del prodotto e la quantità disponibile.

```
<?xml version="1.0"?>
<definitions xmlns:pc="http://www.skatestown.com/services/PriceCheck"
xmlns:avail="http://www.skatestown.com/ns/availability"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://www.skatestown.com/services/PriceCheck" name="PriceCheck">

<types>
        <xsd:schema targetNamespace=http://www.skatestown.com/ns/availability
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                <xsd:complexType name="availabilityType">
                        <xsd:sequence>
                                <xsd:element name="sku" type="xsd:string"/>
                                <xsd:element name="price" type="xsd:double"/>
                                <xsd:element name="quantityAvailable" type="xsd:integer"/>
                        </xsd:sequence>
```

```
                </xsd:complexType>
        </xsd:schema>
</types>

<message name="PriceCheckRequest">
        <part name="sku" type="xsd:string"/>
</message>
<message name="PriceCheckResponse">
        <part name="result" type="avail:availabilityType"/>
</message>

<portType name="PriceCheckPortType">
        <operation name="checkPrice">
                <input message="pc:PriceCheckRequest"/>
                <output message="pc:PriceCheckResponse"/>
        </operation>
</portType>

<binding name="PriceCheckSOAPBinding" type="pc:PriceCheckPortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="checkPrice">
                <soap:operation/>
                <input>
                        <soap:body use="encoded"
                                encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”
                                namespace="http://www.skatestown.com/services/PriceCheck"/>
                </input>
                <output>
                        <soap:body use="encoded"
                                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                                namespace="http://www.skatestown.com/services/PriceCheck"/>
                </output>
        </operation>
</binding>

<service name="PriceCheckService">
        <port name="PriceCheck" binding="pc:PriceCheckSOAPBinding">
                <soap:address location="http://localhost:8080/axis/services/PriceCheck"/>
        </port>
</service>

</definitions>
```

Esercitazione

Riscrivere il documento WSDL che descrive il Web Service CheckPrice utilizzando uno stile Document-centric. Il Web Service fornisce soltanto il metodo priceCheck che prende in input un elemento di tipo string che rappresenta il codice di un prodotto e restituisce un elemento di tipo Availability che contiene il codice ed il prezzo del prodotto e la quantità disponibile.

```
<?xml version="1.0"?>
```

```
<definitions xmlns:pc="http://www.skatestown.com/services/PriceCheckDoc"
xmlns:avail="http://www.skatestown.com/ns/availability"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://www.skatestown.com/services/PriceCheckDoc"
name="PriceCheckDoc">
        <types>
                <xsd:schema targetNamespace="http://www.skatestown.com/ns/availability"
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                   <xsd:element name="sku" type="xsd:string"/>
                   <xsd:complexType name="availabilityType">
                        <xsd:sequence>
                                <xsd:element name="sku" type="xsd:string"/>
                                <xsd:element name="price" type="xsd:double"/>
                                <xsd:element name="quantityAvailable" type="xsd:integer"/>
                        </xsd:sequence>
                   </xsd:complexType>
                   <xsd:element name="stockAvailability" type="avail:availabilityType"/>
                </xsd:schema>
        </types>

        <message name="PriceCheckDocRequest">
                <part name="sku" element="avail:sku"/>
        </message>
        <message name="PriceCheckDocResponse">
                <part name="result" element="avail:stockAvailability"/>
        </message>

        <portType name="PriceCheckDocPortType">
                <operation name="checkPrice">
                        <input message="pc:PriceCheckDocRequest"/>
                        <output message="pc:PriceCheckDocResponse"/>
                </operation>
        </portType>

        <binding name="PriceCheckDocSOAPBinding" type="pc:PriceCheckDocPortType">
                <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
                <operation name="checkPrice">
                        <soap:operation
                       soapAction="http://www.skatestown.com/services/PriceCheckDoc/checkPrice"/>
                        <input>
                                <soap:body use="literal"/>
                        </input>
                        <output>
                                <soap:body use="literal"/>
                        </output>
                </operation>
        </binding>

        <service name="PriceCheckDocService">
                <port name="PriceCheckDoc" binding="pc:PriceCheckDocSOAPBinding">
```

```
                    <soap:address
                        location="http://localhost:8080/axis/services/PriceCheckDoc"/>
                </port>
        </service>
</definitions>
```

Riutilizzabilità degli schemi WSDL

WSDL definisce un elemento import che consente di collegare insieme vari documenti. Questo elemento consente di riutilizzare documenti WSDL definiti in file differenti per costruire documenti più complessi. In realtà, un elemento import college una locazione sulla rete ad un namespace XML. Questo funzionamento potrebbe sembrare simile a quello dell'import di XML Schema ma i due meccanismi funzionano diversamente e non conviene mischiarli: è bene usare l'import di WSDL per importare altre descrizioni WSDL e l'import di XML schema all'interno dell'elemento types per importare altri schemi.
La sintassi dell'elemento import è

<import namespace="uri" location="uri"/>

dove il valore dell'attributo location è un suggerimento su dove si trova il documento WSDL da importare. Il valore dell'attributo namespace dovrebbe essere uguale al targetNamespace del documento WSDL.

Molti programmatori utilizzano il meccanismo di import per dividere il documento WSDL in due parti:

- una definizione dell'interfaccia del servizio che contiene l'elemento portType con i relativi elementi message e types e l'elemento binding;
- una definizione dell'implementazione del servizio che include l'interfaccia da implementare e contiene l'elmento service.

In questo modo, tutte le parti riutilizzabili del documento WSDL sono inserite nella definizione dell'interfaccia che può essere importata da tutti quelli che vogliono fornire un'implementazione conforme a quella descrizione.

Esercitazione

Scrivere il documento WSDL che descrive il Web Service POSubmission che fornisce un sola operazione per sottomettere ordini di acquisto alla Skatestown. L'operazione DoSubmision prende in input un ordine d'acquisto, definito dallo schema po.xsd, e restituisce una fattura, definita dallo schema invoice.xsd. Entrambi gli schemi sono stati definiti nella LO2. Organizzare il documento WSDL in modo da separare la definizione dell'interfaccia da quella dell'implementazione e riutilizzare gli schemi invoice.xsd e po.xsd.

I vari file wsdl sono contenuti nella directory degli esempi (POSubmission.rar).

Primitive di Trasmissione

La specifica di WSDL 1.1 definisce Quattro tipi di operazioni, definite con il termine di primitive di tramissione. Ogni primitiva definisce una relazione tra i messaggi che verranno ricevuti e quelli che verranno trasmessi dal Web Service. Per ogni operazione definita all'interno del portType la

specifica del tipo di primitiva adottata è implicita nella scelta dell'ordine con cui compiaono gli elementi message al suo interno.
Le quattro primitive di tramisssione sono:
Request-Response (input – output – fault)
È il tipo di operazione più comune in cui il Web Service riceve una richiesta all'interno del messaggio di input e restituisce una risposta o una collezione di messaggi di errore. Rappresenta lo schema classico di un'operazione RPC e può essere usata sia per accedere ad una risorsa (query) che per modificare lo stato di una risorsa (update) ricevendo in risposta una notifica del nuovo stato.
One-Way (input**)**
Un operazione one-way riceve un messaggio di input ma non fornisce nessuna risposta ne messaggi di errore. In genere viene usata per modificare lo stato del Provider. Molto spesso, con servizi implementati su SOAP e http le operazioni one-way sono implementate come request-response dove il messaggio di risposta è il pacchetto di ACK di http.
Notification (output)
In un'operazione di tipo notification il Web Service di sua iniziativa invia un messaggio al Requestor. Ovviamente, questo tipo di operazione presuppone una fase precedente di registrazione in cui il Requestor abbia potuto comunicare la Provider la sua volontà di ricevere queste notifiche.
Solicit-Response (output – input - fault)
In un'operazione di tipo solicit-responde il Web Service di sua iniziativa invia al Requestor un messaggio per sollecitarlo ad inviargli qualcosa. Anche in questo caso il tipo di operazione presuppone che il Provider conosca il Requestor a cui mandare il sollecito.

Le ultime due primitive di trasmissione sono comunemente utilizzate in sistemi che supportano il messaging asincrono. Poiché AXIS non supporta queste funzionalità queste due primitive di tramissione non sono supportate da AXIS.

Utilizzo di WSDL2Java

La maggior parte dei motori SOAP fornisce dei tool che consentono di generare lo stub e lo skeleton del servizio a partire dalla descrzione WSDL del servizio stesso. In particolare, AXIS fornisce il tool WSDL2Java che genera classi ed interfacce Java a partire da un documento WSDL.
AXIS fornisce anche il tool Java2WSDL che, a partire dall'interfaccia di una classe Java, fornisce il documento WSDL che descrive un Web Service implementato dall'interfaccia Java. Un esempio di utilizzo di tale tool lo si può avere sulla pagina di AXIS, dove sono elencati i servizi disponibili; per ogni servizio c'è un link al file WSDL di descrizione del servizio. Tale file viene generato dinamicamente invocando Java2WSDL.

