UNIVERSITA' DEGLI STUDI DI MODENA
E REGGIO EMILIA
Facoltà di Ingegneria – Sede di Modena

Corso di Laurea in Ingegneria Informatica

Il protocollo SOAP e la sicurezza.

Relatore:

Chiar.mo Prof.
Sonia Bergamaschi

Tesi di laurea di:

Feverati Chiara

Correlatore:
Maurizio Vincini

Anno Accademico 2002-2003

Parole Chiave:

- RPC
- protocollo applicativo SOAP
- web services e liguaggio WSDL
- sicurezza nei web services
- WS-Security
- Apache SOAP
  Microsoft SOAP Toolkit

Ringraziamenti:

Indice

Indice delle figure

Introduzione e obiettivi

Gli obiettivi di questa tesi sono stati i seguenti:

- Analisi dei Servizi Web.
  Un servizio Web è un elemento funzionale disponibile in un punto qualsiasi di Internet e accessibile tramite protocolli comuni come HTTP o SMTP. Il ricorso a XML, come linguaggio di rappresentazione dei dati per tutti i protocolli e le tecnologie dei servizi web disponibili, realizza di fatto la loro pressoché completa interoperabilità. Il suo impiego nel trasporto dei dati consente di eliminare tutte le specificità dovute alla rete, al sistema operativo o alla piattaforma. L'utilizzatore non è legato in modo diretto a un determinato servizio Web, la sua interfaccia può essere modificata nel tempo senza per questo compromettere la capacità del client di interagire con il servizio stesso. I servizi Web permettono al client di invocare procedure, funzioni e metodi su oggetti remoti, facendo uso di un protocollo basato su XML. Le procedure remote dispongono di parametri sia d'ingresso (input) sia d'uscita (output), che il servizio Web deve supportare.
- Studio delle principali tecnologie per i servizi Web: il protocollo SOAP, il linguaggio WSDL e i servizi UDDI. Ognuna di queste tecnologie è solo evolutiva. Ciascuna costituisce lo standard di base per il miglioramento conseguente dei servizi Web, nella loro descrizione o scoperta, ai fini dell'integrazione automatica e senza problemi di vari componenti: l'obiettivo è la creazione di un software in grado di scoprire, accedere, integrare e richiamare in modo dinamico e senza alcun intervento umano, nuovi servizi offerti da aziende non note. Un progetto di questo genere richiede la presenza combinata delle tecnologie SOAP. WSDL e UDDI per creare un'infrastruttura standard in grado di supportare il business dinamico del futuro. La relazione esistente tra queste tre tecnologie è la seguente: (immagine)
  Il client di un servizio Web deve rintracciare un'applicazione o un elemento funzionale di un programma che si trova nella rete. A questo scopo, interroga un registro UDDI, effettuando una ricerca per nome, categoria, identificativo o specifiche, e ottiene informazioni sulla posizione di un documento WSDL, nel quale sono indicate le modalità per mettersi in contatto con il servizio Web richiesto e il formato dei messaggi di richiesta sotto forma di schema XML. Il client, a sua volta,

creerà un messaggio SOAP conforme allo schema XML trovato nel documento WSDL e invierà una richiesta all'host che dispone del servizio.

- Analisi della Sicurezza nei servizi Web.
  Le nuove caratteristiche di apertura e scambio dati, introdotti con l'avvento dei servizi Web, oltre alle problematiche d'interoperabilità, comportano nuove sfide per la sicurezza dei dati e dell'identità. Alla base dei Web Services c'è l'obiettivo di realizzare un assemblaggio tra componenti distribuiti sulla rete, eterogenee per linguaggi di sviluppo, piattaforme operative e funzionalità erogate, in grado di riconoscersi ed attivarsi tra loro in modo autonomo e dinamicamente riconfigurabile, grazie all'impiego di alcuni determinati standard di interfaccia. I vantaggi di questo modello di sistemi sono rilevanti: si risolvono "per definizione" la gran parte dei problemi di integrazione, si possono creare nuove soluzioni applicative semplicemente ricombinando in maniera diversa i vari componenti, si riduce la necessità di sviluppare del nuovo Software.
  Questo, però, introduce l'assoluto bisogno di considerare la sicurezza nell'uso dei Web Services. Questo porta all'analisi delle firme digitali, della gestione delle chiavi e della crittografia che rappresentano nuove sfide in relazione all'uso di XML e alle comunicazioni di tipo SOAP. Vengono esaminate in dettaglio le specifiche e le implementazioni attuali come XML-Encryption, XML-Signatures, SOAP –Secutity e XKMS.
- Studio dello standard per la sicurezza WS-Security, che nasce per ovviare ai problemi legati alla sicurezza della messaggistica SOAP.
- Applicazioni pratiche per i servizi Web. Studio e utilizzo dei principali toolkit che implementano il protocollo SOAP (e eventualmente il linguaggio WSDL).
  - Apache SOAP 2.3.1
    Questo toolkit è composto da una collezione di package Java che forniscono il supporto alla specifica SOAP 1.1. Questo toolkit non prevede il supporto per il linguaggio WSDL. Apache SOAP 2.3.1 può essere installato sia come client-side per pubblicare servizi SOAP utilizzabili in remoto. Sul lato server, per la pubblicazione dei servizi, il toolkit richiede la presenza di un web application server (ad esempio Apache Tomcat).
  - Microsoft SOAP Toolkit 3.0
    E' un Toolkit che fornisce interfacce di alto e basso livello che permettono di realizzare Web Services e di invocarli. Sono disponibili esempi ed un Wizard per esporre componenti COM già esistenti utilizzando IIS come Web Server e generando automaticamente anche il file WSDL.

Al suo interno troviamo una suite di elementi importanti tra cui un wizard che permette di creare automaticamente i listener SOAP per un oggetto COM. Questo, partendo da un componente COM esistente, è capace dalla TypeLibrary di decidere le interfacce ed i metodi da esportare, e crea per noi tutto il necessario per il corretto funzionamento. Il supporto Client-side è facilitato dall'introduzione di un oggetto COM che può gestire le richieste e le risposte di SOAP

Capitolo 1 – Introduzione

Introduzione generale

La rivoluzione del PC è iniziata quando i computer, PC o mainframe che fossero, erano vere e proprie isole di informazioni. L'era delle reti ha avuto inizio quando i costi dell' hardware si sono ridotti al punto da farne un bene di consumo. Questa trasformazione ha consentito l'introduzione di una prima forma di condivisione dei dati all'interno delle organizzazioni e ciò ha avuto sostanzialmente l'effetto di aumentare le dimensioni delle isole di dati, senza tuttavia contribuire in modo rilevante alla connessione tra di esse.

Internet ha trasformato radicalmente questo modello, consentendo la connessione di reti e sistemi informatici eterogenei attraverso una connettività universale e lo sviluppo di protocolli di presentazione e comunicazione standard come HTTP e HTML (linguaggio per il trasferimento di ipertesti su protocollo HTTP basato sul TCP/IP).

Perciò Internet nasce come strumento di condivisione di enormi quantità di informazioni a carattere scientifico grazie allo sviluppo dell'HTML, standard per la formattazione di ipertesti; le informazioni venivano trasmette in rete e rese disponibili agli utenti che potevano visualizzarne il contenuto (semplici pagine statiche) tramite l'uso di un browser abilitato. Col passare del tempo questo standard risultò limitativo in quanto era sorta la necessità di offrire servizi che scambiavano informazioni in modo dinamico, con continui aggiornamenti.

Questa barriera é stata superata grazie all'utilizzo delle seguenti tecnologie:
• TCP/IP – protocollo universale grazie al quale è possibile la comunicazione e lo scambio di informazioni tra sistemi appartenenti a diverse piattaforme software.
• HTML – linguaggio per il trasferimento di ipertesti su protocollo HTTP basato sul TCP/IP.
• JAVA – linguaggio che consente di scrivere applicazioni indipendenti dalla piattaforma e che permette di supportare la dinamicità dei contenuti.
La necessità di far interoperare in modo dinamico e rapido i sistemi distribuiti ha portato oggi allo sviluppo di una nuova tecnologia: i ***Web Services.***

I Web Services sono strumenti che consentono di creare soluzioni e sistemi in grado di interagire tra loro, in modo semplice e rapido, attraverso Internet e gli standard del settore. Requisito fondamentale per un Web Services è che sia utilizzabile da un'applicazione client scritta in un qualsiasi linguaggio, mediante qualsiasi strumento di sviluppo, e che "giri" su qualsiasi piattaforma. Estendendo quindi la loro capacità di ottenere l'indipendenza dell'applicazione dal protocollo di trasporto e dalla piattaforma di implementazione, essi permettono l'integrazione di tutte le applicazioni esistenti sul web senza dover procedere ad un *restyling* di parti di codice.
I web services rappresentano quindi un nuovo tipo di applicazioni aventi come caratteristica primaria la capacità di poter essere pubblicati, localizzati ed invocati dal web.
Interoperabilità è la parola chiave per inquadrare i nuovi sviluppi di Internet, dovuti proprio al nuovo concetto di Web Services.

Questo tipo di architetture consentono notevoli passi avanti verso il sogno della connettività globale: per svolgersi, i protocolli si basano su standard aperti e possono così realizzare i canali di interazione oggi necessari per accomunare gli utenti, indipendentemente dalle diverse tecnologie.

La comunicazione tra il web services e le altre applicazioni su Internet e/o su Intranet ha impegnato tantissimo gli sviluppatori. La maggior parte delle soluzioni individuate tendono ad essere specifiche della piattaforma, non scalano molto bene, spesso richiedono molti round-trip tra il Client e il Server. Altre soluzioni a questo tipo di problemi sono DCOM, CORBA, ecc. Le stesse comunque, incontrano non pochi problemi nel passare attraverso i Firewall.

Il protocollo SOAP riduce molti di questi problemi poiché fornisce un modo consistente e strutturato di trasporto dei dati e di chiamata dei metodi fra le potenziali applicazioni distribuite. L'esecuzione di questi servizi basati sul web necessita di numerosi componenti posti sui vari computer. Poiché questi sistemi comprendono molti computer, inclusi client, server di medio livello e DBMS (Database Management System), sono denominati sistemi distribuiti.

Solitamente i sistemi distribuiti utilizzano due modelli di comunicazione:

- Trasferimento di messaggi

- Richiesta/risposta

Il primo modello consente la comunicazione tra i sistemi mediante lo scambio di messaggi che possono essere spediti in ogni momento, questo provoca un' azione da parte del destinatario del messaggio, che a sua volta viene "risvegliato" per eseguire l'azione associata. L'elaborazione dei messaggi può essere:

- Sincrona
- Asincrona

Sincrona quando, a fronte della spedizione del messaggio, il destinatario immediatamente inizia l'esecuzione dell'azione associata. Asincrona quando, tra trasmettitore/ricevitore è interposta una coda di messaggi, e solo quando il ricevitore richiede di riceverli in coda li elabora. Ciò può essere fatto anche dopo molto tempo rispetto all' invio del messaggio.
Nel modello richiesta/risposta, la richiesta e la risposta sono unite e quindi si parla di sistema sincrono. La richiesta viene inoltrata da un'applicazione e questa, prima di continuare con l'elaborazione, attende i risultati. Il modello richiesta/risposta è utilizzato per consentire la comunicazione tra i componenti sui diversi computer attraverso le *RPC* (Remote Procedure Call).

Attualmente i due standard più diffusi per l'attivazione remota di procedure sono *DCOM* (Distributed Component Object Model) e *IIOP* (Internet Inter-Orb Protocol). Sono entrambi efficaci, anche se non progettati appositamente per l'interoperabilità, quindi non è possibile richiamare da un client un componente su un server prima di conoscere lo standard utilizzato dal server e prima di avere impostato la security. Su intranet è possibile limitare il sistema all'utilizzo di una piattaforma di riferimento, non appena si opera su internet, di solito non è possibile utilizzare una piattaforma uniforme nell'intero sistema. A questo punto DCOM e IIOP non consentono più la comunicazione tra due componenti all'interno del sistema né consentono agli utenti di transitare all'interno di domini affidabili. Per politiche di security, viene bloccato dai Firewall il passaggio di stream binari da porte TCP/IP.

SOAP rappresenta una soluzione a questi problemi, sfruttando la tecnologia web e la flessibilità e l'estensibilità di XML (eXtensible Markup Language), permette inoltre di

transitare all'esterno delle reti intranet, non procurando problemi ai Firewall. Senza dubbio le specifiche di XML e SOAP hanno segnato l'avvento di una nuova era.
Lo standard SOAP non introduce nuovi concetti in quanto si basa su tecnologia già esistente. Attualmente utilizza il protocollo HTTP come protocollo di trasporto di messaggi richiesta/risposta ed è indipendente dalla piattaforma ponendosi inoltre ad uno strato indipendente dal protocollo di trasporto. Teoricamente è possibile effettuare la richiesta utilizzando qualsiasi protocollo di trasporto ed avere la risposta con qualsiasi protocollo di trasporto. Un Package SOAP contiene informazioni che consentono di richiamare un metodo anche se nella specifica SOAP il nome del metodo stesso non viene definito. SOAP consente il passaggio dei parametri e dei comandi tra i client e i server HTTP indipendentemente dalle piattaforme e dalle applicazioni sul client e sul server. I parametri e i comandi sono codificati mediante XML.

L'ambiente informatico di oggi si sta evolvendo verso un sistema altamente distribuito nel quale il software costituisce la forza motrice, caratterizzato da innovazioni tecnologiche come i servizi web XML, piccoli componenti riutilizzabili basati sul linguaggio XML, che possono essere connessi e utilizzati come elementi di base per la realizzazione di funzioni specifiche.

In questo nuovo ambiente informatico gli scambi di informazioni avvengono attraverso più canali e in direzioni diverse, per consentire agli utenti di ottenere tutte le informazioni di cui hanno bisogno, indipendentemente dalla posizione e dalla piattaforma hardware. L'integrazione dei sistemi costituisce il principio ispiratore, e non una semplice conseguenza, dell'elaborazione distribuita e offre interessanti vantaggi, come uno scambio di dati più semplice e rapido all'interno e all'esterno delle organizzazioni e maggiori opportunità di connessione fra le aziende e i clienti.

Ma questo modello di elaborazione distribuita ha dato vita anche a importanti problemi, in particolare quello della **sicurezza**. Tanto i consumatori quanto le organizzazioni vogliono avere la certezza che i dati trasmessi su Internet, durante le operazioni di e-commerce e gli scambi di informazioni, rimarranno protetti e sicuri. Con i suoi nuovi livelli di integrazione fra client, server e servizi, che spesso operano oltre i confini dei domini, questo nuovo modello di elaborazione decentrato e distribuito esige che la sicurezza divenga una parte integrante dell'ambiente informatico.

L’uso dei web services nelle applicazioni commerciali evidenzia immediatamente l’importanza della sicurezza e della stabilità, caratteristiche indispensabili che devono essere attentamente considerate già nella fase di design delle applicazioni. Analizzando alcune implementazioni in fase di sviluppo o già realizzate, si nota che l’implementazione della sicurezza nella maggior parte dei web services non offre un sufficiente grado di protezione: alcuni non implementano affatto funzionalità di questo tipo, altri la implementano solo in parte appoggiandosi all’infrastruttura, altri invece creano un modello di security privato. L’adozione di soluzioni più o meno sicure è ovvia conseguenza al tipo di servizio erogato.

L’implementazione parziale della sicurezza permette, ad esempio, modalità di login e di autenticazione basate su SSL/TLS, prevedendo in genere operazioni di recovery con le quali viene inviata la password (oppure, nei casi peggiori, username e password) all’interno di un stesso messaggio e-mail (evidentemente in chiaro).

Esiste una ristretta classe di applicazioni basate su web services che si possono ritenere relativamente sicure che è rappresentata dalle applicazioni tipicamente B2B, nelle quali le parti si accordano su uno o più security design pattern per lo scambio di informazioni di sicurezza e applicative. Fino ad oggi questa soluzione si basava pesantemente sul protocollo SSL (quindi su HTTP) in aggiunta a forme di encryption.

Con le nuove specifiche basate su XML come XMLENC (encryption), XMLDSIG (firma digitale), SAML (Security Assertion Markup Language), XrML (eXtensible rights Markup Language), WS-Security (Web Services Security) e altre, SSL continua ad avere un ruolo importante e ben definito sebbene ne venga ridotto l’utilizzo.

SSL ha infatti una serie di svantaggi che si stanno evidenziando con l’evolversi dei nuovi scenari applicativi. Va riconosciuto comunque che SSL è un ottimo protocollo che si basa su algoritmi crittografici standard e a sua volta è diventato un protocollo standard supportato dalla maggior parte dei vendor di infrastruttura, in grado di garantire l’autenticazione, l’integrità e la riservatezza dei dati. Inoltre, trattandosi di un protocollo di sicurezza tra il layer HTTP e TCP, permette agli sviluppatori di non doversi adattare ad alcun pattern di sicurezza specifico. Bisogna però considerare alcuni aspetti e limiti che tale tecnologia implica nelle proprie architetture. Innanzitutto SSL è supportato solo su protocollo HTTP: sebbene questa affermazione venga data per scontata è bene ricordare che tale legame è stato dettato da esigenze di mercato e non da caratteristiche tecniche. SSL si pone come un protocollo separato e ben distinto tra lo strato HTTP e quello TCP, rendendolo utilizzabile anche da altri standard Internet diversi da HTTP, come FTP e

NNTP. Purtroppo, anche se la struttura di SSL lo permette, pochissimi vendor di infrastruttura hanno realizzato questo tipo di scenario. Un altro aspetto cruciale di SSL è il carico di lavoro per le operazioni crittografiche: è molto dispendioso per le CPU dei web server i quali spesso sono soggetti ad analisi approfondite e molto critiche di scalabilità e di performance. SSL utilizza le chiavi simmetriche per la cifratura del canale mentre le chiavi asimmetriche risolvono lo scambio delle chiavi di encryption tra il client e il server. Il certificato del server, inoltre, viene utilizzato dal client per verificare che il server con il quale sta per scambiare delle chiavi di sessione (simmetriche), e successivamente dei dati sensibili, sia realmente la macchina con la quale vuole scambiare tali informazioni. Questo modello implica che SSL sia un protocollo statefull, ovvero deve mantenere delle informazioni tra il client e quel determinato server come la chiave simmetrica e il riferimento all'indirizzo o al nome del server, introducendo alcuni problemi architetturali in ambienti di load balancing e quindi di scalabilità. I tipi di intervento che possono essere adottati in questi scenari sono tipicamente di infrastruttura.

In attività di consulenza sulla sicurezza, capita spesso che le parti di un progetto meno analizzate da un punto di vista di Threat analysis (ovvero analizzare e mitigare le possibili minacce) siano proprio quelle che demandano la sicurezza a SSL. Non è affatto vero che utilizzando un canale criptato i dati sensibili dell'applicazione siano immuni da qualsiasi attacco, infatti la sicurezza delle applicazioni non è una feature che si può aggiungere al prodotto sperando che questa risolva tutti i problemi. La sicurezza non è quindi un prodotto ma è piuttosto un processo tramite il quale l'intera infrastruttura deve essere analizzata. Quindi è indispensabile studiare con attenzione tutti gli aspetti di Risk Management dell'infrastruttura di networking, del Sistema Operativo, delle scelte applicative e soprattutto della qualità del codice scritto.

I web services sono sinonimo di interoperabilità, e di conseguenza le applicazioni che si basano su tali architetture devono poter godere della massima scalabilità disponibile. Per questo motivo si preferisce spostare più ad "alto livello" le informazioni di sicurezza inserendole in un messaggio di testo SOAP completamente indipendente dall'infrastruttura sottostante.

Capitolo 2 – Introduzione a SOAP

Il protocollo SOAP

SOAP è un acronimo per Simple Object Access Protocoll, un protocollo leggero pensato per facilitare l'interoperabilità tra applicazioni e piattaforme eterogenee nell'era della programmazione distribuita su Internet. Alla base di SOAP sta un'idea tanto semplice quanto intelligente che si può riassumere in questo modo: non inventare nessuna nuova tecnologia, ma utilizzare al meglio quelle esitenti.

Con SOAP il web diviene qualcosa di più di un semplice scambio di documenti: infatti SOAP mette le applicazioni in comunicazione tra loro, e si candida per costruire il framework su cui costruire i web services. Inoltre SOAP è candidato a diventare il meccanismo che consente a tutti i sevizi di esporre le proprie caratteristiche e di comunicare "service to service" o "componente to service", sia tra piattaforme omogenee che, soprattutto tra piattaforme eterogenee. Il cambiamento apportato da SOAP è dovuto al fatto che oggi come oggi è possibile utilizzare una varietà di protocolli binari per l'invocazione di oggetti remoti. Questo significa che le applicazioni client sono create per comunicare con applicazioni server specifiche. Ma il più grande dei problemi consiste nel fatto che se si vuole eseguire i client al di là di un firewall, è necessario configurare quest'ultimo in modo apposito, sempre che sia possibile. E SOAP risolve brillantemente entrambi i problemi, consentendo uno sviluppo e soprattutto una distribuzione semplificata delle applicazioni.

Caratteristiche

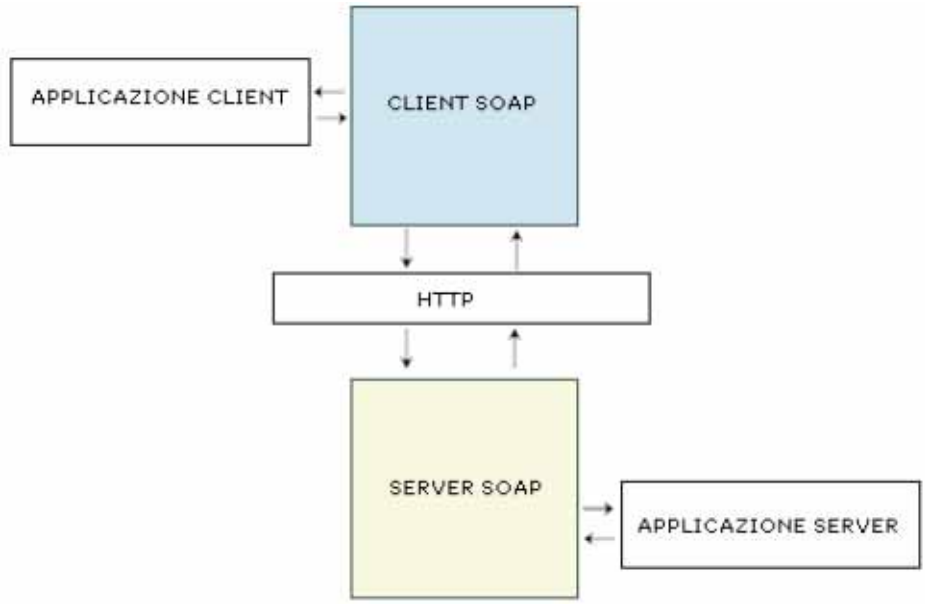
SOAP è un protocollo in via di definizione che consente un meccanismo di interazione client/server affiancabile alle tradizionali metodologie per la chiamata /attivazione di oggetti remoti (DCOM e CORBA). Si avvale del protocollo HTTP (o HTTPS) per il trasporto delle informazioni, e del linguaggio XML per la codifica delle informazioni. L'uso del protocollo HTTP consente a SOAP di lavorare in modo trasparente attraverso gli attuali Proxy Server e Firewall, oltre che di agevolarsi di pratiche comuni nel mondo HTTP , come le tecniche di redirection, caching, connection management e gestione della sicurity. Da un punto di vista tecnico si limita a definire un formato di messaggi che viaggiano in rete

tramite HTTP, e in questo modo può attraversare tutta la struttura Internet esistente, senza necessità di cambiamenti. Ma ci sono anche altri vantaggi e uno di questi è che SOAP non è legato ad una particolare piattaforma operativa, ad un particolare linguaggio di programmazione o modello.

SOAP gestisce due tipi di messaggi CALL e RESPONSE. Di fatto un messaggio di call permette di invocare un servizio remoto, mentre un messaggio di Response altro non è che il risultato proveniente dal servizio invocato. Con SOAP sono sufficienti poche righe di codice scritte in qualsiasi linguaggio di programmazione per costruire un'applicazione RPC. Il protocollo SOAP può essere informalmente caratterizzato dall'equazione: XML + HTTP = SOAP. In maniera più formale, come definito nelle sue specifiche, questo protocollo è costituito da tre parti che sono state pensate per essere funzionalmente ortogonali, permettendo così una maggior semplicità, ottenuta attraverso la modularizzazione. Le tre parti sono:

- *SOAP envelope*: definisce una struttura per descrivere cosa c'è nel messaggio, come deve essere processato e se esso è opzionale o obbligatorio;
- *SOAP encoding rule:* un meccanismo di serializzazione che può essere usato per scambiare istanze di tipi di dati definiti dalle applicazioni;
- *SOAP RPC rapresentation:* una covenzione utilizzata per rappresentare le chiamate e le risposte alle procedure remore (RPC).

L'architettura abbastanza semplice è la seguente: (Figura 2.1)



Origini e obiettivi

SOAP nasce, in buona sostanza, nel contesto della programmazione distribuita, e, infatti, una delle sue applicazioni più comuni è come protocollo per le Remote Procedure Call (RPC). I protocolli per le RPC sono nati in base al paradigma richiesta/risposta, in alternativa allo scambio di messaggi. Negli anni novanta, con l'avvento delle metodologie orientate agli oggetti, sono nati dei protocolli basati su questi paradigmi (Object RPC) che cercavano di accordare le metodologie object-oriented e i protocolli di rete. Attualmente le più note tecnologie basate su ORPC sono DCOM, CORBA, IIOP e EJB. Tutti questi meccanismi sono accomunati da molti aspetti simili, ma esistono altre caratteristiche che li rendono incompatibili. In estrema sintesi, ciascuna di queste metodologie sembra essere buona per effettuare comunicazioni tra server omogenei (ovvero che utilizzano la stessa tecnologia, e, in alcuni casi, lo stesso produttore), ma è altrettanto inadeguata quando s'instaurano delle comunicazioni client-server, soprattutto se la comunicazione avviene via Internet, a causa della stretta corrispondenza che queste tecnologie esigono tra client e server. DCOM e IIOP (EJB è comunque basato su IIOP), infatti, oltre a non poter comunicare fra di loro se non tramite l'utilizzo di appositi bridge, sono tutti poco adatti a presenza di Firewall.

Riassumendo, alcuni dei principali aspetti che risultano essere problematici nel campo dell'attuale programmazione distribuita sul web sono:

- protocolli eterogenei che faticano a comunicare;
- protocolli che non funzionano bene attraverso i firewall.