Uno stub è una classe Java che espone un'interfaccia di programmazione Java-friendly che ricalca la lista delle operazioni messe a disposizione dal servizio. Grazie allo Stub il Requestor può invocare il servizio semplicemente limitandosi ad invocare i metodi locali esposti dallo stub, senza doversi preoccupare di interagire con AXIS Client per istanziare un oggetto Call ed invocare il metodo invoke().

Vediamo ora in dettaglio come opera WSDL2Java. Il tool prende in input il documento WSDL, ne fa il parsing e, per ogni elemento di WSDL, genera classi e/o interface Java. La tabella mostra le corrispondenze tra gli elementi di WSDL e le classi generate. E' da sottolineare la convenzione adottata da WSDL2Java per definire I nomi delle classi Java generate.

portType	Interfaccia Java con lo stesso nome dell'elemento portType e che espone un metodo per ogni operation. Nella terminologia di JAX-RPC questa interfaccia è detta Service Endpoint Interface (SEI)

binding	Una classe Java che implementa l'interfaccia SEI e costituisce lo Stub. Il nome di tale classe è il nome dell'elemento binding con il suffisso Stub.
service	Un'interfaccia Java con lo stesso nome dell'elemento service che consente un accesso type-safe alla SEI. Il metodo principale di questa interfaccia è quello che permette di ottenere un riferimento allo Stub che implementa la SEI.
service	Una classe Locator, che ha lo stesso nome dell'elemento Service con il suffisso Locator ed implementa l'interfaccia descritta al rigo precedente. La classe Locator funge da factory per generare istanze dello Stub.
types	Per ogni tipo XML definito nell'elemento types viene definita una classe Java Bean che lo rappresenta.

Bisogna prestare particolare attenzione a come WSDL2Java definisce la corrispondeza tra i package Java ed i namespace XML. Per default la corrispondenza di un namespace come http://www.skatestown.com/services sarebbe al package com.skatestown.www.services ma questa corrispondenza può essere modificata configurando opportunamente WSDL2Java. L'altra cosa da tenere presente è che ogni volta che viene utilizzato il tool tutte le classi ed interfacce vengono ricreate e le eventuali modifiche apportate vengoo cancellate.

L'architettura di un'applicazione che utilizza lo Stub generato da WSDL2Java per invocare un Web Service è la seguente:

- istanzia un oggetto ServiceLocator;
- invoca il metodo dell'interfaccia Service che restituisce un riferimento ad un oggetto Stub;
- invoca i metodi dell'interfaccia del servizio implementati dallo Stub.

Come si vede, lo Stub generato da WSDL2Java permette di utilizzare il servizio in maniera molto più semplice dell'invocazione dinamica effettuata tramite l'invocazione del metodo invoke() della classe Call.

WSDL2Java può essere utilizzato anche per generare lo skeleton lato provider. Specificando l'opzione –s sulla linea di comando si ottiene la generazione dei file WSDD contenenti i descrittori di deploy e undeploy e di un file SOAPBindingImpl.java che contiene il framework all'interno del quale si deve aggiungere il codice che implementa la logica dell'applicazione.
Il processo per fare il deploy di un Web Service a partire dalla descrizione WSDL del servizio è il seguente:

1. utilizzare WSDL2Java con l'opzione –s per costruire le classi Java del servizio;
2. editare la classe Impl costrutita dal tool aggiungendo la logic dell'applicazione;
3. compilare le classi generate e copiare I file .class nel classpath di AXIS;
4. utilizzare l'applet AdminClient per modificare il file server_config.wsdd e fare il deploy del servizio.

Esercitazione
Scrivere un'applicazione Java che implementa il Web Service CheckPrice a partire dalla descrizione WSDL che abbiamo fornito nelle precedenti esercitazioni (sia quella document-style che quella RPC-style). Scrivere un'applicazione client che invoca il Web Service utilizzando lo Stub generato da WSDL2Java.

Il codice dell'applicazione è contenuto nella directory degli esempi (pricecheck_rpc.rar e pricheck_doc.zip).

Esercitazione

Scrivere un file WSDL che descrive un Web Service che calcola il minimo in un array di stringhe. Scrivere un'applicazione Java che implementa il Web Service utilizzando il framework prodotto da WSDL2Java ed un'applicazione client che invoca il Web Service utilizzando lo Stub generato da WSDL2Java.

Il codice dell'applicazione è contenuto nella directory degli esempi (minimo_wsdl.zip).

Cenni su WSDL 2.0

Il lavoro di standardizzazione di WSDL è inziato all'interno della W3C nel 2001 ed ha portato a modifiche così sostanziali che si è preferito parlare di una major revision piuttosto che di un semplice miglioramento. WSDL 2.0 representa una standardizzazione formale di WSDL. Le principali novità introdotte sono state:

- Rimozione di alcune ambiguità (es. Eliminazione di notification e solicit-response);
- Adeguamento del funzionamento del meccanismo di import a quello di XML Schema;
- Utilizzo di nomi degli elementi più chiari (es. portType è diventato interface);
- Semplificazione di alcuni meccanismi (es. Sono state eliminate le part dei message);
- Aggiunta di nuove funzionalità come l'estensione delle interfacce;
- Eliminazione di alcune funzionalità come per esempio l'overloading delle operazioni.

Il seguente esempio mostra la descrizione in WSDl 2.0 di un servizio di un albergo per verificare la disponibilità di posti.

```
<?xml version="1.0" encoding="utf-8" ?>
<description
   xmlns="http://www.w3.org/ns/wsdl"
   targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
   xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
   xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
   xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
   xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   xmlns:wsdlx= "http://www.w3.org/ns/wsdl-extensions">
        <documentation>
           Questo documento descrive il Web Service di verifica disponibilità del GreatH Hotel
           Esempio estratto dal WSDL 2.0 Primer. Ulteriori dettagli sono disponibili all'URL
           http://greath.example.com/2004/reservation-documentation.html
        </documentation>

        <types>
           <xs:schema
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://greath.example.com/2004/schemas/resSvc"
                xmlns="http://greath.example.com/2004/schemas/resSvc">
              <xs:element name="checkAvailability" type="tCheckAvailability"/>
              <xs:complexType name="tCheckAvailability">
                   <xs:sequence>
                      <xs:element name="checkInDate" type="xs:date"/>
                      <xs:element name="checkOutDate" type="xs:date"/>
                      <xs:element name="roomType" type="xs:string"/>
                   </xs:sequence>
              </xs:complexType>
```

```
                <xs:element name="checkAvailabilityResponse" type="xs:double"/>
                <xs:element name="invalidDataError" type="xs:string"/>
        </xs:schema>
    </types>

    <interface name = "reservationInterface" >
        <fault name = "invalidDataFault" element = "ghns:invalidDataError"/>
        <operation name="opCheckAvailability"
            pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/iri"
            wsdlx:safe = "true">
            <input messageLabel="In" element="ghns:checkAvailability" />
            <output messageLabel="Out" element="ghns:checkAvailabilityResponse" />
            <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
        </operation>
    </interface>

    <binding name="reservationSOAPBinding"
        interface="tns:reservationInterface"
        type="http://www.w3.org/ns/wsdl/soap"
        wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
        <fault ref="tns:invalidDataFault" wsoap:code="soap:Sender"/>
        <operation ref="tns:opCheckAvailability"
            wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
    </binding>

    <service name="reservationService" interface="tns:reservationInterface">
        <endpoint name="reservationEndpoint"
            binding="tns:reservationSOAPBinding"
            address ="http://greath.example.com/2004/reservation"/>
    </service>
</description>
```

Come si può notare dall'esempio, WSDL 2.0 ha aggiunto molti nuovi tipi di operazioni, denominati pattern, che si vanno ad aggiungere ai tradizionali Request-Response e One-Way e che vengono definite esplicitamente tramite l'attributo pattern dell'elemento operation. Inoltre, nell'elemento binding è possibile descrivere l'implementazione di un'operazione facendo esplicito riferimento al MEP da adottare. Nel nostro esempio l'operazione opCheckAvalaibility adotta il pattern In-Out ed è implementato su SOAP utilizzando il MEP SOAP-Response.

Sicuramente una delle novità principali di WSDL 2.0 è l'introduzione del nuovo concetto di interface extension. Questo consente di definire un Web Service combinando operazioni fornite da altri Web Services e favorendo ancora di più il riutilizzo delle descrizioni. Ad esempio l'interfaccia del servizio dell'albergo potrebbe essere riscritta in questo modo.

```
<description ...>
...
   <interface name = "messageLogInterface" >
        <operation name="opLogMessage"
            pattern="http://www.w3.org/ns/wsdl/out-only">
                <output messageLabel="out" element="ghns:messageLog" />
```

```
        </operation>
    </interface>

    <interface name="reservationInterface" extends="tns:messageLogInterface" >
        <operation name="opCheckAvailability"
            pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/iri"
            wsdlx:safe = "true">
                <input messageLabel="In" element="ghns:checkAvailability" />
                <output messageLabel="Out" element="ghns:checkAvailabilityResponse" />
                <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
        </operation>
    </interface>
...
</description>
```

Infine sono state introdotte le property e le feature con finalità simili a quelle di SOAP. Le feature consentono di descrivere delle funzionalità astratte e di associarle a vari componenti del documento WSDL. La feature potrebbero riguardare, per esempio, l'autenticazione, l'autorizzazione e altri aspetti non funzionali.

Descrizioni non funzionali

WSDL fornisce una descrizione funzionale di un Web Service. Per quanto riguarda gli aspetti non funzionali non esiste una specifica sufficientemente matura come WSDL ma la maggior parte dwgli sforzi in questo ambito si sta concentrando sulla specifica WS-Policy e sulle specifiche ad essa collegate. In queste note daremo una breve descrizione, senza nessuna pretesa di completezza, di WS-Policy e del modo in cui si integra con WSDL in modo da fornire una descrizione completa del servizio.

La famiglia di specifiche WS-Policy è basata su tre elementi fondamentali: il framework, le assertions e gli attachment.

Il componente fondamentale di una policy e la policy assertion (definizione di una politica). La policy assertion è una dichiarazione esplicita riguado le richieste, le preferenze o le capacità di un Web Service o del suo ambiente operativo riguardo un determinato dominio. Le policy assertions possono descrivere livelli di QoS, come per esempio l'affidabilità del messaging, aspetti relativi alla sicurezza, come per esempio l'utilizzo di firme digitali, all'affidabilità o alla gestione delle transazioni. I provider di un servizio possono usare le Policy Assertion per descrivere in na maniera comprensibile ad una macchina i comportamenti del servizio; i requestor possono usare le Policy Assertion del servizio per generare codice che permette di interagire con il servizio assumendo i comportamenti previsti dal servizio stesso. Come successo con le librerie di scehemi XML, anche per le Policy si sta sviluppando una collezione di Policy Assertion che descrivono aspetti non funzionali di un Web Service e diverse specifiche WS-* sono corredate delle corrispondenti Policy Assertion che possono essere integrate nel file WSDL. Tra le principali specifiche descritte in termini di Policy ci sono:

- WS-Security,
- WS-ReliableMessaging,
- WS-AtomicTransaction.

Le policy assertions sono raggruppate insieme per formare una Policy. La specifica WS-Policy definisce il contenitore all'interno del quale vengono raccolte le policy assertions, le modalità per raggrupparle ed una serie di attributi standard per definire criteri di preferenza tra le varie policy assertions. Le policy vanno integrate all'interno del file WSDl come elementi di estendibilità, per esempio all'interno dell'elemento Binding. Una policy, come gruppo, può anche essere referenziata da altri Web Services o da una definizione WSDL utilizzando meccanismi XML standard. Ad esempio, si può definire un policy subject, che può essere un Web Service o un componente di un Web Service o una parte dell'ambiente operativo del Web Service che implementa quella policy. La specifica WS-PolicyAttachments descrive come si possono associare le policy ai policy subject.

WS-Policy

WS-Policy è stato originariamente pubblicato nel 2002 da un gruppo di aziende, tra cui Microsoft e IBM, ed è poi stata presentata alla W3C come una raccomandazione. Attualmente è arrivata alla versione 1.5.

WS-Policy definisce come raggruppare varie policy assertions insieme in modo da poter essere referenziate da altri componenti e integrate come elementi di estendibilità in un file WSDL. WS-Policy definisce il quadro di riferimento all'interno del quale definire policy nel mondo dei Web Services. Questo quadro di riferimento consiste di tre elementi:

- un elemento XML che funge da contenitore per le policy assertions;
- un insieme di elementi XML che descrivono come le policy assertions si combinano tra loro;
- un insieme di attributi XML globali che possono essere associati alle policy assertions.

Una caratteristica strana di WS-Policy che complica notevolmente le cose è che una policy può essere definita sia tramite un QName che tramite un'URI.

Gli operatori per combinare le policy assertions sono: All, ExactlyOne e Policy (che è un sinonimo di All). Usando questi operatori e annidando policy in altre policy è possibile costruire anche policy estremamente complesse. Per esempio, la policy seguente specifica che la policyAssertion wsam:Addressing (che descrive WS-Addressing) e la policyAssertion sp:TransportBinding (che fa parte di WS-Security e specifica che viene adottato un livello di sicurezza a livello di trasporto, es. https) sono entrambe adottate dal servizio.

```
<wsp:Policy name="PolicyExample"
TargetNamespace="http://www.skatestown.com/policies" >
        <wsp:All>
                <wsam:Addressing> … </wsam:Addressing>
                <sp:TransportBinding> …. </sp:TransportBinding>
        </wsp:All>
</wsp:Policy>
```

Questo elemento va incluso all'interno del binding che descrive l'implementazione concreta del servizio e prescrive che I requestor devono creare messaggi SOAP con i relativi header-block previsti dalle specifiche WS-Addressing e WS-Security e adottando il transport binding opportuno. E' possibile modificare la Policy prevedendo la possibilità di garantire la sicurezza o a livello di trasporto o a quello di messaggio.

```
<wsp:Policy name="PolicyExample"
TargetNamespace="http://www.skatestown.com/policies" >
        <wsp:All>
                <wsam:Addressing> … </wsam:Addressing>
```

```
            <wsp.ExactlyOne>
                  <sp:TransportBinding> …. </sp:TransportBinding>
                  <sp:AsymmetricBinding> …. </sp: AsymmetricBinding>
            </wsp:ExactlyOne>
      </wsp:All>
</wsp:Policy>
```

Gli attributi globali che possono essere assegnati a tutte le Policy Assertions sono Optional e Ignorable. L'attributo optional identifica un comportamento che il servizio supporta ma che i requestor possonoanche decidere di non adottare. Ad esempio, nel caso precedente potremmo aggiungere alla nostra policy che il servizio supporta una serializzazione MIME ottimizzata, come descritto nella specifica MTOM, ma i messaggi possono essere anche serializzati in maniera standard.

L'attributo Ignorable identifica una PolicyAssertion che viene adottata dal servizio ma che non si traduce in nessuna richiesta per il requestor. Questo significa che il client non avrà codice ad-hoc per supportare questo tipo di comportamento. Per esempio, il provider potrebbe comunicare che come sua politica tutti i messaggi ricevuti vengono loggati. Il requestor ha diritto di conoscere questo tipo di comportamento del provider ma non deve fare nulla per adeguarsi.

Il seguente frammento mostra la nostra Policy modificata con l'aggiunta delle Policy Assertion relative a MTOM e Logging.

```
<wsp:Policy name="PolicyExample"
TargetNamespace="http://www.skatestown.com/policies" >
      <wsp:All>
            <wsam:Addressing> … </wsam:Addressing>
            <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
            <log:Logging wsp:Ignorable="true"/>
            <wsp.ExactlyOne>
                  <sp:TransportBinding> …. </sp:TransportBinding>
                  <sp:AsymmetricBinding> …. </sp: AsymmetricBinding>
            </wsp:ExactlyOne>
      </wsp:All>
</wsp:Policy>
```

Le singole PolicyAssertion possono essere anche molto complesse e includere al loro interno delle Policy. Per esempio, di seguito vediamo la definizione della PolicyAssertion sp:TransportBinding inclusa nella specifica WS-SecurityPolicy.

```
<sp:TransportBinding>
  <Policy>
      <sp:TransportToken>
        <Policy>
            <sp:HttpsToken>
              <wsp:Policy/>
            </sp:HttpsToken>
        </Policy>
      </sp:TransportToken>
      <sp:AlgorithmSuite>
        <Policy>
```

```
                <sp:Basic256Rsa15/>
            </Policy>
        </sp:AlgorithmSuite>
        ….
    </Policy>
</sp:TransportBinding>
```

La specifica WS-Policy definisce un modello dati per definire la struttura di una Policy Assertion e descrive due modi per rappresentarle: una forma normalizzata ed una compatta (per maggiori dettagli consultare la specifica).