E per dare una soluzione a questi problemi che nel Settembre 1999 l'IETF (Internet Engineering Task Force) rilascia la versione 1.0 delle specifiche di SOAP. Soap fa seguito a XML-RPC, un semplice protocollo creato per effettuare chiamate a procedure remote tramite HTTP, ed è il frutto del lavoro che coinvolge alcune tra le più grandi potenze informatiche mondiali, come Microsoft e IBM. La versione 1.1 delle specifiche è stata sottoposta, nel maggio 2000, all'esame del W3C per essere standardizzata, ma la sua rapida diffusione lo sta già facendo diventare uno standard de-facto.

In maniera molto generale, quindi, si può affermare che l'obiettivo di SOAP è di permettere ad applicazioni e componenti eterogenee (ovvero di natura diversa, basate su diverse piattaforme e realizzate in diversi linguaggi) di interagire fra di loro usando messaggi e

chiamate a procedure remote, eliminando gli ostacoli che intralciano gli attuali pacchetti dei sistemi distribuiti, soddisfacendo alle seguenti esigenze:

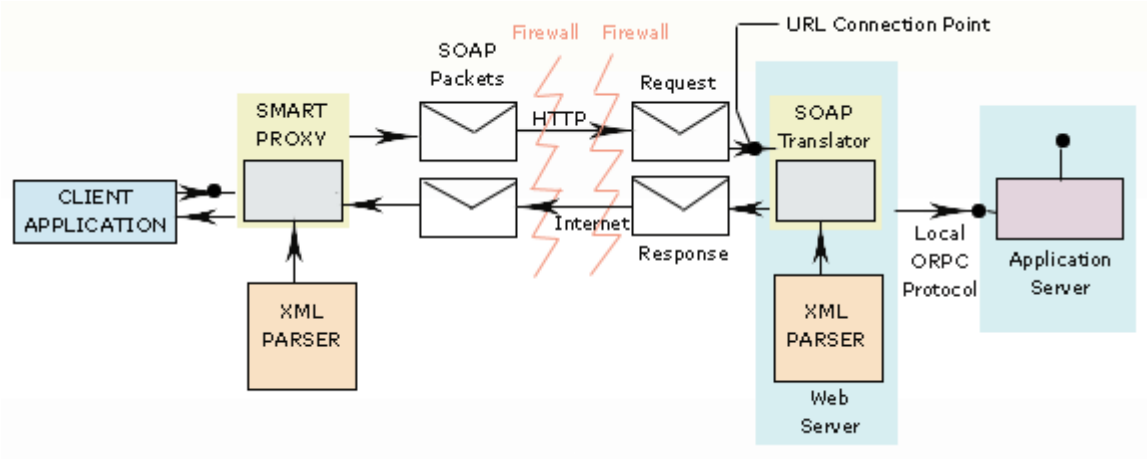
- utilizzare degli standard;
- creare un protocollo che lavori anche attraverso i firewall;
- avere un framework estensibile;
- avere una tecnologia semplice e implementabile con uno sforzo ridotto
- avere una tecnologia agnostica nei confronti delle piattaforme e dei linguaggi di programmazione.

Visione di alto livello

Nelle specifiche disponibili presso il W3C ([W3C00a]), SOAP è definito come: “un protocollo leggero per scambiare informazioni in un ambiente decentralizzato e distribuito”. Più dettagliatamente, si può affermare che SOAP fornisce un meccanismo punto-punto, semplice e leggero, per scambiare informazioni tipizzate e strutturate in un ambiente distribuito utilizzando XML. In effetti, per sua natura, può essere utilizzato in svarianti sistemi, fornendo un meccanismo che va dal semplice scambio di messaggi fino alle chiamate di procedure remote. SOAP si basa su XML e può essere usato in combinazione con molti altri protocolli per il trasporto sulla rete (HTTP, SMTP. FTP), anche se il solo protocollo utilizzato nelle sue specifiche è HTTP. In questo modo vengono sfruttate al meglio le tecnologie esistenti: XML e HTTP hanno già le caratteristiche che SOAP si prefigge di avere , e che da essi eredita, come la semplicità, la flessibilità, l’indipendenza dalla piattaforma, l’essere basato sul testo.

In particolare XML sembra essere la miglior scelta per rappresentare i dati tramite Internet, essendo un protocollo basato sul testo e indipendente dalla piattaforma, semplice, estendibile ed estremamente flessibile grazie ai meccanismi dei namespace e degli XML Schema. HTTP sembra essere un ottimo mezzo di trasporto per le chiamate a procedure remote su internet, per il fatto che esso non è assoggettato alle intercettazioni dei firewall e per di più, quasi tutti i Web Browser ed i Server lo supportano in modo nativo e poi perché è semplice da gestire. SOAP utilizza l’XML e HTTP, per aggirare il firewall e permettere di processare le RPC lanciate dalle applicazioni provenienti sia da Internet che dal Web.

Normalmente, i dati vengono incapsulati all'interno dell'HTTP o di un altro protocollo di trasporto, ed inviati ai Server. I Server estraggono i dati SOAP, li elaborano come richiesto ed inviano i risultati in forma di risposta SOAP.



Dal momento che i protocolli Web sono installati su tutte le principali piattaforme operative, l'HTTP e l'XML costituiscono una soluzione immediatamente disponibile per risolvere il problema dell'interscambio di comunicazioni tra programmi in esecuzione su sistemi operativi diversi. SOAP specifica esattamente come creare un Header HTTP ed un File XML, in modo tale che un programma installato su una macchina possa invocare un programma posto su di un'altra macchina, scambiandosi tutte le informazioni del caso. Nello stesso modo, si può definire come il programma chiamato deve restituire i dati, frutto delle proprie elaborazioni.
In sintesi, SOAP può gestire qualsiasi tipo di messaggio, permettendogli di trasportare dati di ogni genere e scopo. Può operare con tutti i linguaggi di programmazione o di scripting, con tutti i modelli di oggetti e qualsiasi protocollo di connessione Internet.
Permette inoltre la comunicazione, indipendentemente dai singoli modelli applicativi: due computer possono lanciarsi reciprocamente delle RPC senza dover conoscere alcunché dei sistemi di Back-end sui quali queste verranno eseguite.

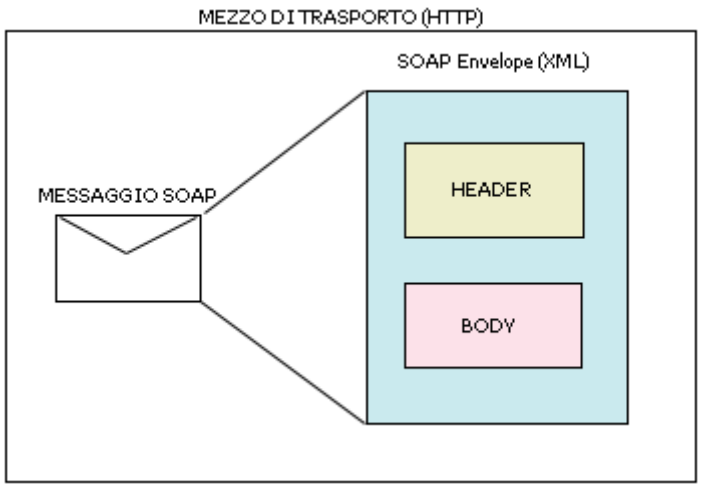
Ad alto livello SOAP può essere visto come una specifica da definire:

- uno stile di codifica che usa XML per rappresentare le informazioni;
- un modo standard per trasportare documenti XML tramite HTTP;
- regole per scambiare messaggi:
- un mezzo per eseguire RPC;
- un meccanismo per la gestione degli errori di fallimento;

- uno strato in un’architettura multi-strato.

SOAP non è una semantica per definire delle interfacce per le applicazioni (non ci sono API). In pratica SOAP è un meccanismo che permette di scambiare messaggi tra un mittente e un ricevente e molto spesso i messaggi SOAP sono basati sul modello della richiesta/risposta (request/response). La sua semplicità si può riassumere in questo modo: il client richiede al server d’invocare l’esecuzione del metodo di un oggetto e il server spedisce la risposta al client. I messaggi sono codificati con XML e, di solito, incapsulati in richieste HTTP, come esemplificato in figura 2.1

Struttura di un messaggio SOAP



Vantaggi e svantaggi di SOAP

Tutti gli strumenti presentano alcuni aspetti vantaggiosi e altri che possono apparire svantaggiosi. La visione che si ha di SOAP dipende interamente dal problema che si intende risolvere e dalle limitazioni imposte dalla situazione in cui si opera.

Un’architettura di oggetti distribuiti richiede l’implementazione di:

- un meccanismo di serializzazione che converta la chiamata di metodo in un formato adatto alla trasmissione in rete;
- un livello di trasporto che trasferisca i dati del metodo tra i sistemi remoti;
- un meccanismo che supporti l’attivazione/disattivazione ed il rilevamento dell’oggetto;
- un’implementazione di un modello di protezione sia per il sistema remoto sia per il sistema locale.

SOAP è il livello minimo che gestisca un meccanismo di serializzazione (usando la semantica XML) e di trasmissione (principalmente con protocollo HTTP). SOAP non è stato progettato per sostituire l'intera architettura distribuita: ad esempio SOAP non implementa la protezione, tuttavia può utilizzare il protocollo HTTP consentendo l'impiego della protezione a livello delle applicazioni.

Diversamente da architetture di oggetti distribuiti, quali ad esempio CORBA, SOAP è semplicemente un protocollo via cavo e quindi per confrontarlo con altre architetture è necessario implementare il protocollo SOAP come parte di un'architettura distribuita autonoma, oppure sostituirlo al protocollo via cavo di un'architettura esistente. In generale quando si confronta il protocollo via cavo SOAP con altri protocolli via cavo viene riscontrata una perfetta adattabilità in diverse aree, SOAP è relativamente efficiente ed è flessibile ad eventuali sviluppi grazie a proprietà derivate da XML.

Consideriamo alcuni vantaggi connessi all'utilizzo di SOAP:

- SOAP è basato su tecnologie aperte e quindi incoraggia le applicazioni distribuite;
- La specifica SOAP può consolidare diversi protocolli HTTP in una singola specifica, inoltre per SOAP l'uso di HTTP non è obbligatorio;
- Usando il protocollo le modifiche apportate all'infrastruttura di SOAP non compromettono le applicazioni;
- SOAP consente di avere un sistema con elevata scalabilità cioè con grande capacità dell'architettura remota di gestire un numero cospicuo di client concorrenti. In modo particolare SOAP è più scalabile di CORBA DCOM e RMI di Java soprattutto se si usa il modello richiesta/risposta di HTTP.

D'altro canto SOAP presenta anche aspetti svantaggiosi:

- sebbene l'ultima versione della specifica SOAP riduca l'uso del protocollo HTTP, è ancora possibile trovare svariate implementazioni che usano HTTP;
- la deserializzazione globale del metodo deve essere eseguita mediante operazioni di rilevamento;
- SOAP serializza per valore ed attualmente non supporta la serializzazione per riferimento; questo implica che diverse copie dell'oggetto conterranno nel tempo informazioni relative allo stato non coerenti.

Soap eliminerà la complessità degli attuali approcci alla comunicazione tra più macchine, specialmente quando i sistemi sono separati da collegamenti Wan o Internet. Purtroppo, alla semplicità e alla facilità d'uso di Soap si contrappongono l'inefficienza delle comunicazioni del suo formato testo e il suo limitato insieme di funzionalità. Ad esempio, Soap non offre nessun supporto

per gli scambi transazionali e le specifiche proposte non comprendono la sicurezza e la consegna garantita dei messaggi. Tuttavia, Soap può essere facilmente utilizzato all'interno di infrastrutture consolidate, come i sistemi per la gestione delle code di messaggi, che forniscono queste funzionalità, purché a entrambe le estremità di una connessione Soap si utilizzino le stesse infrastrutture.

Una piccola nota su XML-RPC: questo protocollo è un modo più semplice per effettuare RPC su XML ma forse meno famoso in quanto probabilmente il rivale SOAP ha ricevuto più attenzioni, sicuramente in quanto proposto da Microsoft ed IBM. Con API standard per l'accesso all'RPC, XML-RPC avrà la possibilità di tornare alla ribalta come una alternativa per il protocollo di messaggio.

Applicazioni future di SOAP

Rispetto alle altre tecnologie per il collegamento di piattaforme diverse, Soap risolve i difficili problemi della programmazione distribuita tra più organizzazioni con maggiore semplicità e minore dipendenza dai diversi produttori. Per esempio, una delle estremità di una connessione Soap può essere basata sullo standard Corba, mentre l'altra sullo standard Dcom. Soap è anche una tecnologia più adatta di Corba e di Dcom per i modelli di programmazione che si utilizzano su Internet privi delle informazioni di stato.

E' possibile utilizzare i principi base di SOAP per la creazione di nuovi protocolli che offrono funzionalità diverse da HTTP. Dal momento che il protocollo HTTP si basa sul modello richiesta/risposta esso non è adatto al paradigma di messaggistica ed accodamento. Molti sistemi si basano su architetture di messaggistica e richiedono quindi questo tipo di supporto.

Capitolo 3 – SOAP e XML

Il Simple Object Access Protocol (SOAP) è un protocollo basato su XML che consente a due applicazioni di comunicare tra loro sul Web, e definisce il formato dei messaggi che due applicazioni possono scambiarsi utilizzando i protocolli Internet, come ad esempio HTTP, per fornire dati e richiedere elaborazioni. SOAP si fonda pesantemente su XML e in effetti un messaggio SOAP non è altro che un documento XML che segue determinate regole. Per maneggiare un documento SOAP si possono usare quindi gli stessi strumenti utilizzati per i documenti XML. Si procederà pertanto ad illustrare cos'è e come si usa un documento XML.

XML

XML, ovvero l' eXtendible Markup Language è nato nel 1997 come sottoinsieme dello Standard Generalized Markup Language (SGML), con l'idea di adattare questo linguaggio all'ambiente di Internet, rendendolo simile all'HTML. Da allora ha continuato a guadagnare importanza, assumendo sempre più un ruolo di rilievo nel panorama della programmazione. La versione 1.0 (seconda edizione) delle sue specifiche è stata standardizzata dal W3C nell'Ottobre 2000.

Definire cos'è l'XML non è banale come può sembrare. XML , in effetti, non è propriamente un linguaggio come il suo nome può, ingannevolmente, far supporre, ma bensì un meta-linguaggio, ovvero un insieme di regole che permettono di costruire molteplici linguaggi specializzati, in genere chiamati "applicazioni". In pratica, XML non è una singola tecnologia, ma un insieme di tecnologie e specifiche che continua a crescere ed evolvere nel tempo. XML è sempre più diffuso e utilizzato, soprattutto nei seguenti scenari;

- pubblicazione di documenti sul Web, ma anche in altri contesti;
- codifica e scambio di dati, anche prelevati da database relazionali, tra sistemi eterogenei;
- configurazione di sistemi;
- applicazioni specifiche, con la creazione di particolari vocabolari.

Documenti XML

XML descrive una classe di oggetti che rappresentano dati, detti documenti XML, e, in parte, definisce il comportamento di un programma software che li processa. Alla base di XML stanno alcune fondamentali linee guida. In particolare i suoi progettisti hanno pensato che esso debba:

- essere utilizzato su Internet;

- supportare una vasta gamma di applicazioni;
- essere in forma facilmente leggibile per gli esseri umani;
- essere facile da utilizzare, veloce da preparare;

e, inoltre

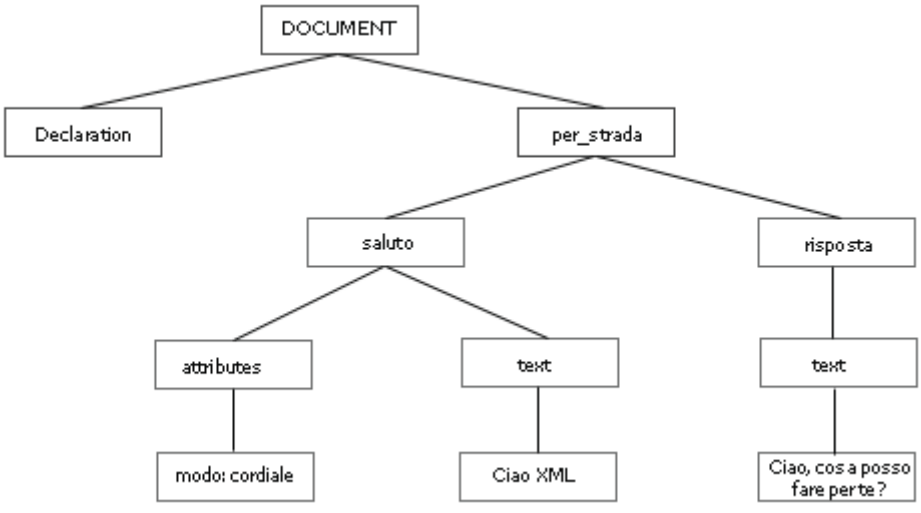
- deve essere semplice realizzare delle applicazioni che lo processano.

La nozione centrale di XML è il documento XML. Un documento XML è un'entità d'informazione che può essere vista in due modi: come una sequenza lineare di caratteri che formano un testo (suddiviso in markup e character data, ossia qualsiasi carattere che non costituisce un markup) (Listato 2.1); oppure come una struttura di dati astratta fatta ad albero e costruita da nodi (Figura 2.1). Per passare da un punto di vista ad un altro è sufficiente un parser in grado di processare il documento.

LISTATO 3.1 ESEMPIO DI UN DOCUMENTO XML

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

<per_strada>

              <saluto modo="cordiale">Ciao XML</saluto>

             <risposta>Ciao, cosa posso fare per te?</risposta>

</per_strada>
```

FIGURA 3.1: Rappresentazione della struttura ad albero del Listato 3.1



I markup (o marcatori) servono a descrivere ogni struttura che abbia un significato rilevante, associando a tali strutture coppie di attributi-valori. Originariamente XML fornisce anche un meccanismo per stabilire dei vincoli sulle strutture dati che esso descrive, o se si preferisce, per specificare sintatticamente le strutture che sono contenute nel documento (una specie di "grammatica" che stabilisce le regole cui deve sottostare un documento). Tale meccanismo è il Document Type Definition (DDT) ma, sebbene esso faccia ancora parte delle specifiche XML, sta progressivamente scomparendo, essendo sostituito da un meccanismo analogo ma preferibile, gli XML Schema. I documenti XML sono modulari, ossia possono essere costituiti da più componenti fisiche diverse; ciò li rende estremamente flessibili e versatili.

Le componenti del documento XML

Si è detto che un documento XML possiede una struttura fisica ed una struttura logica. Se la struttura logica è una struttura ad albero i cui nodi sono gli elementi che si vogliono rappresentare, la struttura fisica invece contiene i dati effettivi utilizzati, in un documento, come il testo memorizzato nella memoria del computer, un'immagine memorizzata nel web o altro ancora.

Dal punto di vista logico, un documento XML può essere composto dalle seguenti parti:

- un prologo, composto dalla dichiarazione XML, che stabilisce la natura del documento XML e dalla definizione di tipo di documento, opzionale, che richiama o contiene il DDT o lo schema;
- l'elemento Document., ossia l'elemento radice dell'albero;
- commenti e/o istruzioni per processare (PI: Processing Instructions), anch'essi opzionali;
- il testo, come visto suddiviso in markup e character data (ed eventualmente altri commenti e PI).

Nel prologo, la dichiarazione XML identifica la versione delle specifiche XML a cui è conforme il documento, e nonostante sia definita come elemento opzionale, è estremamente raccomandato utilizzarla. Può inoltre contenere una dichiarazione di codifica (encoding) per permettere di assegnare diverse codifiche di caratteri e una dichiarazione di documento autonomo (standalone), che indica l'esistenza o meno di dichiarazioni di markup esterne al documento. La dichiarazione XML ha il seguente aspetto:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

La definizione del tipo di documento è costituita da codice markup che indica le regole grammaticali o la definizione del tipo di documento DDT per una particolare classe di documenti. Tale definizione può appartenere ad un file esterno. L'elemento Document contiene tutti i dati di un

documento XML, ed è l'indispensabile elemento radice dell'albero. Tra il suo tag di apertura e quello di chiusura sono annidati tutti gli oggetti che appartengono al documento stesso.

I commenti e le processing instruction (PI) sono delle componenti che si distinguono dagli altri elementi di testo per la loro particolare sintassi, come si può vedere nel codice seguente:

```
<!-- Questo è un commento, la riga seguente è una PI -- >

<?xml – stylesheet href=”style.css” type=”text/css”?>
```

Quando un parser analizza un file XML e incontra una di queste due componenti, non la analizza. Le process instruction servono per invocare delle applicazioni esterne al file XML: la prima stringa dopo il marcatore d'apertura è l'applicazione che deve essere usata, mentre le restanti sono dati da passare all'applicazione. Nell'esempio precedente viene eseguita un'istruzione che applica un foglio di stile al file XML.

Per quanto riguarda la struttura fisica di un documento, essa è rappresentata da tutto il contenuto del documento stesso. Quando si ricorre ai DDT, possono essere definite anche delle unità di memorizzazione, dette entità che possono fare parte del documento stesso oppure trovarsi in un documento esterno. Esistono comunque anche delle entità che sono svincolate dalla definizione del tipo di documento, e che permettono di rappresentare dei caratteri particolari, come quelli solitamente utilizzati dal linguaggio (<,>,”,’,&,/). Tali entità si dividono in entità analizzabili (*PCDATA*), che cioè possono essere elaborate dal parser, ed entità non analizzabili (*CDATA*), che viceversa servono qualora si voglia inserire del testo che non venga processato dal perser.

Linguaggio XML

Come linguaggio, XML deve sottostare a delle regole sintattiche. Poiché XML utilizza fortemente la strutturazione per descrivere un documento, le regole linguistiche rispecchiano tale caratteristica. Alla base di XML sta il markup, che genericamente è usato per descrivere degli elementi. Gli elementi sono contenitori per la descrizione di altri elementi e testo. Sintatticamente, un elemento deve essere descritto con l'utilizzo di tag (etichette). XML richiede, affinché un documento possa essere considerato ben formato, che tutti i tag aperti vengano anche chiusi (e questa caratteristica lo differenzia da HTML). Se un elemento è vuoto, ossia non contiene nessun elemento di testo al suo interno, può essere anche rappresentato con una forma speciale di tag: `<ELEMENTO_VUOTO/>`. Per ogni elemento inoltre XML permette di specificare degli attributi. Gli attributi consentono di associare valori a un elemento senza che siano considerati parte del contenuto dell'elemento stesso. Ritornando all'esempio del Listato 3.1 nella riga:

```
<saluto modo="cordiale">Ciao XML</saluto>
```

modo è un'attributo dell'elemento `<saluto>`.

Documenti XML ben-formati e validi

Il linguaggio XML possiede due caratteristiche fondamentali: la capacità di fornire una struttura ai documenti e di rendere i dati autodescrittivi. Queste caratteristiche non sarebbero di alcuna utilità se non si potessero far rispettare le regole strutturali e grammaticali.

Documenti ben-formati

Affinché un programma che processa un documento XML (un parser) sia in grado di creare la struttura logica del documento in modo corretto, il documento XML deve rispettare una serie di vincoli. Se lo fa, il documento è detto ben formato. I vincoli più importanti sono i seguenti:

- gli elementi devono formare una struttura ad albero, la qual cosa implica che vi sia un elemento radice, ossia un elemento che racchiude tutti gli altri, e che tutti gli altri elementi abbiano esattamente un genitore. Nella pratica, questo vincolo si traduce nell'avere tutti gli elementi annidati gli uni negli altri.
- La struttura ad albero deve essere esplicitata tramite l'utilizzo delle etichette (tag), questo implica che ogni elemento aperto (per esempio `<saluto>`) deve essere anche necessariamente chiuso (`</saluto>`).
- Tutti i valori degli attributi devono essere racchiusi tra virgolette (“”).

In pratica, un documento XML deve essere ben formato per essere tale, e questa è la condizione per poter utilizzare XML. Ma il linguaggio permette di essere ancora più rigidi nella sua definizione ed utilizzo, tramite il meccanismo dei DTD o degli Schema, che introducono al concetto di documento valido.

Documenti XML validi

La definizione del tipo di documento DTD specificata nel prologo delinea tutte le regole “grammaticali” relative al documento. Un documento XML è valido quando rispetta rigidamente tutte queste regole. Un parser sarà detto valicante quando oltre a stabilire se un documento è ben formato, sarà in grado di verificare se esso è anche conforma al suo DTD Schema.

Document Type Definition (DTD)

Il Document Type Definition (DTD) di un documento XML costituisce l'insieme di regole cui il documento deve sottostare, fornendo una sorta di grammatica. Un DTD può essere costituito di due parti: un sottoinsieme DTD interno oppure un sottoinsieme DTD esterno. I sottoinsiemi differiscono solo per la loro locazione fisica, essendo il primo incorporato nel documento XML che descrive mentre il secondo risiede altrove (in un diverso file). Un documento può contenere anche entrambi i sottoinsiemi, con la regola che le definizioni interne hanno la precedenza su quelle esterne, permettendo così, per esempio, di modificare o rendere più dettagliate delle grammatiche generali. Il DTD interno si trova nel prologo del documento nella sezione dedicata appunto alla sua definizione. Un DTD esterno, invece, deve essere espressamente riferito con un'apposita istruzione che riporta il nome del file in cui il processore lo può trovare, come nell'esempio seguente:

```
<!DOCTYPE MODEL SYSTEM "wpdl.dtd">
```

I vantaggi nell'utilizzare DTD esterne sono evidenti: alleggerimento del documento XML e possibilità di riutilizzo delle definizioni. I DTD permettono di dichiarare le seguenti componenti di un documento:

- Elementi (ELEMENT);
- Attributi (ATTLIST);
- Entità (ENTITY).

Le dichiarazioni degli elementi permettono di descrivere quali elementi potranno far parte del documento, ricorrendo alle specifiche di contenuto che possono essere suddivise in quattro tipi: elenco di altri elementi (modello di contenuto), parole chiavi EMPTY e ANY, contenuto di vario tipo. La parola chiave EMPTY viene usata per specificare che un elemento deve essere vuoto. Un esempio:

```
<!ELEMENT VUOTO EMPTY>
```

indica che l'elemento con nome vuoto nel documento potrà assumere solo la seguente forma: `</VUOTO>`. Viceversa, la parola chiave ANY specifica che non vi è nessuna restrizione sulla natura del contenuto, permettendo quindi di inserire qualsiasi elemento in un qualsivoglia ordine. Infine le specifiche di contenuto permettono di definire un singolo insieme di alternative, separandole con il simbolo pipe (|).

La sintassi per dichiarare degli attributi di un elemento invece è la seguente:

```
<!ATTLIST ElementName AttributeName Type Default>
```

dove vengono elencate nell'ordine, dopo la dichiarazione, il nome dell'elemento per cui si sta definendo l'attributo, il nome dell'attributo, il tipo dell'attributo, che può essere di tipo stringa, enumerazione o "tokenized" (cioè che ha vincoli lessicali e semantici variabili), ed eventualmente il valore predefinito per l'attributo.

Le entità, o unità di memorizzazione, possono essere definite utilizzando la sintassi:

```
<!ENTITY Name Definition>
```

Esistono diversi tipi di entità. Le entità analizzabili (o di testo) sono quelle che possono essere processate dal parser. La loro funzione è quella di definire delle porzioni di testo che poi potranno essere richiamate con un semplice riferimento. Un esempio:

```
<!ENTITY QUOTE "quam minimum credula postero">
```

che può essere quindi inserita altrove nel documento, utilizzando il simbolo &:

```
<verso>Carpe diem, &QUOTE </verso>
```

Queste entità sono anche dette generali. Possono essere definite anche delle entità di parametro, che si definiscono con il simbolo % e possono essere utilizzate solo all'interno del DTD stesso. Le entità non analizzabili, per contro, sono dei riferimenti a file binari, come file d'immagini, di testi, in formato binario, audio, video, o altro. Esse richiedono informazioni aggiuntive rispetto alle precedenti, come per esempio un'annotazione riguardante il formato o il tipo della risorsa che rappresenta. Anche entità come i DTD, possono a loro volta essere interne o esterne. Infine, in combinazione con le entità esterne, i DTD permettono di usare la dichiarazione di annotazione (NOTATION) per definire quale applicazione deve essere usata per processare l'attività. E' usata raramente perchè i tipi MIME (Multi-purpose Internet Mail Extension) che possono essere dichiarati come attributi delle entità, perseguono lo stesso obiettivo.

Namespace

Quanto fin qui visto è ciò che è stabilito nelle specifiche di XML 1.0. Con i DTD è possibile creare dei meta-linguaggi che possono soddisfare le esigenze di svariati campi applicativi.

Ma i DTD, pur essendo attualmente molto utilizzati, hanno caratteristiche e limitazioni che li allontanano dalla perfezione, Prima di analizzare quali sono i loro difetti, conviene introdurre un meccanismo utile nell'utilizzo di XML: i namespace.

Permettendo di creare diversi vocabolari (definizioni di elementi e attributi) e di utilizzarne contemporaneamente più d'uno, XML va incontro al problema del conflitto di nomi. Esemplificando, un'applicazione potrebbe utilizzare un tag `<address>` per indicare il domicilio di una persona, mentre per un'altra applicazione potrebbe intendere lo stesso elemento come indirizzo per la posta elettronica. Per risolvere questo problema, il W3C ha introdotto il meccanismo dei namespace. Quindi un Namespace è un nome fittizio, un alias, che si è soliti assegnare ad un percorso remoto di un file, tipicamente un file di schema che viene legato al documento XML. I namespace permettono di definire dei nomi in modo universale. Un namespace è una collezione di nomi, identificati da un Universal Resource Identifier (URI) , utilizzati in XML come tipi di elementi o nomi di attributi. Il meccanismo è molto semplice: consiste di un prefisso e di un URI, separati da un simbolo ":" (due punti). Insieme costituiscono un nome qualificato (Qname).

La sintassi per definire un Namespace XML è la seguente: `xmlns:Nome_Namespace=URI`
L'utilizzo del prefisso permette di distinguere quali elementi appartengono ad un namespace piuttosto che ad un altro. L'attributo `xmlns` serve come introduzione al prefisso utilizzato dal namespace, l'URI invece viene indicato come discriminante del namespace ed ha un valore informativo e non dichiarativo. Si utilizza un URI, piuttosto che un nome fittizio, perché per definizione un URI è unico, quindi si evitano ulteriori problematiche di collisione dei nomi.

L'esempio seguente mostra come un elemento XML venga associato ad un determinato namespace:

```
<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3c.org/1999/XSL/Transform>
```

`<xsl:stylesheet>` è l'elemento costituito da un nome qualificato. Xsl rappresenta il prefisso, mentre stylesheet costituisce la parte locale del nome. Il prefisso viene associato ad un URI mediante l'attributo xmlns (che, si nota, costituisce a sua volta un Qname). In questo modo qualsiasi elemento nel seguito del documento che abbia come prefisso xsl verrà associato al namespace definito nella dichiarazione, evitando qualsiasi ambiguità.

XML Schema

Come già accennato in precedenza, i DTD hanno alcune caratteristiche che fanno di loro un meccanismo non perfettamente soddisfacente. Queste caratteristiche sono così riassumibili:

- La loro sintassi non è XML, rendendoli non processabili dal parser, facendo quindi perdere la possibilità di creare DTD ben formati e validi;
- La loro definizione permette di specificare pochi tipi di dati predefiniti;
- Non sono modulari (non è facile ricavare una parte di un DTD);
- Non sono facilmente estensibili (non esiste un meccanismo che permetta l'ereditarietà).

Nel 1998 è stato avviato il lavoro sugli XML Schema, in modo da creare un meccanismo analogo ai DTD ma che elimini i loro difetti. Attualmente le specifiche degli XML Schema (in corso di standardizzazione presso il W3C) sono suddivise in tre distinti documenti che riguardano nell'ordine, una generica descrizione degli schemi, la specifica per la definizione delle strutture, la specifica per la definizione dei tipi di dati. Nella pratica, gli Schemi permettono, come i DTD, di definire la struttura dell'istanza di un documento XML e di specificare il tipo di dato di ciascun elemento o attributo. Il loro vantaggio è di costruire a loro volta un documento XML, di definire oltre quaranta tipi di dato contro la decina permessa dai DTD, di essere orientati agli oggetti, permettere di definire dei tipi di dati complessi, stabilire dei vincoli sul contenuto degli elementi e dei loro attributi. Uno XML Schema si riconosce perché il suo primo elemento è, per convenzione, l'elemento `<schema>` cui viene associato solitamente il namespace di default, come nella riga seguente:

```
<xsd: schema xmlns:xsd="http: //www.w3c.org/2001/XML Schema">
```

All'interno di uno Schema si possono inserire, in varia forma ed ordine, i seguenti elementi:

- `<import>` e `<include>` per inserire altri schema, o loro frammenti, da altri documenti;
- `<element>` e `<attribute>` per la definizione di elementi ed attributi globali del documento;
- `<simpleType>` e `<complexType>` per la definizione di tipi denominati utilizzabili in seguito;

- `<attributeGroup>` e `<group>` per definire serie di attributi e gruppi di modelli di contenuto complessi;
- `<notation>` per definire annotazioni non XML all'interno di un documento XML;
- `<annotation>` per esprimere commenti comprensibili ad un essere umano.

Per esprimere vincoli sul contenuto di attributi ed elementi, gli Schema ricorrono ai tipi, che si suddividono in:

- tipo semplice: una sequenza di caratteri che non può contenere markup ed avere attributi;
- tipo complesso: può contenere markup ed avere attributi.

Ogni tipo è caratterizzato da alcune proprietà, dette *facets*, che ne descrivono vincoli e formati. Estendendo o restringendo queste caratteristiche, è possibile derivare dei nuovi tipi. I facets rappresentano delle caratteristiche indipendenti tra loro che specificano aspetti come, per esempio:

- length, minLength, maxLength: numero richiesto, minimo e massimo di caratteri;
- precision, scale: numero di cifre significative e di decimali significativi;
- pattern: espressione regolare che il valore deve soddisfare;
- enumeration: lista all'interno della quale scegliere il valore.

XSL e XSLT: trasformare e formattare i documenti XML

XML permette di definire la struttura di un'insieme di dati, ma non comprende nessuna informazione circa il modo in cui tali dati debbano essere visualizzati.

L' aXtensible Stylesheet Language (XSL) è una delle tante applicazioni di XML, e permette di applicare dei fogli di stile ad un documento XML al fine di fornigli una veste grafica per la pubblicazione in un qualche formato (HTML è il più usato, ma potrebbe essere anche WML, PDF o altro). Permette, inoltre, di trasformare i documenti XML in documenti con una diversa struttura logica. XSL si compone di tre parti distinte:

- un vocabolario XML per specificare come formattare un documento;
- un linguaggio per trasformare un documento (XSL Transformations, XSLT);

- un linguaggio per accedere o far riferimento ad una parte di un documento XML (XML Path Language, Xpath).

Modelli di sviluppo di applicazioni XML

Con il consolidamento di XML come strumento per l'interscambio di informazioni tra applicazioni e di java come linguaggio di programmazione general-purpose si è resa necessaria la creazione di strumenti, nella fattispecie di parser XML sotto forma di API, che permettessero la lettura e l'analisi di documenti XML.

Un parser XML è un modulo software che si colloca tra l'applicazione e il documento XML. Esso permette all'applicazione di accedere al contenuto e alla struttura del documento XML. Esistono due tipi di parser: validanti e non validanti. I primi, oltre a controllare se un documento è ben-formato, cioè che ogni elemento sia racchiuso tra due tag (uno di apertura e uno di chiusura), controlla pure se esso è un documento XML valido, cioè se è fedele alle regole definite nella sua DTD. I parser non validanti, invece, si preoccupano solo di vedere se un documento è ben formato.

Attualmente esistono due principali tipologie di parser: DOM-based e SAX-based.
La struttura DOM (descritta in precedenza) è stata sviluppata per ambienti Object-Oriented da un punto di vista statico.
Le SAX (*Simple API for XML*), sono alternative alle prime e sviluppate da un punto di vista dinamico (*event driven*).
Esistono diverse implementazioni dei due modelli in diversi linguaggi di programmazione e script: Java, C++, Python, JavaScript, Perl ...
In genere le implementazioni SAX sono più performanti delle DOM e anche più leggere, ovvero minore occupazione di memoria (memory overhead).
Le operazioni messe a disposizione dalle classi basate su DOM sono di lettura-modifica-creazione-scrittura per ciascuna tipologia di nodi. Per esempio la classe *Document* mette a disposizione funzioni come *getElementsByTagName(*tag_name*)* per la lettura e *createElement(*name*)*, *createAttribute(*name*)* e *createTextNode(*name*)* per la creazione.
Le classi basate su SAX invece forniscono metodi che sono invocati in fase di parsing ad

ogni evento di riconoscimento (inizio-fine documento, tag etc...) e che devono essere necessariamente sovrascritti.

Il DOM, pur essendo molto utile e ampiamente disponibile, ha un grande limite: il file XML deve essere completamente caricato in memoria, prima che vi si possa effettuare qualunque operazione (vi è in realtà un'operazione di caricamento asincrono, ma essa semplicemente permette di iniziare a lavorare con alcuni nodi prima che il documento sia caricato completamente). Questo è in molti casi indesiderabile, per ragioni di efficienza, e in alcuni casi, in cui il file XML è troppo grande per le risorse disponibili, può addirittura impedire all'applicazione di manipolarlo.

SAX 2.0 cerca di ovviare a questo inconveniente proponendo modelli ad eventi: invece di caricare il documento e poi navigare nel DOM costruito dal parser, l'applicazione può registrare delle callback function con il Parser SAX, che verranno invocate quando specifici eventi si verificano. Un tipico evento durante il parsing di un documento XML è la lettura di uno specifico tag da parte del parser. In questo modo, l'applicazione è invocata solo quando il parser sta processando dati che la interessano, e non c'è necessità che l'intero documento sia completamente contenuto in memoria.

Uno dei principali problemi di questo modello ad eventi (push) è la difficoltà di mantenere uno stato globale per l'applicazione: essa è frammentata in tanti event-handler (o callback function) che operano in modo asincrono (o meglio, data driven) l'uno dall'altro. È necessario studiare attentamente delle strutture dati globali in cui lo stato del parsing è rappresentato, e dove ciascun event-handler legge e scrive le informazioni di sua pertinenza.

Modelli più flessibili di DOM2 e SAX, e che supportano anche tecniche pull (controllo da parte dell'applicazione, pur non caricando l'intero file in memoria) sono disponibili in piattaforme avanzate come Java e il recente C#/.NET, ma al momento non mi risulta ne esista qualcuno standard.

XML e RPC

RPC è una semplice estensione del concetto di chiamata di procedura, consente cioè l’interazione tra procedure di applicazioni diverse oppure che girano su macchine distinte.

Concettualmente non c'è differenza tra una chiamata di procedura locale ed un chiamata di procedura remota, quest'ultima però richiede un'implementazione diversa, ha prestazioni diverse (RPC è più lenta) e quindi è usata per applicazioni di tipo diverso.

Le chiamate remote sono serializzate in un formato comprensibile da entrambe le parti che comunicano; un particolare approccio consente la comunicazione tra macchine Windows e macchine Unix.

Il protocollo XML-RPC definisce una modalità attraverso la quale le risposte e le richieste RPC sono serializzate in documenti XML e inviate attraverso una connessione HTTP. XML-RPC offre un semplice approccio per la creazione di un elenco di parametri per una chiamata di metodo. Ogni parametro comprende un elemento che descrive il tipo di parametro ed il reale contenuto dell'elemento.

L'importanza di RPC non è nel formato a cui si appoggia, che può essere liberamente scelto, ma nella modularità delle procedure software e nella loro gestione. Inoltre le procedure XML-RPC possono essere implementate con diversi linguaggi di programmazione quali C/C++, Java, etc.

Un tipico pacchetto XML-RPC contiene un header, in cui si specificano informazioni inerenti il trasferimento del messaggio, ed un body, in formato XML, contenente il messaggio vero e proprio. Negli esempi seguenti analizzeremo un esempio di richiesta XML-RPC, un esempio di risposta ed un esempio di segnalazione di errore.

Consideriamo un esempio di richiesta XML-RPC:

LISTATO 3.2 ESEMPIO DI RICHIESTA XML-RPC

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml                                    HEADER
Content-Length: 181
<?xml version="1.0"?>
<methodCall>
```

```
<methodName>examples.getStateName</methodName>
        <params>
            <param>                                                    PAYLOAD
                <value><i4>41</i4></value>
            </param>
        </params>
</methodCall>
```

Informazioni contenute nell'header

Nella prima riga dell'header non è necessario specificare il formato dell'URI, che può contenere informazioni circa l'instradamento della richiesta (nell'esempio /RPC2), oppure essere vuoto; mentre sono obbligatori i campi `User-Agent` ed `Host`; il campo `Content-Type` è di tipo text/xml (il payload contiene testo e, in particolare, xml) e il campo `Content-Length` deve essere specificato in modo corretto ed indica la lunghezza del payload.

Formato del payload

In XML il payload è una singola struttura `<methodCall>`, che deve contenere un sotto-elemento `<methodName>` e qualora la procedura richiedesse dei parametri la `<methodCall>` deve contenere il sotto-elemento `<params>` che a sua volta può contenere un qualsiasi numero di `<param>` ciascuno con un proprio `<value>`. I parametri passati alla procedura possono essere anche strutture dati di tipo complesso.

Consideriamo un esempio di risposta XML-RPC:

LISTATO 3.3 ESEMPIO DI RISPOSTA XML-RPC

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
```

```
Server: UserLand Frontier/5.1.2-WinNT


<?xml version="1.0"?>
<methodResponse>
    <params>
        <param>                                                    BODY
            <value><string>South Dakota</string></value>
        </param>
    </params>
</methodResponse>
```

Informazioni sul formato di risposta

Se non ci sono errori di basso livello allora viene sempre restituito 200 OK.

Il campo `Content-Type` può essere di tipo text/xml; il campo `Content-Length` deve essere presente e corretto.

Il Body della risposta è una singola struttura xml, una `<methodResponse>` che può contenere un sotto-elemento `<params>`, che a sua volta contiene un unico `<param>` ed il suo corrispondente `<value>`.

Il valore restituito dalla procedura eseguita sul server remoto viene formattato in xml.

Consideriamo un esempio di risposta in cui compare un errore:

LISTATO 3.4 ESEMPIO DI RISPOSTA CON ERRORE

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT
```

```
 <?xml version="1.0"?>
<methodResponse>
    <fault>
        <value>
            <struct>
                <member>
                    <name>faultCode</name>
                    <value><int>4</int></value>
                </member>
                <member>
                    <name>faultString</name>
                    <value><string>Too many parameters.</string></value>
                </member>
            </struct>
        </value>
    </fault>
</methodResponse>
```

In questo caso la `<methodResponse>` contiene il sotto-elemento `<fault>` che contiene un `<value>` che è una `<struct>` contenente due elementi uno chiamato `<faultCode>` (di tipo int) e uno chiamato `<faultString>` (di tipo string).

Una `<methodResponse>` non può contenere sia l'elemento `<fault>`, sia l'elemento `<params>`.

Vantaggi e svantaggi di XML-RPC

Lo scopo principale di XML-RPC è di consentire la comunicazione tra ambienti diversi anche non compatibili tra loro; questo avviene grazie ad un protocollo in grado di adattarsi velocemente a sistemi operativi diversi.

Sebbene XML-RPC fornisca una sintassi flessibile e di semplice implementazione, se paragonata alla specifica SOAP quest'ultima risulta di più semplice utilizzo in quanto offre un livello più alto di estensibilità per sistemi orientati a oggetti.

Capitolo 4 – SOAP E HTTP

Nelle specifiche della versione 1.1 il solo protocollo per cui vengono date delle direttive per trasportare i messaggi SOAP è HTTP, mentre nell'ultima versione si utilizza anche FTP ed altri standard. Per meglio comprendere la natura di questo protocollo, viene di seguito fornito l'esempio di un messaggio SOAP, incorporato in una richiesta HTTP di un client, e (per completezza) la relativa risposta da parte del server. Si fa osservare, a questo proposito, che quando utilizzato in combinazione con HTTP, SOAP ne "eredita" il paradigma della richiesta/risposta.

LISTATO 4.1 RICHIESTA SOAP SU HTTP

```
POST / StockQuote HTTP/ 1.1
Host: www.stockquoteserver.com
Content-Type: application/soap; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope
xmls: SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding//>
    <SOAP-ENV:Header>
        <t: Transaction xmlns:t="some-URI" SOAP:mustUnderstand="1">
                5
        </t:Transaction>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
       <m:GetLastTradePrice xmlns:m="Some-URI">
            <symbol>DEF</symbol>
       </m:GetLastTradePrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

LISTATO 4.2 RISPOSTA SOAP SU HTTP

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=”utf-8”
Content-Length: nnnn
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
SOAP-ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”/>
  <SOAP-ENV:Body>
      <m:GetLastTradePrice xmlns:m=”Some-URI”>
          <Price>34.5</Price>
      </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Le prime cinque righe del messaggio di richiesta riportato nel Listato 4.1 sono proprie del protocollo di trasporto. La seconda parte è un documento XML, codificato secondo SOAP, e costituisce quello che viene chiamato il messaggio SOAP.

HTTP come mezzo di trasporto per SOAP

L’Hyper Text Transfer Protocol (HTTP) è un protocollo a livello delle applicazioni che permette lo scambio di informazioni ipertestuali, basato su testo (ASCII). Il protocollo si basa sul modello della richiesta/risposta e trasmette informazioni sottoforma di messaggi, costituiti da headers e da un body. Quando è utilizzato come veicolo per i messaggi SOAP solo alcuni di questi header sono adoperati. Le prime cinque righe del Listato 4.1 mostrano gli header HTTP utilizzati per trasportare un messaggio SOAP.

```
POST/ StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: application/soap; charset=”utf-8”
Content-Lenght: nnn
```

```
SOAPAction: “Some-URI”
```

Per essere più precisi, le prime quattro righe appartengono agli standard del protocollo, mentre la quantità è propria di SOAP. La prima riga rappresenta, nell’ordine, il metodo della richiesta (POST), il request-URI (/StockQuote) e la versione utilizzata del protocollo (HTTP/1.1). Il metodo di richiesta da utilizzare in combinazione con SOAP deve essere POST, poiché permette (a differenza di GET) di trasferire un blocco di informazioni nel corpo della richiesta, quindi di allegare il messaggio SOAP. La seconda riga (Host) indica qual’ è il server oggetto della richiesta. La terza e quarta riga specificano rispettivamente il tipo MIME (Multi_Purpose Iternet Mail Extension) e la lunghezza della richiesta. SOAP, dalla specifica 1.2, richiede che il content-type sia “application/soap”. Infine seguono eventuali header personalizzati come SOAPAction. Gli header personalizzati vengono trasmessi in coda agli header propri del protocollo di trasporto e di conseguenza viene ricevuto ed elaborato dal server. La SoapAction, obbligatoria nella versione 1.1 e facoltativa nella 1.2) viene utilizzata come chiave di accesso ai firewall. E’ anche possibile riportare come SOAPAction una stringa vuota (“”) per indicare che deve essere utilizzato come obiettivo della chiamata la Request-URI HTTP. Se invece questo header è omesso, significa che non è specificato l’intento della chiamata.

Per quanto concerne la parte HTTP della risposta, SOAP non introduce nessuna peculiarità. Esso segue la semantica del protocollo di trasporto, utilizzando i codici di stato. Come visto nell’esempio, la risposta del server sarà:

```
HTTP/1.1 200 OK

Content_Type: application/soap; charset=”utf-8”

Content-Lenght: nnn
```

Per indicare che la richiesta ha avuto una risposta positiva. Nel caso di errori a livello del protocollo SOAP, il server risponderà con lo stato 500 (“Internal server error”) e include nel messaggio SOAP la codifica dell’errore.

GET E POST

Consideriamo ora una tipica richiesta HTTP per la trasmissione di dati e la corrispondente soluzione fornita dal protocollo SOAP.

Il metodo *GET* è la richiesta HTTP più utilizzata: esso fornisce al server le informazioni necessarie per identificare una risorsa particolare, ricercare i dati ed inviarli al client. In particolare l’URL può contenere una stringa query costituita da coppie nome-valore delimitate e comunica al server i dati

specifici delle applicazioni. Questo meccanismo è estremamente utile per richieste HTTP semplici, ma nel momento in cui una richiesta diventa più complicata la stringa query è più difficile da gestire.

Per trasmettere dati con protocollo SOAP è consigliabile usare il metodo *POST* che trasmette il contenuto della richiesta nel corpo del messaggio di richiesta HTTP piuttosto che sull'URL.

Consideriamo un esempio:

LISTATO 4.3 ESEMPIO METODO POST

```
POST /PartServer.pl HTTP/1.1
Host: www.mpc.com
Accept: text/*
Content-type: application/soap
Content-lenght: nnnn
SOAPAction: the-method-uri#FindPart
{CR}{LF}
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:FindPart xmlns:m="the-method-uri">
            <PartNo>12345</PartNo>
            <GroupID>7</GroupID>
        </m:FindPart>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Il campo `SOAPAction` è stato aggiunto all'intestazione HTTP come campo obbligatorio per fornire informazioni sullo scopo del payload SOAP (in genere una chiamata ad un metodo), in modo che i

firewall (o un elemento equivalente) possano elaborare preliminarmente l'intestazione HTTP per assicurare l'autorizzazione della chiamata.

Se l'intestazione HTTP soddisfa i requisiti dei firewall la chiamata SOAP può essere inoltrata al server appropriato, dove i contenuti di `SOAPAction` vengono confrontati con il nome del metodo associato e con l'uri dello spazio dei nomi all'interno del payload.