Molto spesso può capitare che una stessa azienda abbia varie Policy associate a Web Services differenti e quindi è necessario avere un meccanismo per poter referenziare una Policy. Le Policy sono identificate tramite un identificatore (attributo xml:Id oppure wsu:Id) oppure tramite un'URI. L'elemento utilizzato per referenziare una policy è PolicyReference. Se si vuole far riferimento alla Policy che abbiamo definito in precedenza in un altro punto del file WSDL è sufficiente scrivere

```
<wsp:PolicyReference URI="http://www.skatestown.com/policies/PolicyExample" >
        ….
</wsp:Policy>
```

Se, invece, utilizziamo l'ID "myPolicy" per identificare la Policy avremmo

```
<wsp:Policy wsu:Id="myPolicy" targetNamespace="http://www.skatestown.com/policies" >
        ….
</wsp:Policy>
```

Il valore dell'attributo wsu:Id è un tipo XML ID e l'URI che identifica questa Policy è "http://www.skatestown.com/policies#myPolicy".

Le Policy vengono inserite all'interno di un file WSDl come elementi di estendibilità e quindi possono comparire in tutti i punti dello schema WSDl in cui sono previste tali estensioni: a livello di binding, a livello di singola operation di un binding, a livello di service, a livello di port e a livello di un singolo message. La specifica PolicyAttachment prevede anche degli Attachment point per UDDI.

Riferimenti

1. S. Graham e alt., Building Web Services in Java II ed., SAMS, 2005, cap. 4 pag. 167-231 (tranne 210-213), cap. 5 pag. 253-258, 272-275
2. Web Services Description Language (WSDL) 1.1, (Marzo 2001)
3. Web Services Description Language (WSDL) 2.0 Part 0: Primer, Documento W3C, (Maggio 2007)
4. Web Services Description Language (WSDL) 2.0 Part 1: Core Language, Documento W3C, (Marzo 2006)
5. Web Services Policy Framework (WS-Policy) 1.2, Documento W3C, (Aprile 2006)
6. Web Services Policy Attachment (WS-PolicyAttachment) 1.2, Documento W3C, (Aprile 2006)

7. Web Services Policy Framework (WS-Policy) 1.5: Primer , Documento W3C, (Giugno 2007)
8. Web Services Policy Framework (WS-Policy) 1.5, Documento W3C, (Giugno 2007)
9. Web Services Policy Attachment (WS-PolicyAttachment) 1.5, Documento W3C, (Giugno 2007)

UNIVERSITA' DEGLI STUDI DI SALERNO

Corso di Laurea di Informatica
Insegnamento di Programmazione su Reti

L.O. 6: WSDL e UDDI

Localizzazione e Classficazione di Servizi

docente: Prof. Vincenzo Auletta
email: auletta@dia.unisa.it
sito web: http://www.dia.unisa.it/professori/auletta

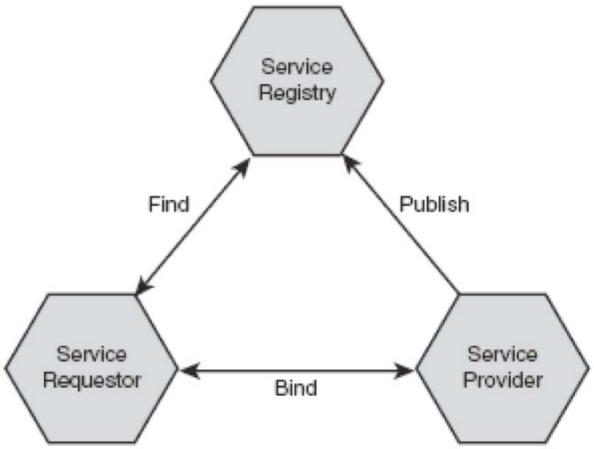
VER. 2.1 - 29/5/2009

Localizzazione e catagolazione di servizi tramite UDDI

Ruolo del registry all'interno di un'architettura SOA

Nelle precedenti unità didattiche abbiamo visto come sia possible descrivere un Web Service e come questa descrizione possa essere utilizzata da un Requestor per nvocare correttamente il servizio. In questa unità didattica dobbiamo affrontare il problema di come il Requestor possa recuperare il documento di descrizione del servizio. Il processo di scoperta del servizio stabilisce la relazione tra un Provider che pubblica un servizio ed un Requestor che richiede di utilizzarlo.

All'interno di un'architettura SOA questa relazione viene fornita dal Registry: il Provider pubblica sul Registry le informazioni sui servizi che ha reso disponibili; il Requestor cerca sul Registry le informazioni relative ai servizi che vuole invocare.



Nel caso specifico dei Web Services i Provider pubblicano sul Registry le informazioni sulle descrizioni WSDL dei loro servizi e i Requestor recuperano dal Registry tali descrizioni. Il Registry consente di effettuare ricerche complesse attraverso le quali il Requestor può recuperare descrizioni di servizi di cui non aveva nessuna conoscenza pregressa.

Il ruolo del Registry in un'architettura SOA può essere svolto usando diversi meccanismi: a meccanismi più semplici corrispondono tecniche di pubblicazione e ricerca limitate; a meccanismi più complessi corrispondono tecniche di pubblicazione e ricerca più sofisticate. Vediamo alcuni esempi.

Il meccanismo di pubblicazione che si può pensare è l'invio da parte del Provider del documento di descrizione del servizio direttamente ai Requestor. Questo meccanismo, però, implica che il Provider debba conoscere i suoi Requestor. Una soluzione alternative potrebbe essere la pubblicazione del documento sul sito Web del Provider. In questo caso, però, sono i Requestor che dovrebbero conoscere il Provider per poter recuperare le descrizioni dei servizi. Entrambe le soluzioni sono insufficienti perché lo scopo delle architetture SOA è di consentire la connessione tra Requestor e Provider che potrebbero anche non conoscersi.

Una soluzione completamente differente consiste nel prevedere un servizio centralizzato dove i Provider pubblicano le descrizioni dei servizi ed i Requestor ricercano informazioni sui servizi disponibili. In questo caso c'è da dover gestire questo servizio centralizzato l'unica cosa che Provider e Requestor devono conoscere è l'indirizzo del servizio centralizzato. Inoltre, questi servizi centralizzati possono fornire tecniche di categorizzazione e di ricerca avanzata.

Il servizio centralizzato può essere implementato in due maniere: un repository dove risiedono tutti i documenti di descrizione dei servizi pubblicati dai Provider oppure un Registry dove sono mantenute soltanto le informazioni su dove è possibile recuperare le informazioni relative ai servizi. La soluzione del repository è quella adottata do CORBA, mentre per i Web Services si è scelti di adottare la strategia del Registry. Il Registry è più efficiente perchè in caso sdi variazione di un documento di descrizione non è necessario ripubblicarlo sul servizio centralizzato.

UDDI (Universal Description, Discovery, and Integration)

L'iniziativa UDDI è stata annunziata per la prima volta nel 2000. Le primeversioni di UddI sono state sviluppate da un consorzio di aziende, tra cui Microsoft e IBM. Successivamente il compito dello sviluppo di UDDI è passato alla OASIS (Organization for the Advancement of Structured Information Standards (OASIS) ed è diventata una proposta di standard. Al momento la versione di UDDI più utilizzata, a cui noi faremo riferimento, è la 2.0 pubblicata nel 2003. OASIS ha già pubblicato la specifica di UDDI 3.0.

L'obiettivo di UDDI è di facilitare la scoperta dei servizi sia in fase di progettazione di un servizio che dinamicamente a tempo di esecuzione. UDDI implementa un registry. Nello scenario dei Web Services i Provider pubblicano su UDDI le informazioni relative a dove è possibile recuperare i documenti WSDL dei loro servizi ed i Requestor interrogano UDDI per scoprire dove è possibile recuperare i documenti WSDL.

Architettura di UDDI

I registry UDDI si dividono in pubblici e privati.

I registry pubblici sono gestiti da un numero limitato di aziende e sono ognuno la replica di tutti gli altri (i registry UDDI pubblici devono eseguire un protocollo di sincronizzazione per garantire la coerenza dei dati). Poiché tutti i registry pubbici sono identici, un Requestor può interrogarne uno qualunque ed essere sicuro di poter scoprire qualsiasi informazione pubblicata. Per esempio, interrogando il registry UDDI della IBM è possibile scoprire un servizio che è stato pubblicato sul registry UDDI della Microsoft. Sebbene inizialmente l'iniziativa UDDI era stata creata espressamente per fornire una sorta di visibilità globale ad aziende e servizi alla prova dei fatti questo servizio non ha riscontrato l'interesse sperato. La maggior parte delle aziende, però, non è interessata a pagare questo tipo di visibilità ma è più interessata ad essere visibile all'interno di una specifica nicchia (utenti della propria area geografica o che operano nello stesso ambito commerciale). Inoltre, pubblicare informazioni sui propri servizi a livello globale potrebbe compromettere la politica di sicurezza dell'azienda. Per questo motivo negli ultimi anni l'interesse si è concentrato soprattutto sui registry UDDI privati.

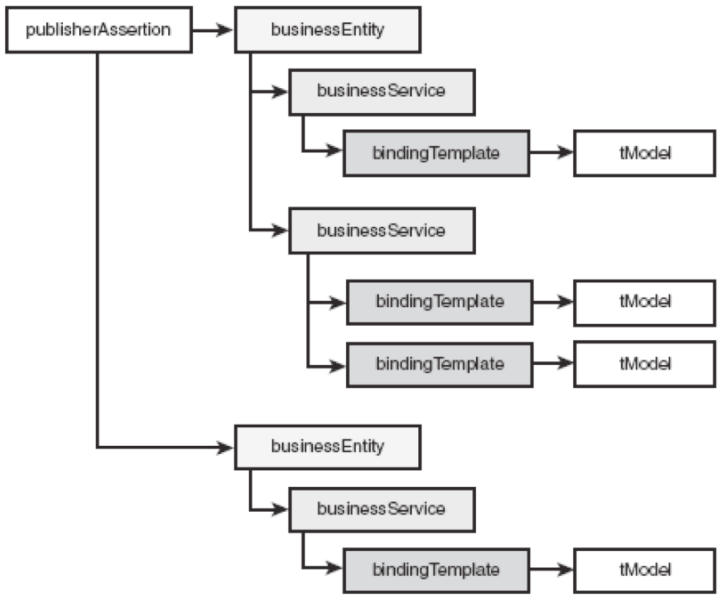
Un registry UDDI privato può essre utilizzato su Internet o su una Intranet nell'ambito d iuna comunità: solo i membri della comunità possono accedere alle informazioni del Registry. In questo modo le aziende che pubblicano le loro informazioni su un registry privato hanno una precisa idea di chi potrà accedere ed utilizzare quelle informazioni.

I tipi di dati UDDI

La specifica UDDI 2.0 prevede che le informazioni contenute nel registry siano modellate usando cinque tipi di dati:

businessEntity	Contiene le informazioni relative all'azienda o alla persona che ha pubblicato informazioni su dei servizi. Può contenere riferimenti ad uno o più servizi pubblicati dall'azienda.
businessService	Contiene la descrizione di un particolare servizio.
bindingTemplate	Contiene le informazioni tecniche relative ad un servizio, quail per esempio il punto di accesso al servizio e le specifiche per utilizzare il servizio.
tModel	Contiene le descrizioni delle specifiche di servizi o di tassonomie utilizzate per catalogare aziende e servizi. I tModel vengono utilizzati come base per definire le cosiddette "technical fingerprints".
publisherAssertion	Contiene le informazioni relative alle relazioni tra due businessEntity, che sono state definite da una delle due parti o da entrambe.

La figura mostra le relazioni tra questi cinque tipi di dati.



Tutti e cinque i tipi di dati sono definitii come elementi XML e la loro struttura fisica e logica è definita da degli schemi XML. Ogni singolo elemento è memorizzato nel registry indipendentemente dagli altri ed è identificato univocamente da un identificatore globale, detto key. Il registry UDDI assegna la key ad un'informazione la prima volta che questa viene memorizzata al suo interno e poi utilizza questa key per referenziare questa informazione all'interno di altri elementi. La key viene utilizzata da parte degli utenti del registry per referenziare l'elemento. Le key sono utilizzate per definire le relazioni di contenimento tra i vari tipi di dati: ogni elemento contiene al suo interno la key che identifica il suo unico elemento padre:

- ogni businessService contiene la key della businessEntity che la contiene;
- ogni bindigTemplate contiene la key della businessService che la contiene;
- ogni publisherAssertion contiene le key delle due businessEntity di cui definisce la relazione;

- ogni struttura contiene le key dei tModel che ne definiscono le technical fingerprints.

La key prende la forma di un Universally Unique ID3 (UUID) che è una stringa esadecimale che i registry UDDI generano in modo da garantire che sia univoca.

Illustriamo ora con un esempio la sintassi di ciascun tipo di dati.

BusinessEntity

La struttura businessEntity rappresenta tutte le informazioni disponibili riguardo un azienda o di un'entità che pubblica informazioni riguardo le entità e i servizi. Lo schema XML che definisce la strutttura di una BusinessEntity è il seguente.

```
<element name="businessEntity" type="uddi:businessEntity" />
   <complexType name="businessEntity">
        <sequence>
            <element ref="uddi:discoveryURLs" minOccurs="0" />
            <element ref="uddi:name" maxOccurs="unbounded" />
            <element ref="uddi:description" minOccurs="0" maxOccurs="unbounded" />
            <element ref="uddi:contacts" minOccurs="0" />
            <element ref="uddi:businessServices" minOccurs="0" />
            <element ref="uddi:identifierBag" minOccurs="0" />
            <element ref="uddi:categoryBag" minOccurs="0" />
        </sequence>
        <attribute name="businessKey" type="uddi:businessKey" use="required" />
        <attribute name="operator" type="string" use="optional" />
        <attribute name="authorizedName" type="string" use="optional" />
   </complexType>
</element>
```

La seguente tabella mostra il significato degli elementi principali di businessEntity.

attributo **businessKey**	contiene l'identificatore globale UUID che identifica questa istanza
attributo **authorizedName**	È il nome dell'utente autorizzato a pubblicare e modificare i dati contenuti in questa entità.
atributo **operator**	È il nome del server UDDI su cui è stata pubblicata l'entità ed è l'unico server autorizzato a modficare i suoi dati.
name	Elemento obbligatorio contenente il nome con cui l'entità viene referenziata.
description	Elemento opzionale contenente una descrizione dell'azienda
contacts	Elemento opzionale contenente una sequenza di elementi contact. Ogni elemento contact contiene i campi useType, persnName, phone, email, address
businessService	Elemento opzionale che contiene la descrizione di un servizio offerto dall'azienda.
categoryBag	Elemento opzionale che contiene una lista di coppie nome-valore utilizzate per identificare l'entità risptto a specifiche tassonomie. Questi nomi possono essere usati dai Requestor per ricerche avanzate sul registry.
IdentifierBag	Elemento opzionale che contiene una lista di coppie nome-valore

	utilizzate per identificare l'entità. Questi nomi possono essere usati dai Requestor per ricerche sul registry.

BusinessService

La struttura businessService rappresenta la descrizione di un servizio. Ogni struttura businessService è figlia di una businessEntity che pubblica il servizio e deve contenere la key che identifica la struttura businessEntity padre. Lo schema XML che definisce la strutttura di una BusinessService è il seguente.

```
<element name="businessService" type="uddi:businessService" />
    <complexType name="businessService">
        <sequence>
            <element ref="uddi:name" minOccurs="0" maxOccurs="unbounded" />
            <element ref="uddi:description" minOccurs="0" maxOccurs="unbounded" />
            <element ref="uddi:bindingTemplates" minOccurs="0" />
            <element ref="uddi:categoryBag" minOccurs="0" />
        </sequence>
        <attribute name="serviceKey" type="uddi:serviceKey" use="required" />
        <attribute name="businessKey" type="uddi:businessKey" use="optional" />
    </complexType>
</element>
```

La seguente tabella mostra il significato degli elementi principali di businessService.

attributo **businessKey**	contiene l'identificatore globale UUID della struttura businessEntity padre di questo elemento.
attributo **serviceKey**	contiene l'identificatore globale UUID della struttura businessService.
name	Elemento obbligatorio contenente il nome con cui l'entità viene referenziata.
description	Elemento opzionale contenente una descrizione del servizio.
bindingTemplates	Elemento opzionale che contiene la descrizione tecnica relativa al servizio.
categoryBag	Elemento opzionale che contiene una lista di coppie nome-valore utilizzate per identificare il servizio risptto a specifiche tassonomie. Questi nomi possono essere usati dai Requestor per ricerche avanzate sul registry.
identifierBag	Elemento opzionale che contiene una lista di coppie nome-valore utilizzate per identificare il servizio. Questi nomi possono essere usati dai Requestor per ricerche sul registry.