Nel nostro esempio, se il server SOAP è in grado di elaborare la richiesta, viene inviata al client la seguente risposta HTTP:

LISTATO 4.4 RISPOSTA HTTP

```
HTTP/1.1 200 OK
Content-type: application/soap
Content-lenght: nnnn
SOAPAction: the-method-uri#FindPart
{CR}{LF}
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <!- The method response or fault ->
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Indipendentemente dall'esito della chiamata di metodo la risposta HTTP deve essere `HTTP/1.1 200 OK`, in tal modo al client viene comunicato che l'errore è nella chiamata e non nel livello di comunicazione.

Capitolo 5 – SOAP: specifica 1.2 del W3C

La specifica 1.1 viene rilasciata dall'IETF nel Settembre 1999 e sottoposta, nel maggio 2000, all'esame del W3C per essere standardizzata: dopo varie Working Draft, per cercare di eliminare le ambiguità e le ripetizioni della precedente versone, il 24 Giugno 2003 il W3C rilascia la Recommendation di SOAP Versione 1.2, costituita da SOAP Version 1.2 Primer, SOAP Version 1.2 Messaging Framework, SOAP Version 1.2 Adjuncts e SOAP Version 1.2 Specification Assertions and Test Collection.
SOAP Version 1.2 è un protocollo leggero studiato per lo scambio di informazione strutturata in un ambiente decentralizzato e distribuito come il Web. Una W3C Recommendation è l' equivalente di uno standard Web, in quanto indica che la specifica sviluppata dal W3C è stabile, contribuisce alla interoperabilità del Web ed è stata rivista dai membri W3C, che ne favoriscono l' adozione da parte dell' industria.

SOAP Versione 1.2 integra tecnologie di base di XML. SOAP Versione 1.2 è progettato per lavorare senza soluzione di continuità con W3C XML schema, massimizzando l' utilità di SOAP con una vasta gamma di strumenti XML e preparando la strada per lavori futuri sul Web Services Description Language (WSDL). Inoltre fa uso dei Namespaces in XML come meccanismo flessibile e leggero per trattare la composizione di linguaggi XML.

SOAP Versione 1.2 descrive un modello di elaborazione migliorato, rimuovendo così ambiguità presenti in SOAP 1.1. SOAP Versione 1.2 comprende messaggi di errore più chiari che aiuteranno gli sviluppatori a scrivere applicazioni di miglior qualità.

La versione 1.2, per nome dello stesso Berners-Lee, Direttore del W3C, punta a migliorare l'adozione di SOAP come protocollo universale per lo scambio dati. Le specifiche sono separate in due documenti: il messaging framework (http://www.w3.org/TR/2003/PR-soap12-part1-20030507/) e le aggiunte (http://www.w3.org/TR/2003/PR-soap12-part2-20030507/).

Un terzo documento funge da introduzione (http://www.w3.org/TR/2003/PR-soap12-part0-20030507/) a chi non conosce SOAP 1.1

La press release (http://www.w3.org/2003/05/soap12-pressrelease.html.en) del W3C indica anche che sono sette (http://www.w3.org/2000/xp/Group/2/03/soap1.2implementation.html) le

implementazioni già pronte della specifica. SUN non è ancora presente, ma Axis ne ha un supporto parziale.

Le specifiche di messaging definiscono il modello di elaborazione, migliorato rispetto alla versione 1.1 (e quindi meno ambiguo), un modello di estensibilità, la struttura del messaggio, ed il collegamento a protocolli fisici come HTTP.
Le aggiunte definiscono un insieme di elementi ulteriori, come RPC, l'encoding ed il binding con HTTP 1.1.

Importante l'integrazione di tecnologie fondamentali di XML, come XML Schema ed i namespace, con conseguente miglior supporto da parte degli strumenti di sviluppo. Questa revisione aggiunge una maggiore chiarezza e robustezza delle specifiche, unitamente ad una indipendenza e miglior utilizzo di HTTP. Forse il 90% delle modifiche di 1.2 sono sintattiche, con cambio di namespace, mentre il modello di richiesta/risposta rimane il medesimo di 1.1.

Il gruppo di lavoro ha risolto 400 problemi, tra cui 150 legati a SOAP 1.1, che è composto dai maggiori player del settore: IBM, Microsoft, Sun Microsystems, and BEA Systems che si riuniscono come *W3C XML Protocol Working Group* (http://www.w3.org/2000/xp/Group/).

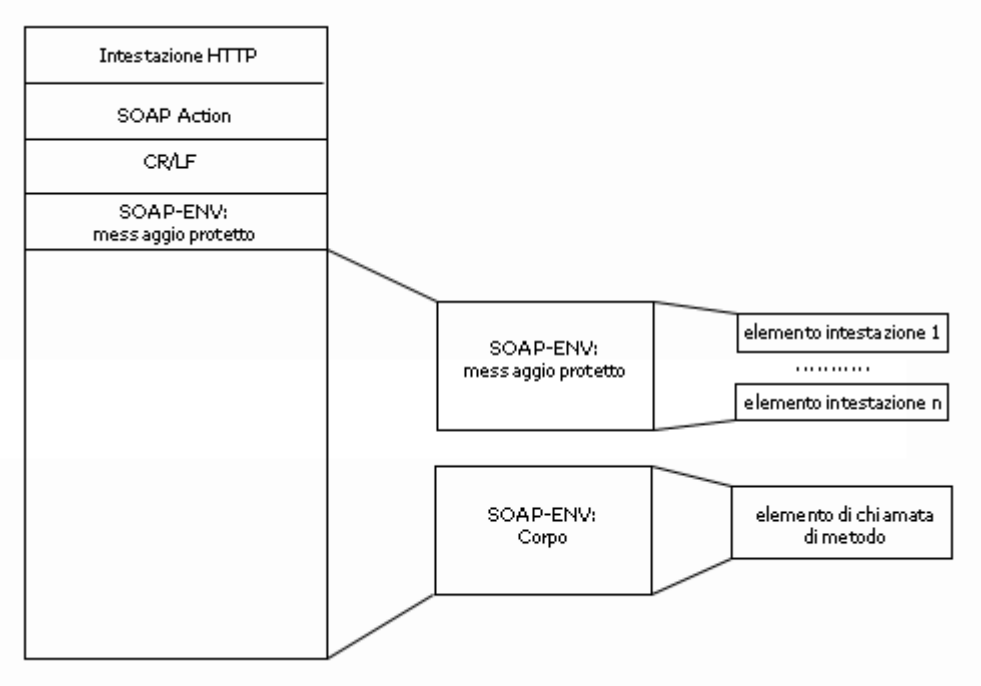
Messaggi SOAP

Abbiamo detto SOAP è basato su XML. In effetti, un messaggio SOAP non è altro che un documento XML che descrive una richiesta di elaborazione o il risultato di una elaborazione, dove le informazioni contenute nella richiesta e nella risposta vengono serializzate secondo un formato prestabilito, utilizzando XML come strumento che garantisce l’indipendenza dalla piattaforma.

SOAP consiste di tre parti: una busta, che definisce un framework, per la descrizione del contenuto del messaggio e della modalità di elaborazione (SOAP envelope construct) , un insieme di regole di codifica per l’espressione di istanze di tipi di dati definiti dalle applicazioni (SOAP encoding)ed una convenzione per la rappresentazione di chiamate e risposte di una procedura remota(SOAP RPC).

SOAP necessita di un protocollo di trasporto: nella sua versione iniziale si appoggiava all’ HTTP standard mentre nelle ultime versioni utilizza anche FTP e altri standard.

Figura 5.1: Layout del pacchetto SOAP



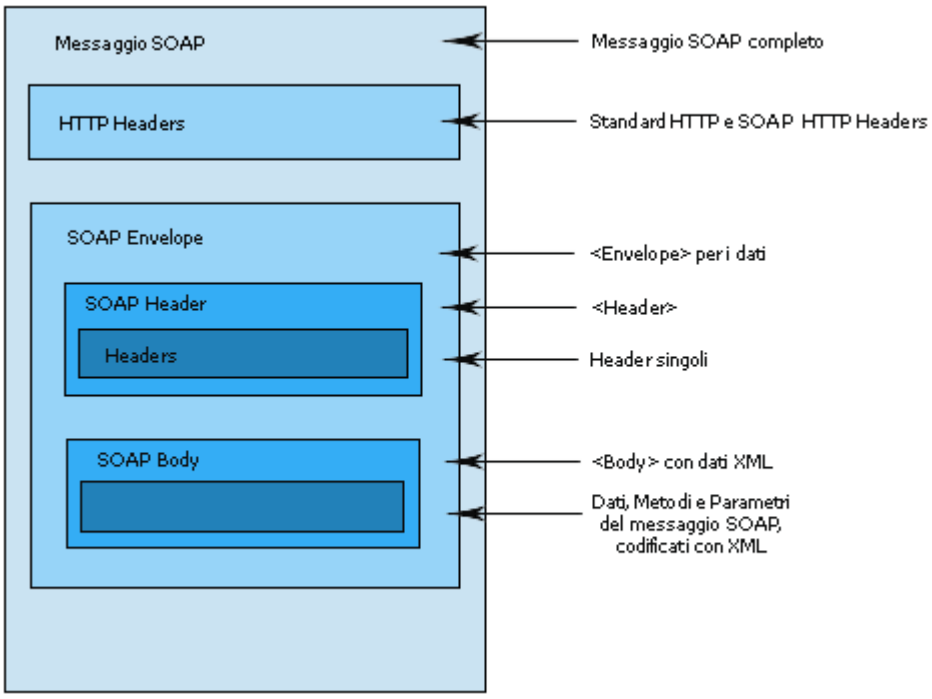
HTTP supporta diversi modi per la richiesta di informazioni mediante un'intestazione di richiesta; utilizza cioè dei metodi per descrivere le intenzioni del server oppure alcuni campi dell'intestazione per includere le coppie nome-valore dei dati di richiesta. Quando il server risponde genera un messaggio costituito da un'intestazione di risposta che include una riga di stato e i campi dell'intestazione per includere coppie nome-valore dei dati di risposta. Metodi e intestazioni forniscono un framework flessibile, semplice e succinto.

Il protocollo SOAP è adatto a supportare un'architettura client-server: i dati richiesti ed elaborati fra client e server sono organizzati in messaggi SOAP e vengono trasportati attraverso il protocollo HTTP o un altro protocollo di trasporto. I messaggi SOAP sono fondamentalmente trasmissioni unidirezionali da un mittente ad un destinatario ma sono spesso combinati per implementare modelli del tipo richiesta/risposta.

Un vantaggio notevole offerto da SOAP consiste nell'imposizione di una struttura di dati per la richiesta e, quando disponibili, per la risposta.

Nella trattazione seguente proporremo alcuni semplici esempi attraverso i quali introdurremo la sintassi di SOAP ed alcune caratteristiche dei messaggi SOAP, che si possono schematizzare come in figura:

Figura 5.2: Caratteristiche di un messaggio SOAP



Si può osservare immediatamente che un messaggio SOAP si compone di alcune parti peculiari, quali un'intestazione, un corpo ed una richiesta di metodo; consideriamo ora un esempio di richiesta SOAP, ossia vediamo come un client può richiedere un servizio ad un server.

```
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:Add xmlns:m="Sum-URI">
            <Param1>3</Param1>
            <Param2>4</param2>
        </m:Add>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Lo scambio di messaggi

I messaggi SOAP sono una trasmissione da un mittente a un ricevente anche se spesso, come visto, si adattano al modello di richiesta-risposta. Si esaminerà adesso, con un esempio, cosa succede quando due applicazioni scambiano messaggi utilizzando SOAP su HTTP. HTTP supporta diversi modi per la richiesta di informazioni mediante un'intestazione di richiesta; utilizza

cioè dei metodi per descrivere le intenzioni del server oppure alcuni campi dell'intestazione per includere le coppie nome-valore dei dati di richiesta. Quando il server risponde genera un messaggio costituito da un'intestazione di risposta che include una riga di stato e i campi dell'intestazione per includere coppie nome-valore dei dati di risposta. Metodi e intestazioni forniscono un framework flessibile, semplice e succinto.

L'esempio mette in luce, fra le altre cose, come SOAP sia uno strato in un'architettura multi-strato di un sistema. Facendo riferimento alla Figura 2.3, si ha che:

1. L'applicazione esegue un comando che genera un'azione nell'application server;
2. Il comando genera un processo nell'applicazione ne il risultato arriva all'interfaccia dell'applicazione;
3. Il messaggio è trasformato in XML e quindi inviato al web server;
4. Il parser XML controlla che il documento XML sia un messaggio SOAP, quindi lo invia tramite HTTP;
5. Il parser dell'applicazione remota controlla la validità del messaggio usando gli header HTTP e XML, e lo accetta o lo rifiuta di conseguenza;
6. Il messaggio è inoltrato all'applicazione, la quale lo deve trasformare (deserializzare) per poterlo utilizzare. L'applicazione deve quindi eseguire i seguenti passi:

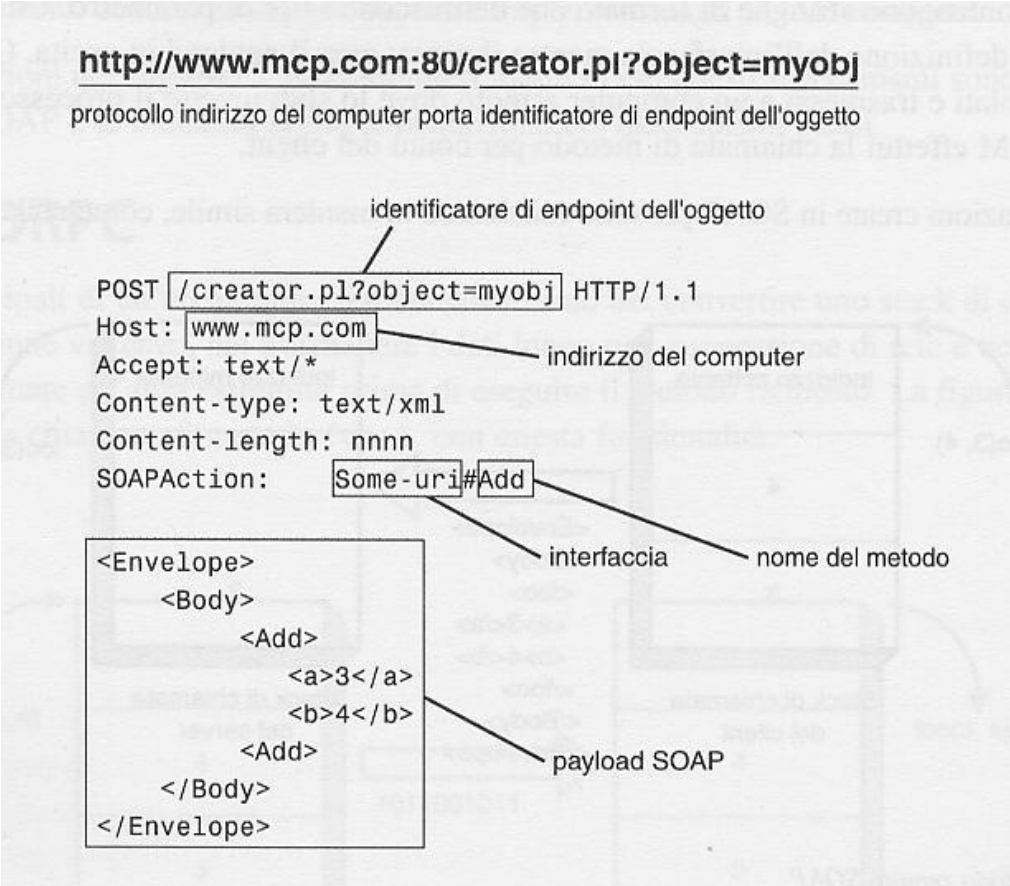
    - Identificare le parti del messaggio SOAP relative all'applicazione;
    - Controllare che tutte le parti obbligatorie possano essere processate, altrimenti segnalare l'errore;
    - Rimuovere tutte le parti riguardanti l'applicazione se non è destinataria ultima del messaggio (cioè se è un'intermediaria);

7. L'applicazione esegue il compito e produce un risultato;
8. Il risultato è ritornato al mittente con lo stesso iter eseguito dal messaggio di richiesta;
9. L'applicazione richiedente può utilizzare il risultato in svariate maniere: mostrare dei dati a video, eseguire altre azioni, accedere ad un database, ecc.

Invocazione di metodo

Per eseguire una chiamata di metodo SOAP il payload SOAP deve essere costruito con un elemento principale `<SOAP:Envelope>` che contiene un elemento opzionale `<SOAP:Header>` e uno obbligatorio `<SOAP:Body>`.

Vediamo qualche elemento di un messaggio SOAP



L'elemento `<SOAP:Header>` è utilizzato per incapsulare informazioni estese relative alla chiamata.

L'elemento `<SOAP:Body>` contiene la chiamata di metodo reale e i parametri del metodo associato sia con elementi incorporati sia con elementi indipendenti.

Risposta di metodo

Nel momento in cui è stata richiesta una chiamata di metodo, il server ha diverse opzioni da seguire: è possibile che venga restituito un codice di errore oppure il risultato dell'esecuzione del metodo; in entrambi i casi si utilizza la sintassi di payload SOAP standard.

Consideriamo un esempio di segnalazione di errore SOAP:

```
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>300</faultcode>
```

```
            <faultstring>Invalid Request</faultstring>
            <runcode>1</runcode>
        </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

L'elemento `Fault` è utilizzato per inserire informazioni sugli errori e/o sullo stato di un messaggio SOAP. Esso definisce alcuni elementi secondari, tra cui l'elemento `faultcode` che viene utilizzato dal SW per fornire un algoritmo di identificazione degli errori. I valori di faultcode SOAP predefiniti sono definiti in modo estensibile per consentire la definizione di nuovi valori faultcode SOAP compatibile con quelli esistenti.

In generale un errore SOAP consta di tre elementi obbligatori: `faultcode`, `faultstring`, `runcode` e di un elemento `detail` opzionale.

Messaggi SOAP: serializzazione dei dati

In particolare, un messaggio SOAP è costituito dai seguenti elementi:

- *Envelope* (obbligatotio)
  Rappresenta il contenitore del messaggio e costituisce l'elemento root del documento XML
- *Header* (opzionale)
  È un elemento opzionale che contiene informazioni globali sul messaggio; ad esempio, nell'header potrebbe essere specificata la lingua di riferimento del messaggio, la data dell'invio, ecc.
- *Body* (obbligatorio)
  Rappresenta la richiesta di elaborazione o la risposta derivata da una elaborazione
- *Fault* (opzionale)
  Se presente, fornisce informazioni sugli errori che si sono verificati durante l'elaborazione; questo elemento può essere presente soltanto nei messaggi di risposta.

È opportuno evidenziare che SOAP definisce soltanto la struttura dei messaggi non il loro contenuto. I tag per descrivere una richiesta di elaborazione o un risultato vengono definiti in uno schema specifico ed utilizzati all'interno della struttura SOAP sfruttando il meccanismo dei *namespace*. Prima di analizzare nel dettaglio la composizione del messaggio, ci sono alcuni aspetti nell'utilizzo che SOAP fa di XML che è opportuno evidenziare. Innanzitutto, un messaggio non può contenere né DTD, né Process Instruction (PI). Per descrivere i tipi di dati sono utilizzati gli XML Schema. In secondo luogo, SOAP ricorre ai namespace per evitare

ambiguità nei messaggi, ed ogni elemento di un messaggio creato da un'applicazione può essere qualificato da un opportuno namespace. Comunque, il protocollo ne definisce due di propri:

- http://schema:xmlsoap.org/soap/envelope/, per la envelope;
- http://schema:xmlsoap.org/soap/encoding/, per il meccanismo di serializzazione.

In un messaggio SOAP, ogni elemento appartiene ad una delle seguenti categorie:

- elementi strutturali;
- elementi radice;
- elementi accessor;
- elementi indipendenti.

Gli *elementi strutturali*, come appunto indica il nome, costituiscono la struttura del messaggio e sono esattamente quelli sopra descritti (SOAP-ENV: Envelope, SOAP-ENV: Header e SOAP_ENV: Body). Il modo in cui si compongono per formare il messaggio è decritto dallo schema seguente (una parte dello schema "http://schema.xmlsoap.org/soap/envelope/"):

```
<schema xmlns='http://www.w3.org/1999/XMLSchema'
                xmlns:tns='http://schemas.xmlsoap.org/soap/envelope/'
                targetNamespace='http:// schemas.xmlsoap.org/soap/envelope/'>
  <! -- SOAP envelope, header and body -- >
<element name="Envelope" type="tns:Envelope"/>
<complexType name='Envelope'>
    <element ref='tns:Header' minOccurs='0'/>
    <element ref='tns:Body' minOccurs='1'/>
    <any minOccurs='0' maxOccurs='*'/>
    <anyAttribute/>
</complexType>
```

L'envelope è il primo elemento del documento XML, e rappresenta l'intero messaggio. L'header è opzionale, ma se presente deve fare seguito all'envelope, e serve per poter estendere il messaggio. Il body, obbligatorio, è il contenitore dove sono inserite le informazioni che si devono inviare o ricevere e che tipicamente riguardano l'invocazione di metodi remoti o condizioni d'errore. Dei quattro elementi che costituiscono un messaggio, quelli strutturali sono i soli che non sono utilizzati per rappresentare istanze di tipi di dati.

Un *elemento radice* è un immediato discendente di uno dei due elementi strutturali, SOAP-ENV:Body o SOAP-ENV:Header. SOAP-ENV:Body può avere esattamente un solo elemento radice che costituisce la chiamata, la risposta o un errore. SOAP-ENV: Header, invece, può avere più elementi radici, uno per ogni estensione associata al messaggio. Gli *elementi accessor* sono usati per rappresentare dati, proprietà o campi di un oggetto. Ad ogni campo di un oggetto, infatti, corrisponde un elemento accessor, il cui nome del tag che lo rappresenta è lo stesso del nome del campo. Gli *elementi indipendenti* rappresentano istanze di tipi che sono referenziate, in modo indiretto, da altri elementi accessor.

Envelope

L'envelope è l'elemento che racchiude l'intero messaggio. Osservando l'esempio seguente si nota che all'envelope sono associati un namespace ed un attributo:

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
```

Il namespace serve per distinguere la versione di SOAP che si sta utilizzando. Se un'applicazione riceve un messaggio il cui elemento envelope è associato ad un namespace differente, deve rispondere con un messaggio di errore. L'attributo `encodingStyle` permette di specificare una lista ordinata di URI che identificano le regole di serializzazione utilizzate nella codifica e decodifica del messaggio. Il protocollo permette alle applicazioni di adoperare la codifica più consona e non prevede nessuna regola di default. Esiste però un namespace che fa riferimento ad uno schema standard cui si può ricorrere qualora non si abbia necessità di regole specifiche, ed è quello già citato in precedenza.

Header

L'header di un messaggio SOAP è opzionale. E' un meccanismo pensato per permettere di estendere un messaggio in maniera flessibile e decentralizzata. Può essere usato, per esempio, per implementare sistemi di autenticazione, gestione delle transazioni, sistemi per il pagamento. Ad ogni suo elemento deve essere associato un preciso namespace, ed eventualmente, si può ricorrere all'attributo encodingStyle per specificare una lista ordinata di URI che identificanole regole di serializzazione adoperate nella sua codifica-decodofoca. La lista è ordinata dalle regole più specifiche fino a quelle più generali. Altri attributi definiti nelle specifiche del protocollo sono `mustUnderstand` e `actor`. Il primo serve (quando assume come valore "true" o 1) a fare obbligatoriamente processare al destinatario del messaggio l'header cui appartiene. Cioè, se un header è essenziale per la corretta comprensione del messaggio,

esso deve avere l'attributo mustUnderstand settato a 1. L'attributo actor, invece, serve ad indicare, sottoforma di URI, chi è da considerarsi il destinatario dell'header cui appartiene. Questo è dovuto al fatto che un messaggio SOAP può viaggiare dal mittente al ricevente passando attraverso una serie di intermediari, ovvero delle applicazioni SOAP che sono anch'esse in grado di ricevere e spedire messaggi. Quindi, non tutte le parti di un messaggio possono essere intese per il destinatario finale; alcune informazioni possono riguardare un intermediario, specificato con l'attributo actor, il quale deve trattenere l'informazione che lo riguarda e inoltrare il resto del messaggio. Ulteriori attributi possono essere aggiunti dalle applicazioni. Oppure l'attributo actor, dalla specifica 1.2, può assumere uno dei seguenti valori:

- *none*: l'header in questione non deve essere elaborato da nessun nodo soap
- *next*: l'header in questione può essere elaborato da tutti i nodi incontrati nel tragitto mittente-ricevente
- *anonymous*: l'header in questione può essere elaborato solo dal nodo finale.Tale valore non viene espresso esplicitamente, ma viene assunto quando nell'header non compare l'attributo actor.

Un altro attributo che può essere utilizzato in un sottoelemento dell'header è `root`. Questo attributo ha come obiettivo quello di far predominare l'elemento in questione sugli altri presenti nella stessa struttura, così che il parser di destinazione possa discriminare sull'importanza da assegnare agli elementi e trattarli nell'ordine corretto.

Body

Il body è una parte obbligatoria di un messaggio SOAP; è in esso che vengono codificate le informazioni che il messaggio trasporta. Si esamina ora come vengono serializzate le informazioni, facendo riferimento allo schema introdotto da SOAP: http://schemas.xmlsoap.org/soap/encoding. Un esempio: si supponga che la classe Java seguente debba essere trasformata in un messaggio SOAP:

```
package com.una banca;

public class ContoCorrente[

public String conto;

public float saldo;

]
```

La corrispondente parte del messaggio SOAP è costituita dagli elementi di tipo accessor:

```
<t:ContoCorrente xmlns:t='urn:java:com.unabanca'>
```

```
    <conto>7899</conto>
    <saldo>500000000</saldo>
</t:ContoCorrente>
```

Gli elementi accessor possono essere di due tipi: semplici o composti. Gli accessor semplici corrispondono a tipi di dati primitivi, come quelli dell'esempio, così come sono definiti nelle specifiche degli XML Schema Parte 2: Tipi di dati. Gli schemi standardizzano i nomi e le rappresentazioni di stringhe, numeri, date, ecc, ed inoltre permettono di descrivere nuovi tipi di dati. Per questi elementi, il valore è codificato direttamente come una sequenza di caratteri. Gli elementi accessor composti, sono delle strutture di elementi sempici. Ci sono due tecniche per codificarli. Il modo più semplice è di incorporare il valore strutturato direttamente nell'elemento accessor (elementi embedded). Tornando all'esempio, se vi è una classe come:

```
    package com.una banca;
    public class Bonifico [
        public ContoCorrente from;
        public ContoCorrente to;
        public float somma;
    ]
```

il corrispondente messaggio SOAP è:

```
    <t:Bonifico xmlns:t='urn:java:com.unabanca'>
       <from>
           <conto>7899</conto>
           <saldo>50000000</saldo>
       </from>
       <to>
           <conto>212onto>
           <saldo>1000000</saldo>
       </to>
       <somma>5000000</somma>
    </t:Bonifico>
```

Ci sono però alcuni aspetti da considerare quando si parla degli elementi composti. Per esempio, uno dei campi da serializzare potrebbe avere un valore nullo. In questo caso SOAP

ricorre di nuovo alla codifica degli XML Schema, utilizzando l'attributo `null` per indicare che uno dei valori referenziati è nullo. Se un oggetto della classe in esempio ha un valore nullo (in questo caso la voce `from`):

```
<t:Bonifico xmlns:t='urn:java:com.unabanca'
xmlns :xsd='http://www.w3c.org/1999/XMLSchema/istance'>
     <from xsd :null='true' />
     <to>
            <conto>212onto>
            <saldo>1000000</saldo>
     </to>
     <somma>5000000</somma>
</t:Bonifico>
```

Un altro aspetto da considerare è quando si ha ereditarietà fra le classi. SOAP consente di rappresentare oggetti che specializzano altri oggetti utilizzando la convenzione degli XML Schema, i quali ricorrono ad un attributo che specifica il tipo con un namespace. Proseguendo con l'esempio, si ha una classe:

```
     package com.una banca;
     public class ContoCorrenteProtetto [
          public int protezione;
          ]
```

utilizzata in un frammento di codice del tipo:

```
     Bonifico bonifico=new Bonifico();
     bonifico.from= new ContoCorrenteProtetto();
     bonifico.from.conto="ABCD";
     bonifico.from.saldo=3000000;
     bonifico.from.protezione=3;
     bonifico.to= new ContoCorrente ();
     bonifico.to.conto="2345";
     bonifico.to.saldo= 100000;
```

Il meccanismo di serializzazione di SOAP crea quindi un corrispondente messaggio di questo tipo:

```
<t:Bonifico xmlns:t='urn:java:com.unabanca'
xmlns :xsd='http://www.w3c.org/1999/XMLSchema/istance'>
     <from xsd :type='t:ContoCorrenteProtetto'>
          <conto>ABCD</conto>
          <saldo>3000000</saldo>
     </from>
     <to>
          <conto>2345</conto>
          <saldo>100000</saldo>
     </to>
     <somma>5000000</somma>
</t:Bonifico>
```

Questo meccanismo però ha alcuni problemi. Il più grave è che non è in grado di rappresentare le identità degli oggetti. Infatti, se nel nostro esempio vi fosse un frammento di codice del tipo:

```
bonifico.to=bonifico.from;
```

e, dopo essere stati serializzati e deserializzati tramite SOAP, gli oggetti venissero confrontati con l'istruzione:

```
bonifico.to.equals(bonifico.from);
```

l'esito sarebbe negativo in quanto la relazione d'identità andrebbe persa. Per ovviare a questo problema,. SOAP permette di creare anche degli elementi detti multireference accessor. Il concetto che sta alla base di questi elementi è questo: anziché serializzare i singoli dati in loco, si utilizzano dei riferimenti a istanze di oggetti serializzati altrove nel messaggio, ovvero agli elementi ndipendenti. Gli elementi indipendenti si distinguono poiché devono obbligatoriamente avere l'attributo soap:id il cui valore è unico in tutto il messaggio, mentre gli elementi multireference accessor devono contenere l'attributo soap:href il cui valore corrisponde all'identificatore dell'elemento indipendente. Il problema predente, quindi, può essere risolto nel seguente modo, utilizzando un elemento indipendente:

```
<t:Bonifico xmlns:t='urn:java:com.unabanca'>
     <from soap:href='#idl' />
```

```
    <to soap:href='#idl' />
    <somma>500000</somma>
</t:Bonifico>
…..
<t:ContoCorrente soap:id='#idl' xmlns:t='urn :java :com.unabanca'>
    <conto>ABCD</conto>
    <saldo>3000000</saldo>
</t:ContoCorrente>
```

L'utilizzo degli elementi indipendenti è soggetto ad alcune restrizioni imposte dalle regole di serializzazione di SOAP. Il protocollo, infatti, permette l'impiego di elementi indipendenti solo come diretti discendenti degli elementi header o body. In alternativa, definisce l'attributo `soap:package` che può essere applicato a qualsiasi elemento del messaggio, per controllare dove possono essere codificati gli elementi indipendenti. Quando è presente questo attributo, l'elemento XML che codifica l'istanza dell'oggetto deve essere completamente autocontenuto, cioè non può avere riferimenti al di fuori del package.