BindingTemplate

Le strutture BindingTemplates contengono le descrizioni tecniche dei Technical dei businessService. Queste strutture permettono di determinare il punto di accesso al servizio e le specifiche tecniche necessarie per invocarlo.
Ogni struttura BindingTemplates appartiene ad una BusinessService e deve contenere la key del suo elemento padre. Lo schema XML che definisce la strutttura di una BusinessService è il seguente.

```
<element name="bindingTemplate" type="uddi:bindingTemplate" />
   <complexType name="bindingTemplate">
      <sequence>
         <element ref="uddi:description" minOccurs="0" maxOccurs="unbounded" />
         <choice>
            <element ref="uddi:accessPoint" />
            <element ref="uddi:hostingRedirector" />
         </choice>
         <element ref="uddi:tModelInstanceDetails" />
      </sequence>
      <attribute name="serviceKey" type="uddi:serviceKey" use="optional" />
      <attribute name="bindingKey" type="uddi:bindingKey" use="required" />
   </complexType>
</element>
```

La seguente tabella mostra il significato degli elementi principali di businessService.

attributo **serviceKey**	contiene l'identificatore globale UUID della struttura serviceEntity padre di questo elemento.
attributo **bindingKey**	contiene l'identificatore globale UUID della struttura bindingTemplate.
description	Elemento opzionale contenente una descrizione del servizio.
accessPoint	Elemento obbligatorio che contiene le informazioni su dove è disponibile il servizio. Non viene fatta nessuna ipotesi sul tipo di indirizzo (può essere un URL, un numero di telefono, un indirizzo email, ecc.).
hostingRedirector	Elemento che può sostituire l'accessPoint. Contiene una bindingKey che fa riferimento ad un'altra BindingTemplates che contiene l'accessPoint del servizio.
tModelInstanceDetails	Struttura che contiene una lista di strutture tModelInstanceInfo e che definisce la techincal fingerprint del servizio. Ogni struttura tModelInstanceInfo contiene una tModelKey, che fa riferimento al tModel della specifica tecnica a cui aderisce il servizio, una descrizione ed una struttura opzionale instanceDetails. La struttura InstanceDetails contiene un elemento OverviewDoc, con il riferimento ad un documento che descrive la specifica tecnica, ed un elemento opzionale InstanceParms che contiene i settino di eventuali parametri.

tModel

Uno degli obbiettivi di UDDI è di rendere disponibili descrizioni di servizi che possano essere utilizzate dai Requestor per capire come invocarli. Per fare ciò, c'è bisogno di un modo per inserire all'interno della descrizione del servizio informazioni su come il servizio opera e a quali specifiche tecniche è conforme. Il ruolo delle strutture tModel all'interno di UDDI è esattamente quello di definire la conformità di un elemento rispetto ad un concetto, inteso nell'accezione più ampia possible.

La struttura tModel è essenzialmente un metadato identificato tramite una key e la sua funzione è definire un sistema di riferimento all'interno del Registry UDDI basato sull'astrazione. Quindi i tModel possono essere usati in riferimento a qualsiasi tipo di informazione. All'interno di UDDI ci

sono due utilizzi principali dei tModel: fornire una specifica tecnica a cui fa riferimento l'implemenazione di un servizio (il riferimento al tModel compare nelle strutture tModelInstanceInfo del BindingTemplate); definire un namespace all'interno del quale definire relazioni tra nomi e valori (il riferimento al tModel che definisce un namespace compare all'interno degli elementi KeyedReference contenuti negli IdentifierBag e CategoryBag).

Lo schema XML che definisce la strutttura di un tModel è il seguente.

```
<element name="tModel" type="uddi:tModel" />
   <complexType name="tModel">
      <sequence>
         <element ref="uddi:name" />
         <element ref="uddi:description" minOccurs="0" maxOccurs="unbounded" />
         <element ref="uddi:overviewDoc" minOccurs="0" />
         <element ref="uddi:identifierBag" minOccurs="0" />
         <element ref="uddi:categoryBag" minOccurs="0" />
      </sequence>
      <attribute name="tModelKey" type="uddi:tModelKey" use="required" />
      <attribute name="operator" type="string" use="optional" />
      <attribute name="authorizedName" type="string" use="optional" />
   </complexType>
</element>
```

La seguente tabella mostra il significato degli elementi principali di businessService.

attributo **tModelKey**	contiene l'identificatore globale UUID della struttura tModel.
attributo **authorizedName**	È il nome dell'utente autorizzato a pubblicare e modificare i dati contenuti in questo tModel.
atributo **operator**	È il nome del server UDDI su cui è stata pubblicato il tModel ed è l'unico server autorizzato a modficare i suoi dati.
name	Elemento obbligatorio contenente il nome con cui il tModel viene referenziato.
description	Elemento opzionale contenente una descrizione del tModel.
overviewDoc	Elemento opzionale contenente riferimenti a informazioni descrittive relative al tModel. Il suo utilizzo è analogo all'elemento overviewDoc di bindingTemplate.
categoryBag	Elemento opzionale che contiene una lista di coppie nome-valore utilizzate per identificare il tModel rispetto a specifiche tassonomie. Questi nomi possono essere usati dai Requestor per ricerche avanzate sul registry.
identifierBag	Elemento opzionale che contiene una lista di coppie nome-valore utilizzate per identificare il tModel. Questi nomi possono essere usati dai Requestor per ricerche sul registry.

publisherAssertions

Permette di definire relazioni tra BusinessEntity.
Lo schema XML che definisce la strutttura di una PublisherAssertion è il seguente.

```
<element name="publisherAssertion" type="uddi:publisherAssertion" />
   <complexType name="publisherAssertion">
```

```
        <sequence>
            <element ref="uddi:fromKey" />
            <element ref="uddi:toKey" />
            <element ref="uddi:keyedReference" />
        </sequence>
    </complexType>
</element>
```

La seguente tabella mostra il significato degli elementi principali di PublisherAssertion.

fromKey	Elemento obbligatorio che contiene il riferimento alla prima entità BusinessEntity coinvola nella relazione.
toKey	Elemento obbligatorio che contiene il riferimento alla seconda entità BusinessEntity coinvola nella relazione.
keyedReference	Elemento obbligatorio che definisce il tipo di relazione esistente tra le due entità.

Esercitazione

Scrivere le strututre per pubblicare sul Registry UDDI le informazioni relative alla SkatesTown ed al suo servizio di sottomissione degliordini di acquisto tramite un Web Service. Fare riferimento al documento WSDL di descrizione del servizio realizzato nella precedente unità didattica.

```
<businessEntity businessKey="54438690-573E-11D8-B936-000629DC0A53">
    <name>SkatesTown</name>
        <description>UDDI businessEntity per SkatesTown.</description>
        <contacts>
            <contact useType="informazioni tecniche">
                <description xml:lang="it">Responsabile supporto tecnico</description>
                <personName>Mario Rossi</personName>
                <phone useType="Main Office">39.089.123456</phone>
                <email useType="CTO">mario.rossi@SkatesTown.com</email>
                <email useType="General Information">info@SkatesTown.com</email>
                <address useType="sede"sortCode="10001">
                    <addressLine>via Roma 17</addressLine>
                    <addressLine>Salerno</addressLine>
                    <addressLine>Italia</addressLine>
                </address>
            </contact>
            <contact useType="informazioni commerciali">
                <description xml:lang="it">Responsabile vendite</description>
                <personName>Anna Fusco</personName>
                <phone useType="Main Office">39.089.2223334</phone>
                <email useType="VP Sales">anna.fusco@SkatesTown.com</email>
                <email useType="Sales Information">vendite@SkatesTown.com</email>
                <address useType="sedservizio di sottomissione ordini basato su
SOAP.</description>
                    <addressLine>via Roma 17</addressLine>
                    <addressLine>Salerno</addressLine>
                    <addressLine>Italia</addressLine>
```

```
                </address>
            </contact>
        </contacts>
        <businessServices>
            <businessService serivceKey="4C379407-3E1E-DC97-B1C7-F68597DA4ADB"
                             businessKey="55BB30D8-565A-4EF9-BA2E-83118AED644D">
                <name>Sottomissione ordini</name>
                <description>Servizio di sottomissione ordini per la SkatesTown.</description>
                <bindingTemplates>
                    <bindingTemplate bindingKey="3F7ABC88-14F2-AEF2-41AE-F86E52908A11"
                                     serviceKey="4C379407-3E1E-DC97-B1C7-F68597DA4ADB">
                        <description>
                            servizio di sottomissione ordini basato su SOAP
                        </description>
                        <accessPoint URLtype="http">
                            http://www.skatestown.com/services/poSubmission
                        </accessPoint>
                        <tModelInstanceDetails>
                            <tModelInstanceInfo
                                tModelKey="uuid:7B581129-7926-5202-AB17-74A234F21BA5">
                                <description>
                                    riferimento al tModel della definizione dell'interfaccia del Web
Service
                                </description>
                            </tModelInstanceInfo>
                        </tModelInstanceDetails>
                    </bindingTemplate>
                </bindingTemplates>
            </BusinessService>
        </BusinessServices>
</businessEntity>

<tModel tModelKey="uuid:7B581129-7926-5202-AB17-74A234F21BA5">
    <name>servizio di sottomissione ordini di acquisto</name>
    <description xml:lang="it">
        Definizione dell'interfacci del Web Service per la sottomissione degli ordini di acquisto.
    </description>
    <overviewDoc>
        <description xml:lang="it">
            Riferimento al documento WSDL che contiene la definizione dell'interfaccia del servizio
per la sottomissione degli ordini di acquisto.
        </description>
        <overviewURL>
            http://www.skatestown.com/services/poSubmission.wsdl
        </overviewURL>
    </overviewDoc>
</tModel>
```

Categorizzazione e Identificazione delle informazioni

Come abbiamo in precedenza, uno degli obbiettivi di UDDI è consentire la scoperta di servizi anche se non si hanno informazioni su di esso. Questo significa consentire ai Requestor di effettuare ricerche non solo in base al nome del servizio o dell'azeinda che lo fornisce ma anche in base a criteri di ricerca avanzata come per posizione geografica, categoria merceologica, tipi di attività, tipo di specifica tecnica a cui è conforme il servizio, ecc. Per consentire questo tipo di ricerche le informazioni presenti all'interno del Registry devono essere arricchite con dettagli relativi alla categorizzazione ed alla classificazione di aziende e servizi.