Un tipo particolare di elementi composti sono gli array; il protocollo indica che devono avere un rank (numero di dimensioni) ed una capacità. Gli elementi di tipo array devono obbligatoriamente contenere l'attributo `SOAP:arrayType`, il cui valore specifica il tipo e la lunghezza dell'array da rappresentare. SOAP permette di rappresentare array di valori primitivi, multidimensionali, di tipi strutturati e anche array che a loro volta ne contengono altri. Inoltre, il protocollo permette di codificare array riempiti parzialmente, rappresentarne una sottoparte utilizzandi l'attributo `soap:offset` oppure ancora trattare array sparsi tramite l'attributo `soap:position` che permette di specificare l'indice assoluto dell'elemento rappresentato. Per la loro codifica si può ricorrere anche ad elementi indipendenti.

Esemplificando, si mostra come vengono serializzari array di tipi strutturati, considerando le seguenti classi:

```
public class Attivita [
    public int codice;
    public String nome ;
]
public class Workflow [
    public Attivita [] processo;
```

```
]
```

e l'oggetto da serializzare, istanziato nel seguente modo:

```
Workflow wf = new Workflow();
Wf.processo = new Attivita[3];
processo[0] = new Attivita (1, "Registra ordine");
processo[1] = new Attivita (2, "Analizza ordine");
processo[2] = new Attivita (3, "Avvia produzione");
```

la versione SOAP risultante è:

```
<t:Workflow xmlns:t='uri di workflow'>
   <processo SOAP=ENV:arrayType='t:Attivita[3]'>
      <Attivita><codice>1</codice><nome>Registra ordine</nome>
        <Attivita><codice>2</codice><nome>Analizza ordine</nome>
        <Attivita><codice>3</codice><nome>Avvia produzione</nome>
   </processo>
</t:Workflow>
```

Gestione degli errori

Nella comunicazione tra un client ed un server, a volte può succedere che qualcosa non funzioni, e si verifichino di conseguenza degli errori, anche a livello in cui opera SOAP. Per esempio, qualcosa nella serializzazionde dei dati può fallire, un header obbligatorio non può essere soddisfatto, una richiesta non può essere adeguatamente servita, oppure l'applicazione invocata può sollevare degli errori. Le specifiche di SOAP permettono di gestire anche queste condizioni d'errore, introducendo un opportuno elemento chiamato `Fault`. Nel caso in cui sia presente, deve trovarsi obbligatoriamnente all'interno del body e non può comparire più di un volta. Lo schema che lo definisce, introduce quattro elementi: *faultcode*, *faultstring*, *faultactor* e *detail*.

- *Faultcode* è un codice identificativo dell'errore pensato per essere computato in maniera automatica (da programma), ed è obbligatorio. E' stato progettato per essere estendibile, definendo codici più precisi, mantenendo comunque la compatibilità, all'indietro. Il meccanismo è molto simile a quello utilizzato nel protocollo http, che impiega cidici numerici di stato. In SOAP però i codici devono avere nomi qualificati, utilizzando stringhe separate dal carattere "." Per separare il codice più generale (a sinistra del punto), da quello più specifico (alla destra). Esistono dei codici di errore

predefiniti: VersionMismatch, MustUnderstand, Client, Server. Un esempio di faultcode è: *Client.Autenticazio*ne.

- *FauksString* è la descrizione, obbligatoria, dell'errore e deve avere una forma comprensibile per un operatore umano.
- *Faultactor* specifica la sorgente dell'errore. Il suo valore è un URI che indicaappunto il responsabile dell'errore.
- *Detail* contiene informazioni di errore specifiche dell'applicazione e relative al contenuto del body. Deve essere presente nel caso in cui il body non sia stato processato correttamente. A differenza dei tre elementi precedenti, tuttidi tipo stringa, può essere un elemento complesso.

Ecco un esempio di errore in SOAP:

```
<SOAP:Fault>
    <faultcode>SOAP:Server.unavailable</faultcode>
    <faultstring>SOAP :Server error</faultstring>
    <detail>
        <e:myfaultdetails xmlns="some-URI">
            <message> The application don't work</message>
            <errorcode>10100</errorcode>
        </e:myfaultdetails>
    </detail>
<SOAP:Fault>
```

Con la specifica SOAP 2.1 viene introdotto un header nuovo, utile nella gestione di un particolare tipo di errore. L'header è Misunderstood e viene usatio insieme a Fault quando un nodo riceve un messaggio SOAP con un header da elaborare obbligatoriamente, quindi con l'attributo mustUnderstand ="true", ma non riesce a capirlo/elaborarlo. All'interno dell'attributo Misunderstood, nell'attributo qname, viene specificato l'header che non è stato capito/elaborato.

Con la nuova specifica vengono inseriti i seguenti codici d'errore, che l'applicazione può utilizzare come valore del sottoelemento `faultcode`:

- *MustUnderstand*: se un elemento della sezione header, contenente l'attributo MustUnderstand attivo, non è stato capito/elaborato dal nodo destinatario.
- *VersionMismatch*: se viene trovato un namespace non valido nell'elemento Envelope

- *Sender*: se il messaggio non è scritto in modo conforme alla specifica, oppure non contiene le informazioni attese nell'ordine stabilito.
- *DTDNot Supported*: se il messaggio SOAP contiene un DTD
- *Receiver*: se il messaggio non può essere correttamente processato, ma le ragioni non sono da ricercare all'interno del messaggio stesso.

SOAP come protocollo per l'RPC

Uno degli obiettivi principali di SOAP è di incapsulare il meccanismo di chiamate a procedure remote. Per eseguire una chiama ta a procedura remota SOAP necessita delle seguenti informazioni:

- L'URI dell'oggetto richiesto;
- Il nome del metodo da invocare;
- La firma del metodo (opzionale);
- I parametri del metodo;
- Intestazioni opzionali.

RPC e il campo <Body> di SOAP

I metodi di chiamata e risposta RPC, sono entrambi posti nel campo Body di SOAP, mediante la seguente rappresentazione:

- l'invocazione al metodo è modellata come una struttura,
    - l'invocazione al metodo è vista come una singola struttura, contenente un ingresso per ogni parametro di input o di input/output. Questa struttura ha lo stesso nome e tipo del metodo,
        - ogni parametro è visto come un ingresso, con un nome e un tipo, corrispondenti rispettivamente al nome e al tipo del parametro. Questi compaiono nello stesso ordine della firma del metodo,
- la risposta del metodo è modellata come una struttura,
    - la risposta del metodo è vista come una singola struttura, contenente un ingresso per il valore di ritorno, e per ogni parametro di input o di input/output. Il primo ingresso è per il valore di ritorno, seguito dai parametri, posizionati nelle stesso ordine della firma del metodo,
        - ogni parametro è visto come un ingresso, con un nome e un tipo, corrispondenti rispettivamente al nome e al tipo del parametro. Il nome dell'ingresso del valore di ritorno non è significativo. Allo stesso modo, il nome della struttura non è significativo. Per convenzione, si usa dare un

nome alla struttura, dopo il nome del metodo, con l'aggiunta della stringa "Response".

- Il fallimento di un metodo, è codificato utilizzando l'elemento SOAP Fault.

Un'applicazione può elaborare delle richieste con dei parametri mancanti, ma può anche ritornare un errore.
Dato che un risultato indica successo, e un errore indica fallimento, è un errore se la risposta del metodo contenere sia il risultato che l'errore.

RPC e il campo <Header> di SOAP

Informazioni addizionali relative alla codifica della richiesta del metodo, ma non facenti parte della formale firma del metodo, possono essere espresse nella codifica RPC. Se questo si verifica, devono essere espresse come sotto-elemento del campo Header di SOAP.

RPC e il campo <Fault> di SOAP

L'elemento Fault nel Body del messaggio SOAP, viene ancora una volta, utilizzato per la gestione degli errori per le RPC. Dalla specifica di SOAP 1.2, i principali codici di errore che il sottoelemento Faultcode può assumere, sono i seguenti:

- *Server*: il server non può gestire il messaggio, per esempio se non ha memoria sufficiente.
- *DataEncodingUnknown*: il server non comprende come sono stati codificati i dati all'interno del Body e/o dell'Header di richiesta (il valore dell'attributo encodingStyle).
- *ProcedureNotPresent*: il server non riesce a trovare la procedura specificata.
- *BadArguments*: il server non riesce ad analizzare i parametri oppure non c'è corrispondenza fra ciò che il server si aspetta e ciò che gli ha inviato il client.

Sviluppi futuri

Don Box di Microsoft ha recentemente annunciato che non intende sostituire la versione 1.2 delle specifiche di SOAP con una versione 1.3 che introduca cambiamenti significativi.
Box ritiene che il World Wide Web Consortium (W3C) impiegherà ancora da 6 a 12 mesi per ratificare SOAP 1.2 e quindi pensa che sia opportuno che la versione 1.2 sia l'ultima prima del decollo del mercato.
"Se avessimo bisogno di un SOAP 1.3 vorrebbe dire che il modello estensibile di SOAP 1.2 non è buono e probabilmente una nuova versione non risolverebbe il problema", ha detto Box.
Il guru di Microsoft ritiene che il modo migliore per potenziare SOAP sia quello di aggiungere protocolli basati sulla Global XML Architecture (GXA), come WS-Security, WS-Coordination e

WS-Transaction, agli header dei messaggi SOAP.

Capitolo 6 – WSDL: specifica 1.2 del W3C

Introduzione

Uno degli scenari in cui SOAP sta trovando maggiori campi applicativi è nello sviluppo dei Web services. Nel corso degli ultimi anni le finalità e le tecnologie che ruotano intorno al web sono radicalmente cambiate, tanto che si potrebbe dire di avere assistito al passaggio tra varie “ere” nel modo di utilizzare e pensare Internet. Dalla pubblicazione di documenti statici si è passati alla creazione di siti che visualizzano informazioni altamente dinamiche derivanti da vere e proprie elaborazioni, comportando una maggiore interazione e personalizzazione per gli utenti che usufruiscono delle informazioni e un cambiamento radicale per lo sviluppatore che ora può intendere il Web come piattaforma di sviluppo.

I web services si pongono come l’ulteriore evoluzione del web che, con essi, offre oltre ad informazioni fruibili direttamente da un browser, anche servizi utilizzabili dalle proprie applicazioni. SOAP sembra essere lo strumento ideale per creare applicazioni distribuite su Internet che si avvalgono di servizi remoti, poiché, come visto, costituisce uno standard (anche se ancora in corso di definizione) per far comunicare piattaforme ed applicazioni eterogenee, con il vantaggio di permettere di imbustare messaggi o chiamate a procedure remote e di definire dei tipi specifici (tramite gli XML Schema) per le applicazioni.

Il fenomeno dei web services è molto recente, ma sta rapidamente crescendo, e diffondendo grazie anche all’ausilio di tecnologie che lo supportano, come appunto SOAP. SOAP però non è sufficiente per creare un servizio su Internet. Occorre anche un modo per far conoscere agli utenti del servizio cosa offre e come va usato, e questo è realizzato tramite il *Web Service Description Language* (WSDL), un’altra applicazione di XML. Altra tecnologia nascente che servirà per integrare sempre più applicazioni eterogenee nell’ambito del web è *L’Universal Description Discovery and Integration* (UDDI), che costituisce una specifica di un’applicazione XML che permette di creare dei registri di servizi web al fine di permettere la loro individuazione su Internet.

La specifica 1.2

WSDL è stato annunciato congiuntamente da Microsoft e Ibm nel settembre 2000, esprimendo quello che un Web service può fare, dove vive e come può essere richiamato.

Il 24 Gennaio 2003 il W3C ha rilasciato il *Web Services Description Language (WSDL) 1.2* e *WSDL 1.2 Bindings* come W3C Public Working Drafts.

WSDL 1.2 e' un linguaggio basato su XML che descrive un Web Service - i dati scambiati, il protocollo da utilizzare, e la locazione nel Web. *WSDL 1.2 Bindings* descrive come usare WSDL1.2 con SOAP 1.2, HTTP, e MIME. WSDL 1.2 fornisce una migliore interoperabilita', e definizione dei componenti, risultato della libera partecipazione nel definizione dei requisiti a partire da quelli di WSDL 1.1, e di quelli del Working Group per una specifica aperta. WSDL 1.2 fornisce diversi punti di miglioramento rispetto a WSDL 1.1

- include chiarimenti sul linguaggio, che rendono più semplice per gli sviluppatori la comprensione e l'utilizzo.
- fornisce supporto per le W3C Recommendations, inclusa XML Schemas and XML Information Set.
- adotta un approccio di framework concettuale per definire la descrizione delle componenti, il che le rende più semplici e flessibili.
- rimuove caratteristiche non necessarie e non interoperabili per WSDL 1.1.
- fornisce una migliore definizione per i binding con HTTP 1.1 e fornirà presto un binding per SOAP 1.2 , che permette la descrizione di servizi utilizzando l'ultima versione di SOAP.

Quindi, Wsdl 1.2 è più semplice e più flessibile di quanto non fosse la versione precedente e dovrebbe, così, facilitare il lavoro degli sviluppatori di servizi Web. L'ultima iterazione dello standard include una miglior definizione dei componenti, dei linguaggi, un framework concettuale che definisce le descrizioni dei componenti e il supporto per gli standard Xml Schemas e Xml Information Set. Inoltre, la versione 1.2 rimuove le caratteristiche non interoperabili di Wsdl 1.1 e rende più efficace il lavoro con HTTP e SOAP.

Caratteristiche

Dal momento che vogliamo che il nostro scambio di dati viva di vita propria, ovverosia che chi si occupa di sviluppare il client sia messo in condizione di sapere esattamente come interfacciarsi col Service, è stata elaborata una grammatica XML apposita, che si chiama Web Service Description Language (WSDL) definita da uno schema XSD.
WSDL è una specifica che interessa sicuramente chi intende sviluppare Web Services. Per ciascun servizio è bene definire un WSDL in modo che il client non acceda direttamente al servizio web tramite il corrispondente file WSDL.
Per alcuni aspetti l’utilizzo di WSDL può essere ridondante, come ad esempio le informazioni

aggiuntive necessarie a definire il binding con SOAP (potrebbero essere implicite), ma dobbiamo ricordare che lo scopo del WSDL è quello di definire una grammatica di definizione di un servizio di network generico e non strettamente legato agli altri protocolli propri dei Web Services.
Quindi un parser di documenti WSDL (utile ad esempio a validare le richieste e le risposte verso e da un servizio web) dovrà interpretare tutte queste informazioni ed estrarne quelle utili.

WSDL ha quindi come obiettivo la descrizione dei servizi Web esposti da un server. In particolare un documento WSDL è un file XML che presenta al suo interno diverse sezioni:

- la definizione del servizio *(service)*: informazioni su un servizio esposto
- i collegamenti *(binding)*: descrive il protocollo supportato, le operazioni consentite ed i relativi input e output
- le operazioni *(portType)*: descrive l'insieme delle operazioni supportate e ognuna delle quali costituita dai messaggi di richiesta e di risposta
- i messaggi *(message)*: contiene i parametri di richiesta e di risposta del servizio
- i tipi (t*ypes)*: contiene le definizioni dei tipi di dati utilizzati in ingresso ed in uscita con la descrizione della loro struttura

WSDL può quindi essere inteso come:

- un linguaggio che permette di esprimere informazioni di contratto fra client e server (l'interfaccia del servizio messo a disposizione)
- un linguaggio per specificare il protocollo di comunicazione fra richiedente e fornitore, vale a dire le regole attraverso cui le applicazioni possono ricevere e spedire correttamente i messaggi SOAP attraverso la rete

WSDL descrive un servizio come una collezione di operazioni; le operazioni e i messaggi sono descritti in termini astratti e poi legati ad un protocollo di rete concreto e ad un preciso stile di codifica dei valori scambiati nei messaggi stessi.
Più precisamente, WSDL è in grado di specificare:

- il nome dell'operazione da invocare
- il numero dei parametri di ingresso e di uscita
- il tipo dei parametri
- il modo in cui i parametri (ossia le istanze di un certo tipo di dato) verranno serializzati per essere trasportati sulla rete.
- L'URL per raggiungere il servizio che implementa la descrizione WSDL stessa

UDDI

A fianco di Soap, uno standard che sta assumendo sempre più rilevanza è Uddi (Universal description, discovery and integration), che fu annunciato nel settembre del 2000 da Ariba, Ibm e Microsoft. L'Uddi è stato la base dei progetti di e-commerce e dei marketplace online, poco prima che scoppiasse la bolla della new economy. Basato su Soap e Wsdl, Uddi permette ai provider di pubblicizzare i Web service che sono preparati a offrire. Equivalente dei name server nel mondo Corba e Java, Uddi opera su tre livelli separati: white page (informazioni sugli indirizzi), yellow page (categorie business), green page (dettagli tecnici su come deve essere condotta la transazione).

UDDI (Universal Discovery Description and Integration) rappresenta le pagine gialle dei servizi Web. Come con le pagine gialle tradizionali, è possibile cercare una societàche offre i servizi richiesti, leggere informazioni sui servizi disponibili e contattare qualcuno per i dettagli. Naturalemente, si può offrire un servizio Web senza registrarlo in UDDI, ma se lìobiettivo è raggiungere un mercato importante, con UDDI si è certi che i clienti non avranno difficoltà a scoprire il servizio offerto.

Una voce di un elenco UDDI è un file XML che descrive un'attività e i relativi servizi. Tale voce si suddivide in tre parti. Le "pagine bianche" descrivono la società che offre il servizio: nome, indirizzo, contatti, ecc. Le "!pagine gialle" comprendono categorie aziendali basate su tassonomie standard quali il North American Industrial Classification . Le "pagine verdi" descrivono l'interfaccia del servizio con particolari sufficienti a consentire ai clienti potenziali di scrivere un'applicazione per utilizzare il servizio Web. I servizi vengono definiti attraverso un documento UDDI denominato Type Model o tModel. In molti casi, il tModel contiene un fiel WSDL che delinea un'interfaccia SOAP per un servizio Web XML, ma il tModel è abbastanza flessibile per descrivere quasi ogni tipo di servizio.

L'elenco UDDI inoltre comprende varie modalità di ricerca dei servizi necessari per creare applicazioni. Ad esempio, è possibile eseguire ricerche per fornitori di serviziz in una località geografica specificata o per attività commerciali di tipo specifico. Quindi l'elenco UDDI fornirà informazioni, contatti, collegamenti e dati tecnici per consentire all'utente di valutare quali servizi soddisfino le proprie esigenze.

UDDI consente di individuare conwesattezza le aziende in grado di offrire servizi Web ottimali.

Capitolo 7 – SOAP e la sicurezza

Sicurezza nei servizi Web

L'avvento dei servizi Web ha causato nuovi problemi che non esistevano in ambienti chiusi. Le nuove caratteristiche di apertura e scambio dati, oltre alle problematiche d'interoperabilità, comportano nuove sfide per la sicurezza dei dati e dell'identità.

- Chiunque può accedere alle applicazioni aziendali e alle relative interfacce: esse sono disponibili sulla porta 80 che generalmente nei firewall è sempre aperta ed è il punto di transito di tutto il traffico HTTP. Non per questo si deve, però, credere che tutto quello che attraversa questa porta sia protetto. Le applicazioni con interfacce di frontend ai dati critici saranno sempre più esposte tramite HTTP e facilmente accessibili dal mondo esterno. Un'ipotesi eccessiva è la pubblicazione in una directory pubblica in modo che chiunque la possa scoprire.
- La tecnica d'incapsulamento dei dati negli envelope SOAP permette di capire la struttura e il significato dei dati inviati in rete.
- Le ímplementazíoni del mittente e del destinatario non devono basarsi obbligatoriamente sulle stesse piattaforme software: non è detto, quindi, che le librerie di sicurezza appartengano allo stesso produttore di software. Per questo motivo è necessario disporre di soluzioni di sicurezza standardizzate e indipendenti dalle singole piattaforme.
- XML è assai prolisso e la crittografia è un processo molto lungo: l'inclusione di dati codificati in XML è già sufficiente ad aumentare in modo considerevole la dimensione dei dati da crittografare.
- La filosofia di programmazione legata ai servizi Web consente la generazione spontanea di comunità di fornítori o clienti, resa possibile dalla possibilità di effettuare ricerche dinamiche. Quest'ottica comporta interazioní complesse, dato che un messaggio SOAP deve toccare diversi nodi intermedi; potrebbero non esserci accordi commerciali stipulati in precedenza o i nodi potrebbero non essere basati su un'infrastruttura comune. Com'è possibile gestire le chiavi che consentono la críttografia in un ambiente di questo tipo?

Per rispondere a queste esigenze è in corso di sviluppo una nuova classe di tecniche di sicurezza. Molti problemi devono ancora venir identificati o le loro soluzioni potrebbero essere ancora in una fase embrionale; tuttavia, le tecnologie di sicurezza esistenti non saranno abbandonate così presto: infatti, le nuove tecniche servono a migliorare gli apparati di protezione esistenti, come le infrastrutture a chiave pubblica PKI (Public Key Infrastructures), HTTPS (Secure HTTP) e SSI, (Secure Sockets Layer).

Inclusione della sicurezza in XML

La sicurezza nell'e-commerce basato su HTTP, del quale si è discusso fino a questo momento, si basa su SSI, che assegna la responsabilità ultima della sicurezza stessa al protocollo di trasporto. Benché quest'approccio finora è risultato il più adeguato, è necessario disporre di una nuova serie di funzionalità che consentano l'impiego di certificati, firme digitali e autenticazione dei documenti XML: infatti, SOAP e i servizi Web dovrebbero essere indipendenti dal protocollo, quindi non possono affidarsi ai protocolli di trasporto per le loro esigenze di sicurezza. L'incorporazione delle funzionalità di sicurezza all'interno degli stessi documenti XML offre, inoltre, numerosi benefici per i servizi Web che finalmente sono in grado di entrare nel merito del contenuto informativo dei documenti: possono, quindi, esservi applicate le restrizioni.

Infrastruttura a chiave pubblica PKI

La crittografia a chiave pubblica si basa su algoritmi matematici che generano una coppia di chiavi per la crittografia e la decodifica: se per crittografare i dati si utilizza la chiave di una coppia, soltanto l'altra sarà in grado di effettuare la decodifica; una delle due è pubblica, mentre l'altra è mantenuta privata (segreta). Nelle procedure d'invio e ricezione che utilizzano meccanismi di crittografia e decifrazione, il mittente si serve della chiave pubblica offerta dal ricevente per crittografare i dati: solo quest'ultimo, che è l'unico a possedere la chiave privata appropriata, sarà in grado di decifrare i dati trasmessi.

Potete crittografare un documento anche con la vostra chiave privata; in questo caso, chiunque abbia accesso alla vostra chiave anche pubblica potrà decriptarlo. Sebbene questa tecnica non sembri particolarmente utile, dal momento che chiunque può accedere a una chiave pubblica, essa garantisce in ogni caso che un documento firmato elettronícamente non si possa falsificare.

Il vero problema della crittografia a chiave pubblica è la generazione, la distribuzione e il controllo delle chiavi. In che modo qualcuno può entrare in possesso della vostra chiave, se vuole entrare in affari con voi? Come potrà essere certo che la chiave ricevuta sia effettivamente la vostra e non un'imitazione? L'infrastruttura PKI viene utilizzata proprio per risolvere questi problemi di cui si parlerà in dettaglio nel corso del testo.

"Non ripudio"

Il concetto di "non ripudio" nell'ambito dei servizi Web indica che il ricevente può verificare che un documento XML, sia esso un ordine d'acquisto o una richiesta RPC, provenga effettivamente dal mittente riportato al suo interno e non sia stato modificato durante il trasporto. Esso è l'equivalente

elettronico dell'affermazione: "Non potete dire che non volevate la casa dipinta di un certo colore. Questo documento è l'ordine di lavoro con la vostra firma!". In maniera più tecnica, il concetto di "non ripudio" rimanda a un'autenticazione valida che è attiva, in grado di verificare le identità di tutti i partecipanti alla transazione e la presenza di elementi validi per ricostruire il percorso fatto dal messaggio fino a un evento specifico in modo da confermare i punti seguenti:

- il mittente è corretto;
- è colui che effettivamente ha fatto la richiesta;
- ha richiesto esattamente tutto quello che è riportato nel modulo di richiesta.

Di norma, le tecniche di “non ripudio" prevedono il ricorso alla firma digitale.

Firme digitali XML

La firma digitale, da non confondersi con il certificato digitale, è l'equivalente elettronico di una firma autografa e viene utilizzata nelle applicazioni distribuite per autenticare l'identità del mittente di un messaggio o di un documento; assicura anche che il messaggio o il documento non siano stati modificati durante il trasporto.

Le specifiche XML per la firma digitale definiscono un elemento XML opzionale, che semplifica l'inclusione della firma digitale in un documento XML: esso fornisce a qualsiasi servizio Web la capacità di garantire l'integrità dei dati, l'autenticazione e il "non ripudio" nei confronti di qualsiasi altro servizio Web.

Oltre a definire la sintassi, le specifiche prevedono raccomandazioni riguardanti i tipi di dati che richiedono una firma digitale: quelle che più di tutte sono da prendere in considerazione sono la firma degli elementi visuali, come i fogli di stile CSS (Cascading Style Sheets) e i plug-ín dei browser che, pur essendo esterni ai dati XML effettivi, possono essere usati insieme ai dati XML. Di regola le specifiche raccomandano che, nei casi della rappresentazione XML dei dati visuali, è opportuno conservare nel tempo la validità delle informazioni firmate: quindi, dovrebbero essere firmati sia i dati XML sia le voci da utilizzare per la rappresentazione visuale. Le specifiche suggeriscono che questa regola sia applicata anche con presentazioni non visuali come nel caso degli elementi audio.