Lacategorizzazione è il processo di creazione di una gerarchia di categorie. La classificazione, invece, è il processo di assegnamento di aziende a servizi a queste categorie. Esempi di categorizzazione sono utilizzati in tutte le biblioteche dove i libri sono divisi per genere; la classificazione, in quel caso, consiste nel definire per ogni libro il genere a cui appartiene.

UDDI definisce tra criteri di categorizzazione che sono:

- Il North American Industry Classification System (NAICS) che viene utilizzato per classificare le aziende nord-americane e canadesi in base al tipo di attività svolta;
- L' Universal Standard Products and Services Classification (UNSPSC) definito da una agenzia delle Nazioni Unite per classificare prodotti e servizi;
- Lo standard ISO 3166 per classificare I luoghi geografici.

Ognuna di queste tassonomie viene identificata e referenziata utilizzando un tModel predefinito. Le tModelKey per referenziare questi tModel sono:

NAICS	uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2
UNSPSC	uuid:CD153257-086A-4237-B336-6BDCBDCC6634
ISO 3166	uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88

Le informazioni sulla classificazione dei aziende e servizi rispetto a queste tre tassonomie sono contenute nel CategoryBag. Un CategoryBag contiene una collezione di strutture KeyedReference, dove ogni KeyedREference contiene gli elementi tModelKey, keyName e keyValue e rappresenta un'associazione nome-valore definite rispetto ad una tassonomia. Per esempio, per specificare che la SkatesTown opera a New York bisognerebbe aggiungere alla struttura BusinessEntity il seguente elemento CategoryBag:

```
<categoryBag>
        <keyedReference keyName="New York" keyValue="US-NY"
                tModelKey="uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88"/>
</categoryBag>
```

E' posibile aggiungere al Registry UDDI altri criteri di categorizzazione. Per farlo bisogna definire un nuovo tModel che identifica questa tassonomia e che possa essere referenziato all'interno dei KeyedReference presenti nei CategoryBag.

Esercitazione

Scrivere il tModel che identifica la tassonomia di Yahoo! E fornire un esempio di CategoryBag che fa riferimento a questa tassonomia.

```
<tModel tModelKey="uuid:3D4EC875-E54F-4D8D-9CBF-346D48BCAD9C">
        <name>Yahoo! Business Taxonomy</name>
        <description xml:lang="en">Yahoo! Business Taxonomy</description>
```

```
        <categoryBag>
                <keyedReference keyName=”Yahoo! Category” keyValue=”categorization”
                        tModelKey=”uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4”/>
</categoryBag>
</tModel>

<categoryBag>
        <keyedReference keyName=”Sporting and Athletic Goods Manufacturing”
                keyValue=”33992”
                tModelKey=”uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2”/>
        <keyedReference keyName=”New York” keyValue=”US-NY”
                tModelKey=”uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88”/>
        <keyedReference keyName=”Yahoo Business Taxonomy”
        keyValue=”Business_and_Economy/Shopping_and_Services/Sports/Skateboarding/Deck_a
nd_Truck_Makers/”
                tModelKey=”uuid:3D4EC875-E54F-4D8D-9CBF-346D48BCAD9C”/>
</categoryBag>
```

Oltre a specificare informazioni di classificazione è possible aggiungere informazioni di identificazione. Queste informazioni sono specificate rispetto ad uno schema di identificazione, come potrebbe essere il codice fiscale, che è descritto da un tModel. UDDI non ha schemi di identificazione predefiniti. Il seguente esempio mostra la definizione di uno schema di identificazione basato sul codice fiscale e la definizione di un IdentifierBag che identifica un’entità tramite il suo codice fiscale,

```
<tModel tModelKey=”uuid:F2390501-A240-4470-8A5A-6088EE5B1A14”>
        <name>codiceFiscale</name>
        <description xml:lang=”it”>codice fiscale</description>
        <categoryBag>
            <keyedReference keyName=”identificatori univoci per persone italiane”
                keyValue=”identificatori”
                tModelKey=”uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4”/>
        </categoryBag>
</tModel>

<identifierBag>
        <keyedReference keyName=”codiceFiscale” keyValue=”PTTBCN07B23C345J”
                tModelKey=”uuid:F2390501-A240-4470-8A5A-6088EE5B1A14”/>
</identifierBag>
```

Esercitazione

Aggiungere alle strutture della SkatesTown informazioni di categorizzazione ed identificazione.

```
<businessEntity businessKey="54438690-573E-11D8-B936-000629DC0A53">
   <name>SkatesTown</name>
        <description>UDDI businessEntity per SkatesTown.</description>
        <contacts>
            <contact useType="informazioni tecniche">
                <description xml:lang="it">Responsabile supporto tecnico</description>
                <personName>Mario Rossi</personName>
```

```
            <phone useType="Main Office">39.089.123456</phone>
            <email useType="CTO">mario.rossi@SkatesTown.com</email>
            <email useType="General Information">info@SkatesTown.com</email>
            <address useType="sede"sortCode="10001">
               <addressLine>via Roma 17</addressLine>
               <addressLine>Salerno</addressLine>
               <addressLine>Italia</addressLine>
            </address>
         </contact>
         <contact useType="informazioni commerciali">
            <description xml:lang="it">Responsabile vendite</description>
            <personName>Anna Fusco</personName>
            <phone useType="Main Office">39.089.2223334</phone>
            <email useType="VP Sales">anna.fusco@SkatesTown.com</email>
            <email useType="Sales Information">vendite@SkatesTown.com</email>
            <address useType="sedservizio di sottomissione ordini basato su
SOAP.</description>
               <addressLine>via Roma 17</addressLine>
               <addressLine>Salerno</addressLine>
               <addressLine>Italia</addressLine>
            </address>
         </contact>
      </contacts>
      <businessServices>
         <businessService serivceKey="4C379407-3E1E-DC97-B1C7-F68597DA4ADB"
                          businessKey="55BB30D8-565A-4EF9-BA2E-83118AED644D">
            <name>Sottomissione ordini</name>
            <description>Servizio di sottomissione ordini per la SkatesTown.</description>
            <bindingTemplates>
               <bindingTemplate bindingKey="3F7ABC88-14F2-AEF2-41AE-F86E52908A11"
                                serviceKey="4C379407-3E1E-DC97-B1C7-F68597DA4ADB">
                  <description>
                     servizio di sottomissione ordini basato su SOAP
                  </description>
                  <accessPoint URLtype="http">
                     http://www.skatestown.com/services/poSubmission
                  </accessPoint>
                  <tModelInstanceDetails>
                     <tModelInstanceInfo
                        tModelKey="uuid:7B581129-7926-5202-AB17-74A234F21BA5">
                        <description>
                           riferimento al tModel della definizione dell'interfaccia del Web
Service
                        </description>
                     </tModelInstanceInfo>
                  </tModelInstanceDetails>
               </bindingTemplate>
            </bindingTemplates>
            <categoryBag>
               <keyedReference keyName=”Sports equipment and accessories”
                  keyValue=”49221500”
```

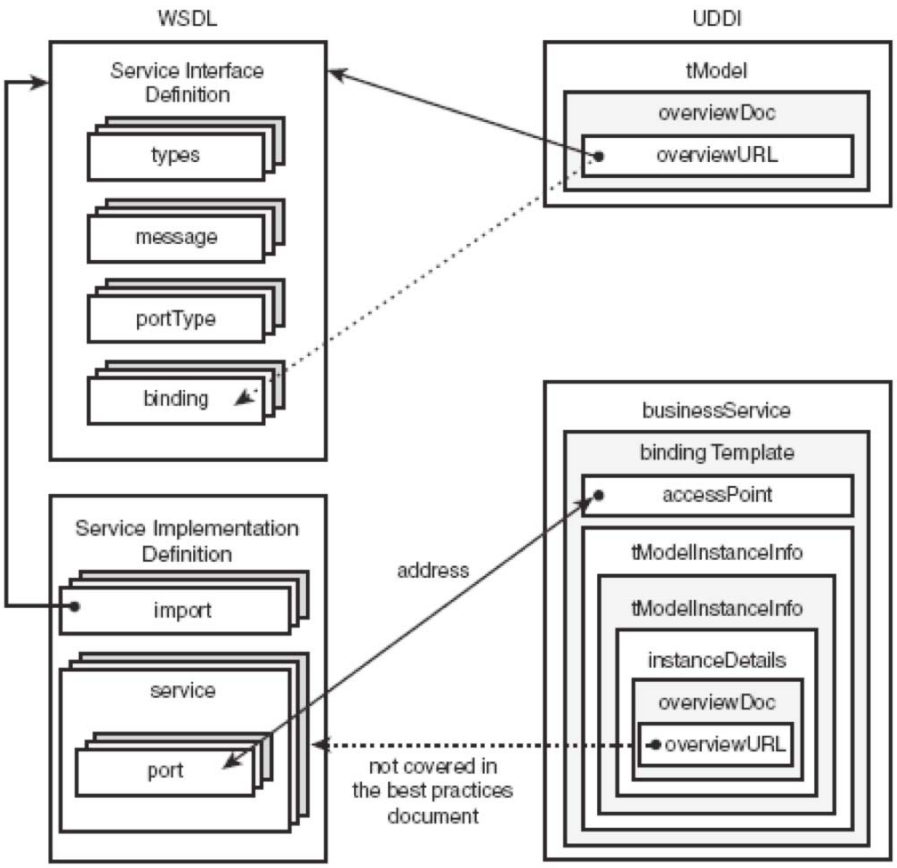
```
                        tModelKey=”uuid:CD153257-086A-4237-B336-6BDCBDCC6634”/>
                </categoryBag>
            </BusinessService>
        </BusinessServices>
        <identifierBag>
            <keyedReference keyName=”DUNS”  keyValue=”00-111-1111”
                    tModelKey=”uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823”/>
        </identifierBag>
        <categoryBag>
            <keyedReference keyName=”Sporting and Athletic Goods Manufacturing”
                    keyValue=”33992”
                    tModelKey=”uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2”/>
            <keyedReference keyName=”New York” keyValue=”US-NY”
                    tModelKey=”uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88”/>
        </categoryBag>
</businessEntity>

<tModel tModelKey="uuid:7B581129-7926-5202-AB17-74A234F21BA5">
    <name>servizio di sottomissione ordini di acquisto</name>
    <description xml:lang="it">
        Definizione dell'interfacci del Web Service per la sottomissione degli ordini di acquisto.
    </description>
    <overviewDoc>
        <description xml:lang="it">
            Riferimento al documento WSDL che contiene la definizione dell'interfaccia del servizio
per la sottomissione degli ordini di acquisto.
        </description>
        <overviewURL>
            http://www.skatestown.com/services/poSubmission.wsdl
        </overviewURL>
    </overviewDoc>
</tModel>
```

Integrazione tra WSDL e UDDI

Nell’architettura dei Web Services il Registry UDDI deve consentire di pubblicare e ricercare le descrizioni dei servizi fornite dai documenti WSDL. Il gruppo che coordina l osviluppo di UDDI ha pubblicato una serie di raccomandazioni (Best Practice) sul modo in cui si dovrebbe pubblicare il documento WSDL di descrizione di un servizio.

Queste raccomandazioni fanno riferimento ad una organizzazione del documento WSDL in due parti: la definizione dell’interfaccia del servizio e la definizione dell’implementazione del servizio.

La figura mostra come gli lementi di un documento WSDL vengono mappati sui tipi di dati UDDI.



Poichè la definizione dell'interfaccia rappresenta una definizione riutilizzabile che può essere riciclata per varie implementazioni, allora può essere pubblicata come un tModel. Questo tModel può essere visto come una specifica tecnica a cui le varie implementazioni si conformano. La definizione dell'implementazione del servizio descrive un'istanza del servizio che corrisponde ad un BusinessService. In particolare il BindingTemplate che fornisce le informazioni tecniche del servizio specifica nel campo accessPoint il valore address specificato nell'elemento port di WSDL. Sebbene non specificato nelle Best Practice di UDDI, sarebbe opportuno che nel campo overviewURL contenuto all'interno del tModelInstanceInfo venisse indicato l'URL del documento WSDL che contiene la definizione dell'emplementazione del servizio.

Utilizzare i servizi di un Registry UDDI

Un Registry UDDI è esso stesso implementato come un Web Service e quindi tutte le interazioni tra il Registry e gli utenti sono basate sullo scambio di messaggi SOAP. L'API di programmazione di UDDI è divisa in due parti: Inquiry API e UDDI Publication API.

La maggior parte dei registry UDDI richiede all'utente di registrarsi prima di poter pubblicare delle informazioni. Il processo di registrazione, in genere basato su una user ID ed una password, fornisce un account che può essere utilizzato o per pubblicare nuovi dati oppure per modificare (o cancellare) quelli esistenti. Il servizio UDDI prevede che i dati possano essere modificati solo sul server su cui sono stati pubblicati (controllo sull'attributo operator di BusinessEntity) e dall'utente che li ha inseriti (controllo sula valore dell'attributo authorizedName di BusinessEntity). La Pubblication API di UDDI fornisce il supporto per creare, aggiornare e cancellare ciascuno dei cinque tipi di dati di UDDI. Per poter utilizzare le funzioni della Pubblication API bisogna prima invocare il servizio get_authToken per ottenere un Authentication Token. Un Authentication Token è un oggetto opaco che contiene al suo interno informazioni relative all'utente ed all'istante in cui è

stato rilasciato il Token. I Token hanno un periodo di validità, terminato il quale devono essere rinnovati. Per invocare tutte le altre operazioni della Pubblication API è necessario fornire un Token valido.

Le operazioni della Pubblishing API sono:

- save_binding e delete_binding (BindingTemplate),
- save_business e delete_business (BusinessEntity),
- save_service e delete_service (BusinessService),
- save_tModel e delete_tModel (tModeli).

Le operazioni save_XXX sono utilizzate per creare nuovi elementi o aggiornare quelli esistenti; le operazioni dolete_XXX sono utilizzate per rimuovere un eliminare un elemento dal Registry UDDI.

L'Inquiry API viene utilizzata per effettuare ricerche di informazioni nel Registry UDDI. A differenza della Pubblichino API, l'utilizzo di questa API è libero per tutti. Quindi, per invocare le operazioni dell'Inquiry API non c'è bisogno di fornire l'Authorization Token. L'API contiene due tipi di operazioni: le operazioni find, che permettono di definire una ricerca sul Registry e restituisce una lista di UDDI keys corrispondenti al criterio di ricerca specificato, e le operazioni get Get, che prendono In input una key e restituisce il contenuto dell'elemento referenziato.

Le operazioni della Inquiry API sono:

- find_binding e get_bindingDetail (BindingTemplate),
- find_business e get_businessDetail (BusinessEntity),
- find_service e get_serviceDetail (BusinessService),
- find_tModel e get_tModelDetail (tModel).

Esistono diverse interfacce Java per costruire applicazioni client che utilizzano i servizi di un Registry UDDI. Le principali alternative sono:

- *UDDI4J* — La UDDI for Java (UDDI4J) API fornisce un interfaccia di programmazione specifica per UDDI. Questa API fornisce una classe Java per rappresentare ogni tipo di dato di UDDI e c'è una classe UDDIPRoxy che espone un'interfaccia I cui metodi corrispondono ai Web sErvices implementati dal REgistry UDDI.
- *JAXR* — La Java API for XML Registries (JAXR) fornisce un API che può essere utilizzata per costruire applicazioni Java che utilizzano registry basati su standard aperti, come ebXML e UDDI;
- *AXIOM* — L'API di AXIOM e le API Client di Axis2 possono essere usate per gestire direttamente l'interazione con i Web Services esposti dal Registry UDDI.

Riferimenti

1. S. Graham e alt., Building Web Services in Java II ed., SAMS, 2005, cap. 6 pag. 307-343
2. UDDI Version 2.04 API Specification, documento OASIS, (Luglio 2002)
3. UDDI Version 2.04 Data Stucture Reference, documento OASIS, (Luglio 2002)
4. Technical Note on Using WSDL in a UDDI Registry, Version 2.0.2, documento OASIS, (2004)
5. Providing A Taxonomy For Use In UDDI Version2, UDDI Working Draft Technical Note, (Luglio 2001)