Consideriamo il seguente file prova.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<PurchaseOrder xmlns="urn:oreilly-jaws-samples">
        <shipTo country="US">
                <name>Joe Smith</name>
                <street>14 Oak Park</street>
                <city>Bedford</city>
```

```
            <state>MA</state>
            <zip>01730</zip>
        </shipTo>
        <items>
            <item partNum="872-AA">
                <productName>Candy Canes</productName>
                <quantity>444</quantity>
                <price>1.68</price>
                <comment>I want candy!</comment>
            </item>
        </items>
</PurchaseOrder>
```

Quella che segue è una versione con firma digitale del file precedente, prova.xml: le informazioni relative all'ordine d'acquisto restano invariate, ma le dimensioni del documento sono aumentate, in gran parte a causa del nuovo elemento <*Signature*>:

```
<PurchaseOrder xmlns="urn:oreilly-jaws-samples">
        <shipTo country="US">
            <name>Joe Smith</name>
            <street>14 Oak Park>/street>
            <city>Bedford</city>
            <state>MA</state>
            <zip>01730</zip>
        </shipTo>
        <items>
            <item partNum="872-AA">
            <proructName>Candy Canes</proructName>
            <quantità>444</quantity>
            <price>1.68</price>
            <comment>I want candy!</comment>
            </item>
        </items>
        <Signature ld="EnvelopedSig" xmins='http://www.w3.org/2000/09/xmldsig#">
            <SigneInfo Id="EnveIopeSig.SigInfo">
            <CanonicaIizationMethod Algorithm=
            "http://www.w3.org/TR/2001/REC-xml-cl4n-20010315"/>
            <SignatureMethod Algorithm=
            "http://www.w3.org/2000/09/xmidsig#rsa-sha1"/>
            <Reference Id="EnvelopeSig.Ref" URI="">
            <Transforms>
            <Transform Algorithm=
            "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </Transforms>
            <DigestMethod Algorithm=
            "http://www.w3.org/2000/09/xmidsig#sha1"/>
            <DigestValue>
            yHlsORnxE3nAObbjMKVo1qEbToQ=
            </DigestValue>
            </Reference>
            </SigneInfo>
            <SignatureValue Id="EnvelopedSig.SigVaIue">
        GqWAmNzBCXrognOBIC2VJYA8CS7gugxH/XVWFaO8eY9HqVnrfU6Eh5lg6wlcvj4
        RrpxnNklBnOuvv£Kq110y4e76Tduq/N8kVdOSkYf20ZAC+jllqUPFQe8CNA0Cf
        UrHZdiS4TDDVv4sfOVlc6UBj7zTAeMAdgpOg/2Cxc=
            </SignatureVaJue>
            <KeyInfo ld="EnvelopedSig.Keyinfo">
            <KeyValue>
            <RSAKeyValue>
            <Modulus>
        ANPU2eRs9C5ffic61PAOtQ5fM+g3R1Yr6mJVd5zFrRRrJzB/awFLXb73kSlWqHao+3nxuF38
        rRkqiQOHmqgsoKgWChXmLuQ5RqKJilqxOG+WoTvdYY/KB2qgmTDjOX8+0GlkSCZPRTkGiKiD
        7rw4Vvml7nKlqWg/NhCLWCQFWZ
            </Modulus>
            <Exponent>AQAB</Exponent>
            </RSAKeyValue>
        </KeyValue>
```

```
        </KeyInfo>
</Signature>
</PurchaseOrder>
```

Il primo passaggio nella creazione di una firma digitale consiste nel garantire che i dati firmati non possano essere alterati: questa specifica viene eseguita applicando un algoritmo matematico, chiamato *secure bash*, a una parte dei dati del messaggio: il risultato viene definito "riassunto" (*dígest*) del messaggio. La fase successiva consiste nel prendere quest'ultimo e tutte le informazioni aggiuntíve da firmare (quelle contenute nell'elemento <Signeinfo>, farne un nuovo riassunto, crittografarle e inserirle nel messaggio XML stesso come firma digitale. In prova. xml l'algoritmo selezionato e il riassunto iniziale sono inseriti negli elementi <DigestMethod> e <DigestValue>. La firma digitale, in tal caso il riassunto críttografato, viene inserita nell'elemento <SignatureValue>, mentre la chiave di decodífica è posta nell'elemento <KeyInfo>. Il ricevente, per determinare se la firma è valida, dovrà decriptare il riassunto percorrendo a ritroso l'intero processo eseguito dal mittente per crearlo: se il risultato corrisponde all'originale, il contenuto firmato probabilmente non ha subito modifiche durante il trasporto.

L'elemento <Reference>

L'elemento <Reference> indica le informazioni utilizzate per generare il riassunto del messaggio, che rendono conto di tutte le trasformazioni o normalizzazioni dei dati eseguiti durante il trasporto, inclusa la conversione in formato canonico del messaggio, la cosiddetta canonicalization. L'associazione di una firma digitale a un documento XML può avvenire nei tre modi elencati di seguito.

Enveloped

La firma è un elemento figlio dei dati da crittografare.

Enveloping

La firma contiene i dati crittografati.

Detached

La firma è una copia fedele dell'elemento crittografato alla quale si può fare riferimento mediante un collegamento locale o che può trovarsi sulla rete.

Queste informazioni devono essere trasportate nella firma usando il tag <Transforms>. Nell'esempio si è deciso di ricorrere al metodo Enveloped:

```
<Transforms>
<Transform Algorithm=
  "http://www.w3.org/2000/09/xmidsig#enveloped-signature"/>
</Transforms>
```

Altri esempi algoritmi di trasformazione (transform) sono la codifica in base 64, il filtro XPATH, la trasformazione XSLT e la validazíone dello schema.

Nell'esempio di cui si sta discutendo, l'algoritmo selezionato e il riassunto sono identificatí dai tag seguenti:

```
<DigestMethod Algorithm=
        "http://www.w3.org/2000/09/xmidsig#shal V>
<DigestValue>
yHlsORnxE3nAObbjMKVo1qEbToQ=
</DigestValue>
```

E' il caso, a questo punto, di approfondire l'analisi dell'elemento <Signature>. L'elemento <SigneInfo> è obbligatorio, perché specifica quali dati sono effettivamente crittografati e i relativi algoritmi. Il tag <SigneInfo> possiede tre sottoelementi: <CanonicalizationMethod>, <SignatureMethod> e <Reference>.

Conversione in formato canonico

Un algoritmo di tipo *secure hash* non tollera il minimo cambiamento in un documento: qualsiasi modifica, anche solo l'introduzione di uno spazio, può produrre un hash di codifica completamente diverso. Una simile intolleranza assoluta verso le modifiche è una caratteristica peculiare della natura dell'algoritmo: è praticamente impossibile modificare il documento originale e produrre il medesimo hash o determinare a priori gli effetti che la modifica di un documento avrà su di esso. Questa caratteristica, tuttavia, rappresenta anche un problema: i documenti XML sono spesso analizzati durante il percorso di trasmissione dal mittente al destinatario e i parser possono produrre modifiche apparentemente ínsignificanti, come l'eliminazione degli spazi vuoti. La "dichiarazione di autentícità" sottopone il documento a una formattazione standard prima di elaborarne il riassunto, per assicurare che quello calcolato dal mittente e dal destinatario sia il medesimo a prescindere dalle elaborazioni che interverranno lungo il percorso.

Questo cosiddetto "formato canonico" (*canonical format*) è stato standardizzato dal Consorzio W3C, nella specifica contrassegnata dal numero xml-c14n. Nell'elenco seguente, in sintesi, sono elencate le operazioni garantite da una conversione compatibile con xml--c14n:

- il documento è codificato in UTF-8;
- prima di essere sottoposto a parsíng, i marcatori a fine riga del documento sono

normalizzati come #xA (A è esadecímale, 10 decimale o newline ASCII), in fase d'input;

- i valori degli attributi sono normalizzati come se avessero subito un processore di valídazione;
- i caratteri e i riferimenti alle entità sottoposte a parsing vengono sostituiti;
- le sezioni CDATA sono sostituite dal loro contenuto in caratteri;
- la dichiarazione XML e il DTD (Document Type Declaration) vengono rimossí;
- gli elementi vuoti sono convertiti in coppie di tag d'inizio e di fine;
- Gli spazi presenti all'esterno di un elemento del documento e contenuti tra una coppia di taà d'inizio e di fine sono normalizzati;
- tutti gli spazi vuoti nel contenuto in caratteri restano cosil come sono a eccezione di quelli rimossi in fase di normalizzazione dei marcatori di a capo (1inefeed);
- i delimitatori dei valori degli attributi sono le virgolette doppie;
- i caratteri speciali contenuti nei valori degli attributi e nel contenuto testuale vengono sostituiti con riferimenti ai caratteri;
- le dichiarazioni di namespace superflue sono rimosse in ogni elemento;
- a ogni elemento sono aggiunti gli attributi predefiniti;
- le dichiarazioni dei namespace e gli attributi d'ogni elemento sono in ordine “lessicografico".

<SignatureMethod>

La seconda fase della creazione del riassunto consiste nello specificare e controllare a metodo effettivamente utilizzato per creare la firma, contraddistinto dall'elemento <SignatureMethod>. Quando si genera la versione "canonica" del codice XML, i dati che compongono l'elemento <SignedInfo> devono essere convertiti nel valore di firma effettivo e posizionati nell'elemento <SignatureValue>. L'elemento <SignatureMethod> specifica l'algoritmo che sarà impiegato nell'operazione in questione.

L'algoritmo utilizzato per creare la firma e anche quest'ultima vengono specificati nei tag <SignatureMethod> e <SignatureValue>:

```
<SignatureMethod Algorithm=

        "http://www.w3.org/2000/09/xmidsig#rsa-sha1"/>
<Reference Id="EnveJopedSig.Ref" URI="">
  . . . . . .
<SignatureValue Id="EnveIopedSig.SigValue">
GqWAmNzBCXrognOBIC2VJYA8CS7gugxH/ XVWFaO8eYgHqVnrfU6Eh51g6w]cvj4RrpxnNk]BnOuvv
WKq110y4e76Tduvq/ N8kVdOSkYf20ZAC+jl 1qUPFQe8CNAOCfUrHZdiS4TDDVv4sfOV1 c6UBj7zT
71eCOAdgpOg/2Cxc=
</SignatureVaIue>
```

Non appena il messaggio giungerà a destinazione, la firma sarà decodificata servendosi della chiave pubblica del mittente e il riassunto sarà verificato controllando la firma del mìttente. Nel listato seguente la chiave di decifrazione è contenuta nell'elemento <KeyInfo>:

```
<KeyInfo Id="EnvelopedSig.KeyInfo">
        <KeyValue>
        <RSAKeyValue>
        <Modulus>
Alv13Y132e13s9C5FRc61 PAOtQ5fM+g3R1 Yr6mJVd5zFrRRrJzB/awFLXb73kS]WqHao+3nxUF38r
Rke;iQ011mqpoKgWMmU0513qUil qxOG+WoTvdYY/ KB2qgmTDjOX8+0GikSCZPRTkGiKjD7rw4
Vvml7nKlqWg/NhCLWCQFWZ
        </Modulus>
        <Exponent>AQAB</Exponent>
       </RSAKeyValue>
    </KeyValue>
</KeyInfo>
```

Si noti che la firma XML non garantisce la veridicità delle informazioni sulla chiave. E’ infatti, compito dell'applicazione determinare quanto essa sia affidabile. Se non esístesse un modo per verificare che la chiave di decifrazíone fornita appartiene al mittente, il processo di críttatura sarebbe certamente di poca utilità: chiunque potrebbe intercettare il messaggio, modificarne il contenuto, rigenerare una coppia di chiavi pubbliche e private e firmare nuovamente il documento affermando che la chiave pubblica appartiene al mittente. Proprio per evitare tutto questo, sono stati introdotti i certificati digitali che contengono le informazioni relative alla relazione tra l’identità del possessore della chiave pubblica e quest'ultima. Solo nel caso in cui venga omesso l'elemento <KeyInfo>, il ricevente dovrà identificare la chiave che sarà utilizzata sulla base del contesto applicativo. Questo problema è oggetto delle specifiche XKMS che vedremo in seguito. Attraverso XKMS o un'altra infrastruttura PKI, il ricevente del messaggio potrà ottenere un certificato digitale, estrarne la chiave pubblica e verificare la sua effettiva appartenenza al mittente.

Crittografia XML

La fase successiva all'introduzione di una firma digitale nel codice XML consiste nel criptare il documento in tutto o in parte. La crittografia XML estende la potenza del sistema di firma digitale XML, consentendo di sottoporre a crittografia un messaggio contenente una firma digitale. Le specifiche indicano una metodología standard per operare con qualsiasi forma di contenuto digitale e consentono la crittografia di un intero messaggio di tipo XML, di una sua parte o di un messaggio XML contenente sezioni già crittografate in precedenza. Di seguito viene mostrato il file prova.xml con il contenuto del tag <Items> crittografato:

```
<PurchaseOrder xmlns="urn:oreilly-jaws-samples">
        <shipTo country="US">
                <nameJoe Smith</name>
                <street>14 Oak Park</street>
                <city>Bedford</city>
                <state>MA</state>
                <zip>0173</zip>
        </shipTo>
      <items>
```

```
        <EncryptedData Id="ED" Nonce="16"
         Type=http://Www.w3.org/2001/04/Xmlenc#Content
         xmins="http://Www.w3.org/2001/04/xmlenc#"
         xmlns:ds="http://www.w3.org/2000/09/xmidsig#"
         <EncryptionMethod Algorithm
          ="http://www.w3.org/2001/04/xmlenc#aesl28-cbc"/>
          <ds:KeyInfo>
            <ds:KeyName>jaws</ds:KeyName>
          </ds:KeyInfo>
          <CipherData>
           <CipherValue>
dRDdYjYs11jW5eyOlucPkWsBB3NmKOAFNxvFjfeUKxP75cx7KPOPB3BjXPgl4kJv74i7FOOXZ5 Whq
01SswikdN/ piVeqRZWqOVjFA8izR6wqOb7UCpH+weoGtOUFOEkIDGbemm23eu8120b5eYVL8n/DtO8
10hYeMsSMGUZiUNj/tfBCAjvqG2jisIQM6n4jJ3QNaR4+B2RisOD6Ln+x2UtNu2J7wlYmlUe7mSg
ZiJ5eHym8EpkE4vjmr2oCWwTUuglxcayZtbEpOFVFs6A==
           </CipherValue>
          </CipherData>
         </EncryptedData>
       </items>
      <Signature ld="EnvelopedSig" xmlns="http://www.w3.org/2000/09/xmldsig#">
      . . . . . .
      </Signature>
</PurchaseOrder>
```

La parte crittografata del documento è caratterizzata da due nuovi tag: <EneryptedData> e <CipherData>. Il primo elemento definisce lo schema di crittografia da applicare: in questo caso l'operazione viene eseguita ricorrendo a schemi disponibili sul sito Web del Consorzio W3C; l'elemento <CipherData> serve a contenere la serializzazione crittografata dell'elemento <Items>. In questo esempio, il risultato è inserito nell'elemento <CipherValue> o, in alternativa, l'elemento <CipherReference> consente di utilizzare un URI di riferimento alla posizione nella quale risiede il contenuto cifrato. I tag <EncryptionMethod> e <KeyInfo> sono opzionali.

La capacità di crittografare dati in base alle proprie necessità è una caratteristica realmente potente che consente ai servizi Web di gestire a piacere la loro stessa sicurezza: questa funzionalità può oltrepassare tutte le limitazioni che s'íncontrano quando si vuole applicare un meccanismo esterno di crittografia, soprattutto il fatto che la sua applicazione si estenda obbligatoriamente a tutto il documento.

Toolkit Java

Per la creazione degli esempi di questo capitolo, sono stati esaminati due toolkit XML per la sicurezza: *XML Security Suite* di IBM e *XML Phaos*. Usano i parser Xerces e Xalan per effettuare l'analisi dei dati XML e API proprietarie per combinare le firme con i dati cifrati: sono corredati da un'ottima serie di programmi di esempio e di utilità per generare il certificato contenente una coppia di chiavi pubbliche e private. Sono dotati, inoltre, di esempi sulla creazione di documenti firmatí di tipo enveloped, enveloping o detached e sulla crittografia e la decodifica di porzioni di documento a partire da un tag dell' elemento. Il toolkit di IBM impiega proprie estensioni personalizzate per eseguire il parsing: è possibile, quindi, utilizzare un'espressione Xpath, come /PurChaseOrder/ShipTo, per identificare l'elemento da críttografare. Per accedervi, l'esempio di

Phaos, al contrario, ricorre semplicemente a un'API di tipo doc.getE1ementsByTagName (tagName), come mostrato nel listato seguente:

```
//Copyright Phaos Technologies
public class XEncryptTest
{
    public static void main (String[ ] args) throws Exception
    {
        ... //Utilizzo, argomenti da riga di comando...

     //Prende il file XML e recupera l'elemento XML da crittografare
     File xmlFile = new File(inputFileName);
     DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
     dbf.setNamespaceAware(true);
     DocumentBuilder db = dbf.newDocumentBuilder();
     Document doc = db.parse(xmlFile);
     Element inputElement = null;
     NodeList = doc.getElementsßyTagName(tagName);
     if(list.getLength()!= 0)
        inputElement = (Element) list.item(0);
     else
     {

       System.err.println("XML element with tagName"
          + tagName + " unidentified.");
       System.exit(1);
     }

      //Crea una nuova istanza XEEncryptedData con il documento
     //che contiene il file xm] d'input, l'URI dei tipo di dati
    //e l'identificativo "ED" per questo elemento EneryptedData.
    XEEncryptedData encData
         = XEEncrypted Data.newInstance(doc, "ED", dataType);

    ... //Determina l'algoritmo di crittografia

    //Imposta l'elemento figlio EncryptionMethod
   XEEncryptionMethod encMethod = encData.createEncryptionMethod(algURI);
   encData.setEncryptionMethod(eneMethod);

   //Imposta la chiave simmetrica che deve essere usata per la crittografia
   SymmetricKey key = null;
   File keyFile = new File(keyFileName);

  ... //Gestione dei file

  //Imposta l'elemento figlio ds:KeyInfo con il keyName
  XSKeyInfo keylnfo = encData.createKeyInfo();
  keylnfo.addKeyinfoData(encData.createKeyName(keyName));
  encData.setKeyInfo(keyInfo);

  //Imposta un valore occasionale da inserire prima dei testo puro
  byte[ ] nonce = new byte[16];
  encData.setNonce(RandomBitsSource.getDefault().randomBytes(nonce));

  //Codifica l'elemento XML e lo sostituisce con l'elemento
  //EncryptedData appena generato
  System.out.print("Encrypting the xml data... ");
  XEEncryptedData newEncData
        = XEEncryptedData.encryptAndReplace(inputElement, key, encData);
  System.out.println("done");



  //Indirizza in un file il documento XML contenente il nuovo elemento
    EncryptedData
. . . . . .
 }
```

```
}
```

La parte importante di questo codice è la chiamata al metodo encryptAndReplace() che esegue esattamente quello che indica il suo nome: prende l'elemento che gli è stato passato, recuperato mediante la chiamata a getElementsByTagName(), ne esegue la crittografia utilizzando la chiave fornita e sostituisce all'elemento originale l'elemento crittografato con i tag opportuni.

Il toolkít Phaos si è rivelato assai più semplice da impostare e utilizzare di quello IBM. Tutti i file di estensione jar necessari fanno parte di un unico download; il toolkit IBM ha richiesto il download di una versione beta di Xerces, di una beta di Xalan e del toolkit JCE (Java Cryptology Extensions) di IBM, Sun, Cryptix o IAIK.

Autenticazione di tipo single-sign-on

L'autenticazíone di tipo single-sign-on è la possibilità data a un utente finale, o a un'applicazione, di accedere ad altre applicazioni in un ambiente sicuro senza dover eseguire una procedura di autenticazione per ogni applicazione. L'esempio più comune di utilizzo è offerto dalle applicazioni aziendali intranet che si basano sul Web. In questo ambiente, gli utenti desiderano servirsi di diverse applicazioni che permettono l'accesso a fogli presenze, note spese, piani aziendali e previdenziali: l'autenticazione dell'utente da parte di ogni applicazione sarebbe scomoda e lenta e non limita il valore del sito intranet. L'approccio migliore, in questo caso, consiste nel permettere l'accesso a tutte le applicazioni dopo una verifica iniziale delle credenziali d'accesso, ricorrendo a un profilo in cui sono riportati i diritti di accesso dell'utente.

Molti produttori forniscono soluzioni per l'autenticazíone e l'autorizzazione single-sígn-on sul Web, tra cui aziende come Netegrity, Securant (ora componente di RSA), Oblix e Verísign: in generale, questi prodotti prevedono un processo intermedio che controlla e gestisce il passaggio delle credenziali di un utente da un'applicazione all'altra. All'utente è assegnato un contrassegno, chiamato “ticket" che contiene le informazioni riguardanti i suoi diritti per il collegamento e contestualmente gli consente di avere accesso a diverse applicazioni senza bisogno di essere autenticato da ciascuna di esse. Il contrassegno fa sì che, in un ambiente sicuro, le applicazioni possano delegare l'autenticazione e l'autorizzazione a un terzo attore di fiducia limitandosi all'implementazione della logica di programma.

Il concetto dì single-sign-on può essere esteso con facilità al mondo dei servizi Web, ai quali si può assegnare un contrassegno, inserendolo in un messaggio XML/SOAP, da utilizzare per la loro

validazione presso altri servizi Web. In ogni caso, un utilizzo sicuro dei servizi Web dipende dalla capacità di scambiare le credenziali degli utenti su vastissima scala. I singoli servìzí risiederanno su un certo numero d'ambienti protetti, ciascuno dei quali disporrà di diversi prodotti e tecnologie di sicurezza: poter integrare questi ambienti e consentirne l'interoperabilità è un fattore decisivo per l'impiego efficiente di questi servizi.

Essendo a conoscenza della necessità di fornire specifiche generali per l'autenticazíone single-sign-on dei servizi Web, le industrie più importanti in questo settore si sono riunite per sviluppare uno standard. Il linguaggio SAML (Security Assertíon Markup Language), basato su XML, è una specìfica sostanzialmente completa dell'associazione OASIS (Organìzation for the avancement of Structure Information Standards). L'obíettivo principale di SAML è consentire l'interoperabilità tra diversi sistemi che garantiscono servizi di sicurezza. Le specifiche SAML, non definiscono tecnologie o approcci nuoiì per l'autenticazione o l'autorizzazione: al contrario, si occupano di un linguaggio XML standard che descrive le informazioni o gli output generati da questi sistemi. Microsoft e Sun Microsystems, aloro volta, si sono impegnate entrambe nella realizzazione di sistemi alternativi in grado di offrire le medesime funzionalità, ma limitati alle ríspettive piattaforme.

Molti fornitori d'autenticazione single-sign-on hanno già reso disponibili prodotti che si basano sulle prime versioni di SAML, promettendone l'aggiornamento gratuito quando le specifiche saranno definitive. Il primo a commercializzarli è stato Netegrity, con il rilascio di JSAML per la costruzione di applicazioni Java che utilizzano SAML:questo toolkit, disponibile gratuitamente sul sito Web di Netegrìty, consente ai servizì Web, basatì su Java, dì supportare soluzioni di autenticazíone single-sign -on compatibili con altri ambienti di sicurezza SAML.

Gestione delle chiavi

Una delle sfide principali introdotte dall'utilizzo di queste nuove tecnologie, basate sulla críttografia, la firma digitale e l'autenticazione, sarà quella di riuscire a mantenere catalogate e protette tutte le chiavi pubbliche e private, le firme e i certificati digitali. Al momento sul mercato sono disponibili prodotti PKI progettati per semplificare la gestione di questi componenti di sicurezza; tuttavia, non esiste ancora un metodo standard per avere accesso a questi sistemi in un ambiente di servizi basato su SOAP.

Sviluppato dal Consorzio W3C, le specifiche XKMS (XML Key Management Speciftcation) rappresentano uno dei primi tentativi che tentano di fornire definizioni per transazioni standard basate su XML, che intervengano nella gestione dei servizi di autenticazione, crittografia e firma digitale. XKMS è stato progettato per integrare e migliorare gli standard XML Digital Signature e

XML Encryption, ma non per competere con essi. Queste specifiche XML per la crittografia e la firma digitale delineano le modalità con cui è possibile usare e incorporare le chiavi di crittografia e i certificati digitali, con il presupposto che il servizio Web deputato all'elaborazione del codice XML risieda in un ambiente in cui le chiavi e i certificati siano gestiti in modo sicuro e nel quale il programmatore sappia esattamente quali di essi utilizzare.

XKMS fornirà una serie standardizzata di definizioni XML che permetterà agli sviluppatorí di disporre di un terzo attore fidato che s'incaricherà di localizzare e fornire le chiavi e i certificati appropriati, verificarne la disponibilità, garantirne la validità assumendo il ruolo d'intermediario e liberando il programmatore di servizi Web da questo compito.

In sintesi, XKMS fornirà una serie standardizzata di definizioni XML che consentíranno agli sviluppatori di usare servizi remoti di crittografia a decifrazione offerti da terze parti fidate oltre a quelli di creazione, gestione e autenticazíone di chiavi e firme digitali. Le specifiche prevedono una serie di tag per l'interrogazione dei servizi esterni di gestione chiavi e validazíone firme e altri da usare per l'invio delle risposte: per esempio, un client potrebbe chiedere a un servizio remoto di risposta: “Questo certificato è valido?" oppure "E’ presente un riferimento a una chiave che molto probabilmente stai gestendo tu. Qual’ è il suo valore?".

Recupero della chiave

XKMS offre un metodo semplice per il recupero di una chiave di decifrazione da una fonte remota, basato sul tag <RetrievalMethod> presente nell'elemento <KeyInfo>, secondo la definizione di XML-SIG. Il documento seguente assume che esista un servizio in grado di fornire informazioni su una certa chiave:

```
<ds:KeyInfo>
<ds:RetrievalMethod
URI="http://www.PKeyDir.test/CheckKey"
        Type="http://www.w3.org/2000/09/xmldsig#X509Certificate"/>
</ds:KeyInto>
```

Questa verifica è molto semplice e non obbliga il servizio a eseguire il controllo della validità della chiave restituita.

Servizio di localizzazione

Il location service definisce una serie di tag, utilizzabili dal client applicativo per interrogare un servizio remoto alla ricerca d'informazíoni su una chiave pubblica. Se, per esempio, il client di un servizio Web volesse crittografare un documento basandosi sul valore della chiave pubblica del

ricevente, dovrebbe prima di tutto prendere contatto con il servizio di localizzazione per ottenere questa chiave. Il listato che segue illustra i tag <Locate>, <Query> e <Respond> impiegati nel codice della richiesta:

```
<Locate>
<Query>
        <ds:KeyInfo>
        <ds:KeyName>Aiice Cryptographer</ds:KeyName>
        </ds:KeyInfo>
        </Query>
        <Respond>
        <string>KeyName</string>
        <string>KeyValue</string>
        </Respond>
</Locate>
```

Il tag <Query> fornisce il nome della chiave richiesta e l'elemento <Respond> elenca le voci su cui il clíent vuole avere informazioni. La risposta potrebbe avere l'aspetto seguente:

```
<LocateResult>
        <Result>Success</Result>
        <Answer>
        <ds:KeyInfo>
        <ds:KeyName>Alice Cryptographer</ds:KeyName>
        <ds:KeyValue>Some key value</ds:KeyValue>
        </ds:KeyInfo>
</Answer>
</LocateResuit>
```

Servizio di validazione

Il servizio di validazione è garantito da un terzo attore fidato che effettua la validazione della relazione tra chiave e attributo (per esempio un nome). Supponete di eseguire l'ínterrogazione seguente:

```
<Validate>
        <Query>
        <Status>Valid</Status>
        <ds:KeyInfo>
        <ds:KeyName>...</ds:KeyName>
        <ds:KeyValue> ...</ds:KeyValue>
        </ds:KeyInfo>
</Query>
<Respond>
        <string>KeyName</string>
        <string>KeyValue</string>
</Respond>
</Validate>
```

Il servizio di validazione produrrà i risultati seguenti:

```
<ValidateResult>
        <Result>Success</Result>
        <Answer>
        <KeyBinding>
        <Status>Valid</Status>
        <KeylD>http://www.xmltrustcenter.org/assert/20010120-39</KeyID>
```

```
        <ds:KeyInfo>
                <ds:KeyName> ... </ds:KeyName>
                <ds:KeyValue> ... </ds:KeyValue>
        </ds:KeyInfo>
        <ValidityInterval>
                <NotBefore>2000-09-20T12:00:00</NotBefore>
                <NotAfter>2000-10-20T12:00:00</NotAfter>
        </ValidityInterval>
        </KeyBinding>
</Answer>
</ValidateResult>
```

Nel listato precedente, gli elementi <Result> e <Status> hanno significati diversi. Il valore Success, indicato nell'elemento <Result>, indica semplicemente che la richiesta è stata elaborata con successo dal servizio. L'elemento <Status> contiene i risultati dell'elaborazione e in questo caso corrisponde a Valid. Le informazioni opzionali <ValidityInterval> riportano il periodo di validità dei risultati forniti dal servizio di validazione. I certificati e le chiavi digitali non sono, infatti, validi incondizionatamente: a essi generalmente è associato un limite temporale specifico, trascorso il quale essi scadono e non sono più validi. XKMS definisce, inoltre, le richieste e le risposte per le aree elencate di seguito.

Registrazione della chiave

Registrazione delle informazioni relative alla chiave presso un servizio di gestione chiavi offerto da terze parti.

Revoca di una chiave

Invio di una richiesta al servizio di gestione chiavi offerto da terze parti, per ínformarlo che non volete che gestisca la chiave al vostro posto.

Ripristino della chiave

Nel caso ci si dimentichi la chiave privata, come si potrà inviare una richiesta per ottenerla e come sarà la risposta? Le specifiche non indicano regole per il ripristino della chiave privata; per esempio, il servizio potrebbe revocare la vecchia chiave, rilasciandone una nuova. In ogni caso, questa decisione è assoluta competenza del singolo provider.

Verísign è uno dei primi fornitori di XKMS e ha già rilasciato un toolkit Java che supporta lo sviluppo XKMS.

Le estensioni di sicurezza SOAP

Il programma SOAP 1.1 supporta l'uso delle tecnologie di sicurezza relative a XML, poiché è un contenitore di messaggi basati su XML.

Estensioni SOAP per le credenziali digitali

Come si è già visto nei paragrafi precedenti, le tecnologie di sicurezza delle applicazíoni end-to-end (crittografia, autorizzazione e autenticazione) richiedono lo scambio di credenziali digitali che possono assumere diverse forme: quella più utilizzata è il certíficato digitale conforme allo standard denominato X.509, ma Microsoft ha annunciato recentemente nuovi progetti per supportare anche le cosiddette credenziali Kerberos. Entrambe contengono informazioni relative al proprietario, tra cui quelle relative ai metodi di críttografía e alla sua firma digitale.

Microsoft e IBM hanno proposto di estendere le specifiche SOAP 1. 1 in modo da includere un'intestazione interamente dedicata alle credenziali specifiche per la sicurezza che consentirebbe di standardizzare l'impiego di tipi di credenziali diversi nell'ambito del medesimo messaggio SOAP. Lo scopo di queste estensioni è offrire ai servizi basati su SOAP la possibilità di applicare la firma digitale solo alle relative parti di envelope.

Estensioni SOAP per la firma digitale

Per usare in modo efficace le firme digitali XML, o un altro tipo di firma, nei messaggi SOAP occorre definire una metodologia standard per il loro inserimento nel messaggio.

Per soddisfare questa necessità, IBM e Microsoft hanno proposto una serie di estensioni d'intestazioni per SOAP 1.1: l'obiettivo è permettere agli envelope SOAP di contenere una firma digitale relativa a uno o più elementi del programma. Di seguito è mostrato un esempio dell'uso di tali estensioni per la firma digitale:

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
<SOAP-ENV:Header>
<SOAP-SEC:Signature
 xmins:SOAP-SEC="http://Schemas.xmlsoap.org/soap/Security/2000-12"
 SOAP-ENV:actor="some-URI"
 S0AP-ENV:mustUnderstand="1">
 <ds:Signature xmins:ds="http://www.w3.org/2000/09/xmldsig#">
       <ds:Signedinfo>
       <ds:CanonicalizationMethod
              Algorithm="http://www.w3.orgfrR/2000/CR-xmL-cl4n-20001026">
       </ds:CanonicalizationMethod>
       <ds:SignaturoMethod
              Algorithm="http://www.w3.org/2000/09/Xmldsig#dsa-shal"/>
     <ds:Reference URI="Body">
       <ds:Transforms>
        <ds:Transform
            Algorithm="http://www.w3.org/TR/2000/CR-xml-cl4n-20001026"/>
       </ds:Transforms>
       <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
       <ds:DigestValue>j61wx3rYEPOOvKtMup4NbeVu8nk=</ds:DigestValue>
     </ds:Reference>
    </ds:Signodinfo>
  <ds:SignatureValue>MCOCFFrVLtRik= ... </ds:SignatureValue>
 </ds:Signature>
```

```
 </S0AP-SEC:Signature>
</S0AP -ENV:Header>
 <SOAP-ENV:Body
  xmlns:SOAP-SEC="http://schemas.xmisoap.org/soap/security/2000-12"
  S0AP-SEC:id="Body">
  <m:GetLastTraePrice xmlns:m="some-URI">
  <m:symbol>IBM</m:symbol>
  </m:GetLastTraePrice>
 </S0AP-ENV:Body>
</SOAP-ENV:Envelope>
```

La firma digitale XML appartiene a un namespace specifico ed è contenuta nell'elemento <ds: Signature>. Il tag destinato a contenere la firma è <SOAP-SEC:Signature> che specifica il namespace della firma e il lettore a cui è destinata, contrassegnato dall'elemento <actor> che può essere un nodo intermedio SOAP oppure il destinatario del messaggio. L'attributo SOAP-ENV:mustUnderstand indica ai nodi intermedi che, nel caso non riescano a interpretare questo attributo d'intestazione, devono evitare di elaborarlo.

Queste estensioni costituiscono un metodo standard per aggiungere firme digitali ai messaggi SOAP: abilitando l'intestazione SOAP all'ímpiego dell'estensione <SOAP-SEC:Signature>, qualsiasi servizio Web può introdurre in un messaggio SOAP ogni tipo di firma.

Questa proposta offre un supporto flessibile all'impiego della crittografia XML per rendere parti sicure di messaggi SOAP: attualmente è allo studio l'aggiunta di un tag <SOAP-SEC:Encryption>.

Specifiche per la sicurezza

Abbiamo detto che esistono diverse specifiche che hanno tutte le caratteristiche giuste per superare i problemi inerenti la sicurezza in ambito dei servizi web e che si trovano a vari livelli di sviluppo.
Le principali sono: *Saml* (Security assertion markup language) e *Ws-Security,* entrambi standard OASIS.

Come mai all'OASIS e non al W3C (World Wide Web Consortium, l'organismo di standardizzazione più autorevole nel mondo Internet)? Di fatto, perché l'approvazione dello standard sta assumendo un carattere di urgenza e la burocrazia del W3C viene sempre più considerata con ostilità dai produttori che sentono la necessità di reagire con rapidità alle esigenze del mercato. Di fatto, l'OASIS è a sua volta un consorzio no-profit estremamente autorevole, che tra i propri membri conta oltre 400 imprese di 100 diverse nazioni del mondo e che nel passato ha già definito numerosi standard nelle aree dell'e-Business, in particolare per quanto riguarda l'EDI, l'XML e, più di recente, anche per i Web Services. Il primo standard in area sicurezza ed

infrastrutture Web a portare la firma OASIS è stato SAML. Per quanto riguarda WS-Security, il primo meeting del Comitato tecnico istituito dall'OASIS per la messa a punto e l'approvazione dello standard si è tenuto il 4-5 settembre 2002 presso Sun Microsystems. Il Comitato, che condurrà le proprie attività partendo dal documento di specifiche e dalla Roadmap pubblicata nell'Aprile 2002 nel documento realizzato congiuntamente da IBM, Microsoft e Verisign, opererà in continua sintonia con una serie di altri comitati impegnati nella definzione di componenti per la sicurezza in ambienti Web, tra i quali quelli delle organizzazioni:

- OASIS: Access Control (XACML - per il controllo degli accessi); XML Common Biometric Format (XCBF - per la descrizione dei dati di tipo biometrico); Provisioning (PSTC - per l'interscambio dei dati sull'identità degli utenti); Rights Language (XrML - per la gestione dei diritti di intervento); Security Services (SAML - per i servizi di autenticazione ed autorizzazione).
- W3C: XML Signature, XML Encryption, XML Key Management.
  XML Signature, che è stato impostato in modo tale da supportare l'apposizione di varie firme da parte di attori diversi, opera in abbinamento ai meccanismi di sicurezza basati su Token e si occupa di assicurare l'integrità dei messaggi per garantire che questi non vengano manipolati lungo il loro percorso. XML Encryption funziona in modo analogo, ma garantisce la riservatezza dei messaggi SOAP. I suoi meccanisimi di crittografia sono stati progettati in maniera da essere pronti a recepire qualsiasi tecnologia, processo o strumento di cifratura.

Salm

La proposta avanzata da OASIS lo scorso maggio, divenuta poi standard OASIS a novembre, può risolvere diverse delle questioni aperte. SAML, security assertion markup language, è un framework basato su XML e indipendente dai vendor per lo scambio di informazioni di sicurezza, le 'assertion' appunto, tra business partner via Internet.
SAML è stato pensato per consentire molta di quella interoperabilità che si è dimostrata assolutamente necessaria tra i prodotti di sicurezza e i sistemi di gestione degli accessi sicuri ai siti Web. Con SAML versione 1.0 gli utenti dovrebbero essere in grado di effettuare il log-in su un sito Web e trasferire le proprie credenziali automaticamente sui siti partner. Lo scopo di SAML è realizzare un framework unificato che sia in grado di convogliare le informazioni di sicurezza dell'utente quando interagisce con un sito - qualcosa di simile a una lingua franca delle credenziali di sicurezza che permetta alle aziende di scambiarsi informazioni senza modificare i sistemi di sicurezza interni. SAML non si occupa, tuttavia, dei problemi di privacy, lasciati del tutto alla responsabilità dei gestori dei siti, e non si tratta di una nuova tecnologia o di un approccio diverso

al problema dell'autenticazione. Al contrario, definisce degli schemi che inquadrano le 'assertion' di sicurezza e i protocolli strutturali dei documenti che trasportano le informazioni di sicurezza.

Specifiche di Salm

SAML è stato ideato per funzionare sui meccanismi di trasporto più comuni, come HTTP, SMTP, FTP e alcuni framework XML, tipo SOAP ed eb-XML. Fornisce un metodo standard per definire l'autenticazione dell'utente, le autorizzazioni e gli attributi all'interno di un documento XML. Le componenti principali sono:

- *assertion*: ce ne sono di tre tipi diversi, ma sono sempre dichiarazioni di fatti riguardanti l'utente, sia esso una persona fisica o un computer. Le assertion di autenticazione richiedono che un utente provi la propria identità. Gli attributi (*attribute assertion*) contengono i dettagli specifici dell'utente, come la nazionalità o il tipo di carta di credito. I permessi (*authorization decision assertion*) identificano invece quali operazioni un utente può compiere (per esempio autorizzazioni all'acquisto di un bene);

- *request/response protocol*: definisce come si richiedono e ricevono le assertion. SAML supporta messaggi SOAP su HTTP in questo momento, ma in futuro saranno supportati altri meccanismi di trasporto;

- *bindings*: fornisce i dettagli esatti su come le richieste SAML devono essere mappate nei protocolli di trasporto, come SOAP su HTTP;

- *profili*: sono le istruzioni su come inserire o trasportare le assertion SAML tra sistemi di comunicazione diversi;

Il punto interessante è che SAML, pur definendo 'assertion' specifiche su un utente e sulle sue credenziali, di fatto non li autentica né autorizza. La parte di autenticazione è svolta da un authentication server, eventualmente collegato a un server LDAP. Quello che SAML realizza è il collegamento, una volta terminato con successo il processo di autenticazione, con altri siti o sistemi, e propaga delle 'assertion' basate sul risultato del processo di autenticazione.

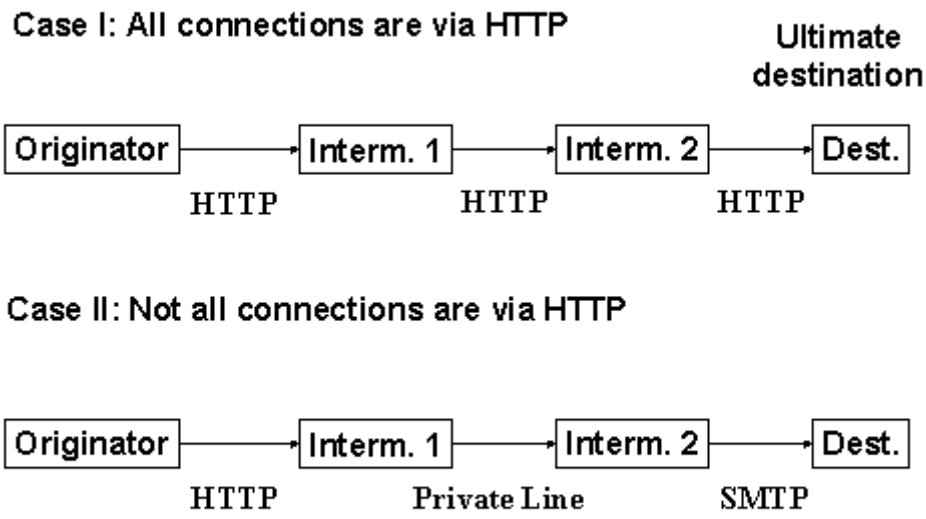
L'obiettivo di Saml è molto ambizioso: controllare lo scambio delle informazioni di sicurezza tra i diversi intermediari Soap (per esempio le applicazioni che codificano alcuni dei dati di una carta di credito prima di passare le informazioni ai processi che eseguono l'ordine).

Capitolo 8 – WS-Security 1.0

La mancanza di standard nell'ambito della sicurezza, oltre a quelli per la semantica, costituiva uno dei freni più rilevanti per la vera diffusione dei Web Services. Così, come già fatto nel passato, IBM, Microsoft e, per l'occasione, VeriSign si sono unite per definire congiuntamente uno standard, al quale è stato dato il nome di WS-Security, che nell'estate scorsa è stato offerto all'OASIS Group affinché fosse recepito ed esteso a tutta la comunità dei produttori di tecnologie e di applicazioni per il Web e l'e-Business. La prima versione della specifica WS-Security è quindi datata 5 Aprile 2002. Dopo di che si sono rapidamente aggiunti nel supporto di questo standard, Sun e la quasi totalità dei produttori di tecnologie di sicurezza tra i quali Baltimore Technologies, Entrust, Netegrity, Oblix, RSA Security e Novell. Anche sul fronte dei produttori di altre tecnologie, WS-Security ha raccolto numerosi consensi, compresi quelli di BEA Systems, Fujitsu, Intel, SAP, IONA, Oblix, Blockade Systems, OpenNetwork, Documentum, Sonic Software, per cui già ora gode di ottime prospettivie di affermazione nel prossimo futuro.

WS-Security nasce per ovviare ai problemi legati alla sicurezza della messaggistica SOAP. La sezione 8 della “Soap 1.1 World Wide Web Consort ium Note”, intitolata “Security consideration”, afferma candidamente che “in questo documento non è descritto nessun metodo per la protezione della privacy e dell’integrità dei dati. Questi problemi saranno affrontati in una futura versione.” La release 1.2 di SOAP dal punto di vista della sicurezza continua a presentare le stesse carenze.

Molti considerano SOAP un meccanismo per lo scambio di messaggi tra due endpoint su HTTP. Su HTTP è possibile autenticare il mittente, firmare il messaggio e crittografare il contenuto del messaggio. Il messaggio viene protetto sotto vari aspetti: il mittente è noto, il destinatario può verificare che il messaggio non abbia subito modifiche durante il transito e le entità che osservano il traffico non possono venire a conoscenza dei dati che vengono scambiati. Tuttavia, la messaggistica SOAP non è in grado di offrire una soluzione efficace per problemi più complessi, per il semplice motivo che la protezione basata su HTTP non è sufficiente. Molti di questi problemi sono legati all'invio dei messaggi lungo percorsi più articolati rispetto ai semplici scambi di richieste e risposte oppure su trasporti che non impiegano HTTP. Ad esempio, può essere necessario preservare l'identità, l'integrità e la protezione dei messaggi in più passaggi, oppure lungo il percorso potrebbero venire utilizzate più chiavi di crittografia, o magari il messaggio deve attraversare più domini trust. HTTP e i suoi meccanismi di protezione rappresentano una soluzione efficace solo per la protezione point-to-point. Per soluzioni più complesse è necessario ricorrere alla protezione end-to-end. WS-Security è in grado di garantire un contesto protetto in percorsi multi-point.



Specifiche di WS-Security

WS-Security si occupa di estendere i messaggi SOAP per garantire un meccanismo standard e sicuro per lo scambio di informazioni tra Web Services. Con WS-Security si vuole garantire l'integrità, la confidenzialità e l'autenticazione all'interno di un unico messaggio.

WS-Security affronta il problema della protezione facendo ricorso a standard e a specifiche esistenti. Questo evita la necessità di definire una soluzione di protezione completa all'interno della specifica WS-Security e tale flessibilità permetterà in futuro alla specifica di evolvere e integrare al suo interno le nuove tecnologie e architetture di sicurezza mantenendo però inalterata la propria struttura e quindi l'interazione con il mondo delle soluzioni applicative. Molti problemi, infatti, sono già stati risolti dal settore. Per l'autenticazione sono state sviluppate le specifiche Kerberos e X.509. Inoltre, la specifica X.509 utilizza l'infrastruttura PKI esistente per la gestione delle chiavi. Le specifiche XML Encryption e XML Signature descrivono come crittografare e firmare il contenuto dei messaggi XML. La specifica XML Canonicalization descrive come predisporre il contenuto XML per la firma e la crittografia. Il valore aggiunto offerto da WS-Security rispetto alle specifiche esistenti è costituito da un'infrastruttura che consente di incorporare questi meccanismi nei messaggi SOAP. Ciò avviene in modo neutrale rispetto al trasporto.

In WS-Security viene definito un elemento intestazione SOAP (SOAP Header) che trasporta i dati relativi alla protezione. Se si utilizza la firma XML (XML Signature), questa intestazione può contenere le informazioni definite dallo standard XML Signature che indicano in che modo il

messaggio è stato firmato, quale chiave è stata usata e il valore risultante della firma. In modo analogo, se un elemento all'interno del messaggio è crittografato, l'intestazione di WS-Security può contenere le informazioni di crittografia, ad esempio quelle specificate dallo standard XML Encryption. WS-Security non specifica il formato della firma o della crittografia. Specifica, invece, come incorporare in un messaggio SOAP le informazioni sulla protezione definite in base ad altre specifiche. WS-Security è in primo luogo una specifica per un contenitore di metadati sulla protezione basati su XML.

Ma oltre a sfruttare protocolli esistenti per l'autenticazione, l'integrità e la riservatezza dei messaggi, WS-Security specifica un meccanismo per il trasferimento di semplici credenziali dell'utente mediante l'elemento *UsernameToken*. Definisce, inoltre un elemento *BinarySecurityToken* per l'invio di token binari utilizzati per la crittografia o la firma dei messaggi. Nell'intestazione i messaggi possono memorizzare informazioni sul mittente, sul modo in cui il messaggio è stato firmato e sul modo in cui è stato crittografato. Poiché mantiene tutte le informazioni sulla protezione nella parte SOAP del messaggio, WS-Security rappresenta una soluzione end-to-end per la protezione dei servizi Web.

Finalità di WS-Security

WS-Security quindi cerca di trasferire molti dei concetti relativi a identificazione e autorizzazione nel mondo della messaggistica SOAP. Per poter ottenere risultati significativi con un messaggio SOAP, il messaggio deve contenere informazioni che:

- Consentono di identificare l'entità o le entità interessate dal messaggio.
- Provano che le entità appartengono ai gruppi corretti.
- Provano che le entità godono dei diritti di accesso corretti.
- Provano che il messaggio non ha subito modifiche.

Infine, deve esistere anche un meccanismo in grado di celare le informazioni riservate a tutti tranne che ai reali destinatari. Grazie all'impiego di token di protezione per identificare il chiamante e accertarne i diritti, un messaggio è in grado di produrre le informazioni seguenti:

- Identità del chiamante: io sono l'utente Bill.
- Appartenenza a un gruppo: io sono uno sviluppatore di Microsoft.com
- Diritti: dal momento che sono uno sviluppatore di Microsoft.com, posso creare database e applicazioni Web nei computer di Microsoft.com.

Per creare un messaggio che possa creare un nuovo database nei server di Microsoft.com utilizzando una tecnologia di autenticazione quale Kerberos, l'applicazione deve ottenere vari

token di protezione. Per iniziare, l'applicazione che crea il messaggio deve ottenere un token di protezione che indichi che il messaggio agisce per conto dell'utente Bill. L'utente Bill specifica tale token quando esegue l'accesso con nome utente/password o con una smart card. Presupponendo che l'infrastruttura di protezione utilizzi Kerberos, l'ambiente utilizzato da Bill dispone di un Key Distribution Center che quando Bill esegue l'accesso gli concede un TGT (Ticket Granting Ticket, ticket di concessione di ticket). Quando Bill decide di creare un nuovo database in Microsoft.com , l'ambiente si rivolge a un servizio TGS (Ticket Granting Service, servizio di concessione ticket) e richiede un ST (Service Ticket, ticket di servizio) che dimostri che Bill è autorizzato a creare un nuovo database in Microsoft.com. L'ambiente recupera tale ST e lo presenta al server di database di Microsoft.com. Il server di database convalida il ticket, quindi autorizza Bill a creare il nuovo processo.

WS-Security cerca di incapsulare le interazioni sopra descritte in una serie di intestazioni SOAP.

Con il nome di *Binary Security Token* si indica invece la possibilità di inserire e di effettuare l'encoding in XML di informazioni binarie come i certificati X509 v3 oppure i ticket TGT (Ticket Granting Ticket), servizio di convalida basato su nome utente/password e ST (Service Ticket) di Kerberos (Listato 3).

LISTATO 8.1 Struttura di un BinarySecurityToken

```
<wsse:BinarySecurityToken
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
   Id="myToken"
   ValueType="wsse:X509v3"
   EncodingType="wsse:Base64Binary">
   MIIEZzCCA9CgAwIBAgIQEmtJZc0...
</wsse:BinarySecurityToken>
```

Questo tipo di token offre nella maggior parte degli scenari applicativi una migliore garanzia di sicurezza.

Figura 8.2. Flusso di messaggi tipico.

Questo servizio può anche non essere basato sui servizi Web. In realtà, sarebbe possibile accedere a un servizio TGS di Kerberos tramite i protocolli Kerberos utilizzando le funzioni di protezione del sistema operativo. Dopo aver ottenuto i token per il messaggio, il client provvede a incorporarli nel messaggio. Il client deve firmare il messaggio con dati ignoti a terzi. Il server potrà ricavare la firma in vari modi: Se il client utilizza un elemento UsernameToken per l'autenticazione, allora deve inviare una password sottoposta a hash e firmare il messaggio utilizzando tale password. Se le firme generate dal server per il messaggio corrispondono alle firme contenute nel messaggio, allora il server avrà la conferma che il messaggio è stato inviato dal client.

La password può non essere inviata nel caso l'architettura lo preveda (si ricordi che WS-Security come i Web Services non saranno utilizzati solamente per comunicazioni tra server geograficamente distribuiti ma verosimilmente anche tra componenti appartenenti alla stessa architettura, ipoteticamente all'interno della stessa rete e con esigenze di autenticazioni diverse). In ogni caso sarà compito dell'applicazione reperire tutte le informazioni necessarie per compiere un processo di autenticazione ed eventualmente di impersonation. Nel caso di invio della password è possibile optare per la trasmissione in chiaro o l'hash della password. Resta inteso che nel momento in cui la password viene inserita nel token il Program Manager dovrà farsi carico di prevedere qualche forma di trasporto sicuro delle informazioni, come SSL per HTTP. Il semplice invio dell'hash della password non è sufficiente a proteggere da alcuni tipi di attacco come il replay-attack e il dictionary attack. Per mitigare queste tipologie di attacco è possibile passare alla funzione di hash non la sola password ma anche delle informazioni di timestamp e un nonce generato on demand.

In caso di utilizzo di certificati X.509, il messaggio può essere firmato utilizzando la chiave privata. Nel messaggio il certificato deve essere contenuto in un elemento BinarySecurityToken. Quando si utilizza X.509, chiunque conosca la chiave pubblica X.509 può verificare la firma. Infine, quando si utilizza Kerberos, il messaggio può venire firmato o crittografato con una chiave di sessione incorporata nel ticket Kerberos. Dal momento che il ticket Kerberos sarà cifrato per il destinatario del token, solo il destinatario potrà decrittare il ticket, scoprire la chiave di sessione e verificare la firma.

Nei casi in cui l'autenticazione è importante, è essenziale che i messaggi SOAP siano firmati o crittografati. Perché? Non è sufficiente aggiungere un token di identità valido al messaggio. Questi token, infatti, possono essere prelevati da un messaggio valido ed essere aggiunti a messaggi utilizzati da pirati informatici. Deve esistere la prova che l'identità indicata nel messaggio è la stessa identità che ha creato il messaggio. Se non si utilizza la specifica XML Signature e non si firma il messaggio, non si può avere certezza dell'integrità del messaggio e della validità del token di identità.

LISTATO 8.2 Struttura di un UsernameToken

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Mario</wsse:Username>
                <wsse:Password>;JK123</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
            ...
    </S:Header>
    ...
</S:Envelope>
```

Un *Username Token* (Listato 8.2) indica semplicemente una struttura per inserire nel messaggio SOAP l'informazione del nome utente e opzionalmente della password.

WS-Security possiede tre diverse variazioni per l'elemento UsernameToken che vengono mostrate di seguito:

```
<!-- Niente password -->
<UsernameToken>
  <Username>Bob</Username>
</UsernameToken>

<!-- Password non crittografata -->
<UsernameToken>
  <Username>Bob</Username>
  <Password Type="wsse:PasswordText">Opensezme</Password>
</UsernameToken>

<!-- Digest: hash SHA1 della password codificata in base64 -->
<UsernameToken>
  <Username>Bob</Username>
  <Password Type="wsse:PasswordDigest">
    QSMAKo67+vzYnU9TcMSqOFXy14U=
  </Password>
</UsernameToken>
```

La prima non contiene una password; è quindi un'opzione valida per quelle situazioni in cui un altro meccanismo viene utilizzato per l'autenticazione e il token nome utente viene utilizzato soltanto per agevolare l'identificazione. La seconda opzione contiene una password non crittografata. Il processo di autenticazione all'altro capo implica il controllo nel database con i nomi utenti e password valide per stabilire se c'è corrispondenza. La terza opzione invia un digest della password invece che una password non crittografata. L'aspetto positivo di questo approccio consiste nel fatto che la password non viene inviata attraverso la rete ed è quindi impossibile identificarla da parte di estranei. Quello negativo è che un estraneo potrebbe inviare una password hash ed essere autenticato come mittente originale.

Per evitare questo problema, è stato aggiunto il supplemento *Web service Security Addendum* come ulteriore protezione. Invece che inviare un hash della password, il supplemento indica che si

debba inviare una versione digest della password. Il digest contiene un hash che è una combinazione della password, di un parametro nonce (una stringa univoca che identifica la richiesta) e della data di creazione. Di conseguenza, non ci possono essere due hash della password uguali. Di seguito viene mostrata una versione di UsernameToken modificato.

```
<!-- UsernameToken rivisto -->
<UsernameToken>
  <Username>Joe</Username>
  <Password Type="wsse:PasswordDigest">
    TPd+eP29FfJLbHpUnAyKGLiTa10=
  </Password>
  <Nonce>FHi7zbHpTmfsk/1f2xV3ow==</Nonce>
  <Created >2002-08-14T17:33:27Z</Created>
</UsernameToken>
```

Sebbene ogni richiesta legittima possieda un hash diverso, è necessario guardarsi da utenti senza scrupoli che si impossessano dell'UsernameToken della richiesta legittima di qualcun altro per utilizzarlo nelle loro richieste non legittime. È possibile minimizzare il rischio impostando la scadenza del `Timestamp` su un tempo sufficientemente breve e imponendo una scadenza al server. Ad esempio, si può indicare che il messaggio scade dopo 30 secondi in modo tale da non venire propagato in ritardo, e quindi comunicare al server di non accettare UsernameTokens 30 secondi dopo il tempo immesso nell'elemento `Created`. In questo caso, però, un problema di sincronizzazione tra gli orologi di diversi computer potrebbe comportare il rifiuto di una richiesta valida, e in certe circostanze consentire richieste passate. L'utilizzo della data di creazione non rappresenta quindi una soluzione definitiva. Per eliminare il rischio, un servizio Web potrebbe mantenere una tabella dei valori `Nonce` degli UsernameTokens ricevuti più di recente e non consentire alcuna richiesta se il parametro Nonce è già stato utilizzato. La tabella dovrebbe contenere le voci dei parametri nonce fino alla loro scadenza. Se si ricevono più richieste con lo stesso parametro Nonce, occorre eliminarle entrambe dal momento che la richiesta non legittima potrebbe essere quella ricevuta per prima. Inoltre, occorre ricordare che il controllo dei parametri Nonce non può fare nulla nel caso in cui un estraneo blocca un messaggio in arrivo prima di raggiungere la destinazione e lo sostituisce con un altro messaggio utilizzando l'UsernameToken del messaggio originale. Per proteggere il messaggio da questo tipo di attacchi, è necessario aggiungere una firma digitale.

Ovviamente tutta la protezione hashing di questo mondo non preclude il fatto che sia il mittente che il destinatario debbano conoscere la password dell'utente. Dal lato del client, ci si aspetta che agli utenti venga richiesta la loro password. Dal lato del server, tuttavia, c'è la necessità di una tabella che contenga le coppie utenti e password valide.

WS-Security introduce alcuni elementi base che compongono la specifica: il *claim*, il *security-token*, il *proof-of-possession*, *l'integrità*, la *confidenzialità*, *l'hash* e *la firma digitale*.

- Il claim rappresenta un oggetto o un privilegio che il client possiede, come una chiave, una password, oppure l'appartenenza ad un gruppo.
- Il security token è semplicemente un insieme di claim. In questo caso viene fatta una distinzione quando il security token è firmato, come nel caso di un certificato X509V3 o dei ticket Kerberos, definendo così i "Signed Security Token".
- Il proof-of-possession, ovvero la prova di possesso, indica l'azione di verifica compiuta da un'entità verso un'altra entità, che quest'ultima è a conoscenza di un "segreto".
- l'integrità, la confidenzialità, l'hash e la firma digitale, fanno parte della sfera di crittografia e come tale WS-Security si basa su specifiche già adottate estendendone, in alcuni casi, l'utilizzo. Ad esempio nella specifica di XMLDSIG viene definito come legare codice XML a delle chiavi crittografiche, mentre in WS-Security, partendo da queste regole, si estendono queste associazioni definendo una relazione tra la firma elettronica e i claim.

LISTATO 8.3 WS-SECURITY

```
<?xml version="1.0" encoding="utf-8"?>

<S.Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
          xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<S:Header>
    <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
      <m:action>http:// fabrikam123.com/getQuote</m:action>
      <m:to>http://fabrikam123.com/stocks</m:to>
      <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641cld6</m:id>
    </m:path>
    <wsse:Security
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
       <wsse:UsernameToken Id="MyId">
          <wsse:Username>Zoe</wsse:Username>
       </wsse:UsernameToken>
      <ds:Signature>
        <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm= "http://
            www.w3.org/2001/10/xml- exc-c14n#"/>
        <ds:SignatureMethod Algorithm="http://
            www.w3.org/2000/09/xmldsig#hmac-sha1"/>
        <ds:Reference URI="#MsgBody">
          <ds:DigestMethod Algorithm=" http://
              www.w3.org/2000/09/xmldsig#hmac-sha1"/>
          <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
        </ds:Reference>
        </ds:SignedInfo>
      <ds:SignatureValue<DJbchm5gK....</ds:SignatureValue>
      <ds:KeyInfo>
           <wsse :SecurityTokenReference>
           <wsse:Reference URI="#MyID"/>
           <wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
</S:Header>
  <S:Body Id="MsgBody">
    <tru:StockSymbol xmlns:tru="http:// fabrikam123.com/payloads">
         QQQ
    </tru:StockSymbol>
  </S:Body>
</S:Envelope>
```

WS-Routing

WS-Security

Operazione di Binding

WS-Security non è in grado di risolvere tutti i problemi di sicurezza delle applicazione basate sui Web Services, e i software architect e gli sviluppatori non devono dedurre che il semplice utilizzo di questa specifica renda le applicazioni automaticamente immuni da ogni possibile attacco. Come sempre avviene per la gestione corretta degli aspetti di sicurezza di applicazioni e architetture, è necessario integrare all'interno dell'intero ciclo di vita del prodotto una completa gestione delle Operations tramite il Security Assessment, il Risk Management e il Threat analysis (Microsoft a questo proposito offre due soluzioni complementari : MOF- Microsoft Operations Framework e MSF – Microsoft Solutions Framework).

Intestazione SOAP

La specifica WS-Security definisce una nuova intestazione SOAP. Per comprendere il contenuto dell'intestazione SOAP di WS-Security è utile esaminare, in primo luogo, la parte dello schema relativo all'elemento.

```
<xs:element name="Security">
    <xs:complexType>
        <xs:sequence>
        <xs:any processContents="lax"
            minOccurs="0" maxOccurs="unbounded">
        </xs:any>
        </xs:sequence>
        <xs:anyAttribute processContents="lax"/>
    </xs:complexType>
</xs:element>
```

L'elemento intestazione Security consente a qualsiasi attributo o elemento XML di esistere al suo interno. In questo modo l'intestazione è in grado di adattarsi a qualsiasi meccanismo di protezione necessario per l'applicazione. Per chiarire il punto, è sufficiente pensare a come funzionano l'intestazione e il corpo SOAP. Sia l'intestazione sia il corpo possono contenere un insieme di elementi XML. Pur non ponendo particolari vincoli per quanto riguarda il contenuto di questi elementi, la specifica SOAP stabilisce che non possono essere istruzioni di elaborazione XML.

Questo tipo di struttura è necessaria per poter consentire all'intestazione di svolgere il suo compito. Infatti, deve poter trasportare più token di protezione per identificare i diritti e l'identità del chiamante. Se il messaggio è firmato, l'intestazione deve contenere informazioni su come è stato firmato e su dove sono memorizzate le informazioni relative alla chiave. La chiave può essere nel messaggio o altrove, nel qual caso il messaggio conterrà un riferimento ad essa. Infine, l'intestazione deve poter trasportare le informazioni sulla crittografia.

Quindi, come fa un intermediario a sapere quale intestazione di WS-Security è destinata a lui? Un messaggio SOAP può contenere più intestazioni di WS-Security e ogni intestazione è identificata da un attributo actor univoco. Non esisteranno mai due intestazioni di WS-Security con lo stesso attributo actor, né intestazioni di WS-Security prive di tale attributo. È grazie a questo meccanismo che un intermediario è in grado di sapere quale intestazione di WS-Security contiene le informazioni di cui ha bisogno. Naturalmente l'intermediario deve conoscere l'URI associato all'attributo actor. Associare un URI a un attributo actor e far sì che l'intermediario sappia quali compiti deve svolgere sono due fattori che devono essere gestiti a livello di programmazione. L'attributo actor delle intestazioni SOAP significa "questa intestazione è destinata a un endpoint in grado di svolgere la funzionalità indicata dall'URI dell'attributo actor". Ma che cosa o chi attribuisce questo significato all'URI? Il team che crea il servizio Web. Questo implica che un intermediario può assolvere a varie funzioni. Di conseguenza, l'intermediario potrebbe utilizzare una sola intestazione, o nessuna, o magari più intestazioni. Sì, potrebbe anche utilizzare più intestazioni di protezione.

WS-Security Addendum

Dopo un periodo di valutazione di WS-Security erano emersi alcuni punti che richiedevano maggior chiarezza, in particolare per la protezione. Inoltre, era risultato necessario specificare ulteriori elementi per i servizi Web in generale. Vediamo le parti dell’Addendum relative alla protezione, in particolare due elementi che sono specifici della protezione: wsu:Id e wsu:Timestamp.

wsu:Id

L'attributo Id utilizza il tipo ID dello schema XML. Questo elemento è stato aggiunto per semplificare l'elaborazione per gli intermediari dei servizi Web e per i destinatari. Il valore dell'attributo deve essere unico nel documento. L'addendum non specifica in dettaglio in che modo l'elemento deve essere utilizzato, a parte il fatto che deve essere utilizzato come identificatore univoco nelle specifiche GXA. In questo modo viene lasciato ampio spazio ad altre specifiche per quanto riguarda i limiti di utilizzo di Id.

wsu:Timestamp

Nei sistemi orientati ai messaggi, l'attualità dei dati riveste particolare importanza. I messaggi con dati troppo vecchi possono venire scartati. In presenza di due messaggi contraddittori, è possibile utilizzare i relativi timestamp per decidere quale deve essere eseguito e quale ignorato. Per gestire le questioni relative al tempo che sono emerse in WS-Security e che emergeranno in altre specifiche GXA, è stato definito l'elemento `wsu:Timestamp` e alcuni elementi di supporto.

Gli eventi di interesse nella vita di un messaggio sono la data e l'ora di creazione, la data e l'ora di scadenza fissate dal mittente e la data e l'ora di ricezione. Conoscendo la data e l'ora di creazione e di scadenza, il destinatario può decidere se i dati sono sufficientemente recenti oppure se sono diventati talmente vecchi da dover essere scartati. Gli elementi che trasmettono questi dati sono:

- `wsu:Created`: contiene la data e l'ora di creazione del messaggio.
- `wsu:Expire`s: impostato dal mittente o da un intermediario, indica la data e l'ora di scadenza del messaggio.
- `wsu:Received`: indica la data e l'ora di ricezione del messaggio da parte di un dato intermediario.

Tutti questi elementi possono comparire in modo indipendente oppure come parte di un elemento `wsu:Timestamp`. Ognuno può contenere un attributo `wsu:Id`, che lo identifica in modo univoco. Per impostazione predefinita, in questi timestamp la data e l'ora vengono espressi con il tipo `xs:dateTime`. Per garantire la flessibilità necessaria per accettare anche altri timestamp non standard che potrebbero essere significativi in altri ambiti, ognuno di questi elementi contiene anche un attributo ValueType. Questo attributo non deve essere specificato se la data e l'ora sono espressi con il tipo `xs:dateTime`.

L'elemento `wsu:Received` accetta due attributi aggiuntivi, non utilizzabili negli elementi `wsu:Created` e `wsu:Expires`. L'elemento può esprimere l'URI del proprio attributo actor mediante l'attributo Actor, e il ritardo, espresso in millisecondi, causato dall'attributo actor mediante l'attributo Delay.

Come già accennato in precedenza, è possibile utilizzare gli elementi `wsu:Received`, `wsu:Created` e `wsu:Expires` all'interno di altre strutture. Ad esempio, l'elemento `wsu:Created` può venire utilizzato per indicare data e l'ora in cui un particolare elemento è stato aggiunto al messaggio. Nel caso in cui sia necessario fornire più informazioni su un messaggio e utilizzare più di uno di questi elementi alla volta, è possibile inserire gli elementi in un elemento `wsu:Timestamp`. Ognuno dei tre elementi può comparire una sola volta in un timestamp. Il timestamp può essere utilizzato sull'intero messaggio, nel qual caso compare come figlio del nodo `soap:Header`. Ad esempio, un messaggio potrebbe specificare di essere valido per i cinque minuti successivi utilizzando la seguente intestazione `wsu:Timestamp`.

```
<wsu:Timestamp>
    <wsu:Created wsu:Id=
        "Id-2af5d5bd-1f0c-4365-b7ff-010629a1256c">
            2002-08-19T16:15:31Z
    </wsu:Created>
    <wsu:Expires wsu:Id=
        "Id-4c337c65-8ae7-439f-b4f0-d18d7823e240">
            2002-08-19T16:20:31Z
    </wsu:Expires>
```

```
</wsu:Timestamp>
```

Autenticazione

Abbiamo visto che con WS-Security è possibile convalidare un utente in moltissimi modi. La specifica descrive tre metodi:

- Nome utente/Password
- PKI tramite certificati X.509
- Kerberos

In questa sezione verranno descritti questi tre metodi di autenticazione e verrà spiegato in che modo le informazioni vengono codificate in un messaggio SOAP.

Nome utente/password

Uno dei modi più comuni per trasmettere le credenziali del chiamante consiste nel fornire la combinazione di nome utente e password. Si tratta di una tecnica impiegata nell'autenticazione di base e nell'autenticazione classificata (Digest) HTTP. Per poter passare le credenziali dell'utente con questo meccanismo, in WS-Security è stato definito l'elemento UsernameToken. Lo schema dell'elemento è il seguente:

```
<xs:element name="UsernameToken">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Username"/>
            <xs:element ref="Password" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="Id" type="xs:ID"/>
        <xs:anyAttribute namespace="##other"/>
    </xs:complexType>
</xs:element>
```

In questo frammento viene fatto riferimento ad altri due tipi: `Username` e `Password`. Questi due tipi sono essenzialmente stringhe che, all'occorrenza, possono contenere altri attributi. Password contiene un attributo `Type` che indica in che modo la password viene trasmessa. Una password può essere trasmessa come testo semplice o come digest. Quando si trasmette un elemento UsernameToken in un messaggio SOAP, il codice XML sarà analogo al seguente:

```
<wsse:UsernameToken>
    <wsse:Username>giovanni</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">password</wsse:Password>
</wsse:UsernameToken>
```

Questo è un esempio di codice per l'invio della password come testo semplice. Questa soluzione, in specifico, sembra alquanto facile da violare. Se si desidera inviare la password in un formato più protetto, è consigliabile inviarla come hash digest.

```
<wsse:UsernameToken>
    <wsse:Username>giovanni</wsse:Username>
    <wsse:Password Type="wsse:PasswordDigest">
        KE6QugOpkPyT3Eo0SEgT30W4Keg=</wsse:Password>
    <wsse:Nonce>5uW4ABku/m6/S5rnE+L7vg==</wsse:Nonce>
    <wsu:Created xmlns:wsu=
        "http://schemas.xmlsoap.org/ws/2002/07/utility">
            2002-08-19T00:44:02Z
    </wsu:Created>
</wsse:UsernameToken>
```

L'impiego di un hash SHA1 oscura la password, a vantaggio della protezione. Il `digest` della password è dato dalla concatenazione dell'elemento `nonce` più l'elemento `created` più la password. L'elemento `nonce` è lungo 16 byte e viene passato come valore codificato Base64. In pratica, il client crea l'hash della password utilizzando tutte queste informazioni più la password. Il destinatario verifica i dati recuperando la password semplice e ricreando l'hash. Se i risultati sono uguali, allora la password è corretta. Questa protezione non è efficace contro gli attacchi di tipo replay. Se la si utilizza, è importante ricordarsi di includere anche un'intestazione `wsu:Timestamp` con un intervallo sufficientemente piccolo fra data e ora di creazione e data e ora di scadenza. Inoltre, è importante firmare gli elementi `wsu:Timestamp` nel messaggio, in modo che sia possibile scoprire eventuali tentativi di alterazione del timestamp. In caso contrario, un pirata potrebbe utilizzare l'intero elemento UsernameToken per attaccare il servizio Web. Per proteggersi da attacchi di tipo replay è necessario, inoltre, includere un meccanismo che sia in grado di tenere traccia di alcune caratteristiche uniche dei messaggi in arrivo. In pratica, questo meccanismo deve salvare tali caratteristiche in una cache per almeno il periodo di timeout del messaggio.

Certificati X.509

Un'alternativa per l'autenticazione degli utenti consiste nell'inviare semplicemente un certificato X.509. Un certificato X.509 comunica con esattezza l'identità dell'utente. Utilizzando PKI è possibile associare il certificato a un utente esistente nell'applicazione. L'uso del certificato di per sé renderebbe possibili attacchi di tipo replay, quindi è buona norma obbligare il mittente del messaggio a firmare il messaggio con la propria chiave privata. In questo modo, quando il messaggio viene decrittato si viene a conoscere la vera identità dell'utente.

Quando un messaggio reca un certificato X.509, il messaggio passa la versione pubblica del certificato in un token di WS-Security denominato BinarySecurityToken. Il certificato stesso viene inviato sotto forma di dati codificati Base64. Di seguito è riportato lo schema del token BinarySecurityToken:

```
<xs:element name="BinarySecurityToken">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="Id" type="xs:ID" />
                <xs:attribute name="ValueType" type="xs:QName" />
                <xs:attribute name="EncodingType" type="xs:QName" />
                <xs:anyAttribute namespace="##other"
                    processContents="strict" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

Nella sua forma più semplice, questo elemento contiene una stringa, un identificatore univoco e alcune informazioni sul tipo di valore incluso e sulla modalità di codifica. L'elemento ValueType può essere uno dei seguenti valori, definito da ValueTypeEnum nel documento dello schema di WS-Security:

- wsse:X509v3: un certificato X.509 versione 3.
- wsse:Kerberosv5TGT: un TGT, conforme alla definizione della sezione 5.3.1 della specifica Kerberos.
- wsse:Kerberosv5ST: un Service Ticket, conforme alla definizione della sezione 5.3.1 della specifica Kerberos.

Nella prossima sezione verranno approfondite queste osservazioni su Kerberos. L'elemento EncodingType è un altro valore di enumerazione, che può essere impostato su wsse:Base64Binary o wsse:HexBinary. Questo valore indica, semplicemente, quale metodo di codifica è stato utilizzato. In un'intestazione WS-Security, per la trasmissione di un certificato X.509 questo elemento sarebbe analogo al seguente:

```
<wsse:BinarySecurityToken
    ValueType="wsse:X509v3"
    EncodingType="wsse:Base64Binary"
    Id="SecurityToken-f49bd662-59a0-401a-ab23-1aa12764184f"
>MIIHdjCCB...</wsse:BinarySecurityToken>
```

È importante ricordare che quando si utilizza un certificato X.509 è opportuno che il messaggio sia firmato. La firma, creata tramite la chiave privata del certificato, dimostra che il client è il legittimo proprietario del certificato. Un messaggio con queste caratteristiche può essere riprodotto. Per prevenire gli attacchi di tipo replay è possibile definire un criterio che specifichi dopo quanto tempo un messaggio deve venire ignorato. Il periodo di tempo deve essere specificato in un elemento wsu:Timestamp incluso nel messaggio come intestazione SOAP.

Kerberos

Per utilizzare Kerberos, un utente presenta le credenziali sotto forma di nome utente/password o certificato X.509. Se i controlli hanno esito positivo, il sistema concede all'utente un ticket TGT. Il TGT è un dato non trasparente, che l'utente non può leggere ma la cui presenza è essenziale per poter accedere alle risorse. Di norma l'utente presenta il TGT per ottenere un ST. Il sistema funziona in questo modo:

1. Un client viene autenticato da un Key Distribution Center (KDC) e ottiene un TGT.
2. Il client prende il TGT e lo utilizza per accedere a un servizio TGS.
3. Il cliente richiede un ticket ST per una risorsa di rete specifica. Il TGS emette il ticket ST per il client.
4. Il client presenta il ticket ST alla risorsa di rete e inizia ad accedere alla risorsa con le autorizzazioni indicate nel ticket ST.

Kerberos è molto interessante, perché contiene un meccanismo che consente al client di dimostrare la propria identità a un servizio e al servizio di dimostrare la propria identità al client. Il ticket ST è valido solo per l'accesso alla risorsa di rete specificata e può essere utilizzato per scoprire l'identità del mittente. Quando si presenta un ticket Kerberos in un messaggio, i dati devono essere in copia nascosta nel messaggio stesso. WS-Security non spiega in che modo vengono ottenuti i ticket TGT o ST.

Firma

È quasi impossibile alterare i messaggi firmati. La firma non evita che terzi vedano il contenuto del messaggio, semplicemente garantisce al destinatario del messaggio SOAP che il messaggio non ha subito modifiche nel percorso. È consigliabile firmare i messaggi con XML Signature ogni volta possibile. Perché XML Signature gestisce alcuni dei problemi più spinosi oggi esistenti. WS-Security spiega semplicemente come utilizzare la firma per dimostrare che il messaggio non è stato modificato. Tutti e tre i meccanismi di autenticazione sopra menzionati consentono di firmare il messaggio, in modo da poter essere sicuri di due punti:

- L'utente identificato dal certificato X.509, UsernameToken, o dal ticket Kerberos è l'utente che ha firmato il messaggio.
- Il messaggio non è stato alterato dopo l'apposizione della firma.

Entrambi i metodi prevedono una chiave segreta da utilizzare per firmare il messaggio. X.509 consente al mittente di firmare il messaggio utilizzando la propria chiave privata. Kerberos prevede una chiave di sessione che il mittente crea e trasmette nel ticket. Solo il destinatario effettivo del messaggio potrà leggere il ticket, scoprire la chiave di sessione e verificare l'autenticità della firma. Infine, è possibile firmare l'elemento UsernameToken utilizzando la password.

La firma viene generata utilizzando lo standard XML Signature. Per firmare un messaggio semplice, quale "Hello World", è necessario che quasi tutti gli elementi del messaggio siano firmati singolarmente. wsu:Timestamp presenta un problema interessante, perché un intermediario potrebbe aggiungere un elemento wsu:Received a wsu:Timestamp. Ogni volta che un elemento cambia, è necessario aggiornare la firma, altrimenti l'autenticazione ha esito negativo. Questo perché se il contenuto cambia, le firme non corrisponderanno. In un messaggio SOAP le firme e i dati extra richiesti comportano un'aggiunta di informazioni.

Crittografia

In alcuni casi non è sufficiente dimostrare l'identità del mittente del messaggio e provare che il contenuto non è stato modificato. Se si inviasse un numero di carta di credito o un numero di conto bancario come testo semplice, ma firmato, un pirata informatico potrebbe verificare che nessun altro pirata informatico abbia modificato il contenuto del messaggio e avrebbe la certezza che i dati sono validi. Di sicuro non è questo il nostro obiettivo. Al contrario, l'obiettivo è crittografare i dati in modo tale che solo il destinatario reale del messaggio possa leggerli. Chiunque possa osservare lo scambio di informazioni deve rimanere all'oscuro del contenuto del messaggio. Per quanto riguarda la firma dei messaggi, la specifica WS-Security adotta, giustamente, uno standard che esiste già e che, inoltre, prevede anche la crittografia. Infatti, incorpora lo standard XML Encryption.

Quando si sottopongono i dati a crittografia, è possibile scegliere se utilizzare la crittografia simmetrica o asimmetrica. La crittografia simmetrica richiede una chiave segreta condivisa, ovvero la chiave utilizzata per crittografare il messaggio è la stessa chiave che dovrà essere utilizzata per decrittare il messaggio. È consigliabile avvalersi della crittografia simmetrica se si ha il controllo di entrambi gli endpoint e se si ritengono affidabili sia le persone, sia le applicazioni che utilizzano le chiavi. La crittografia simmetrica, tuttavia, presenta un problema in merito alla distribuzione delle chiavi: a un certo momento la chiave deve essere inviata al destinatario. Come avviene questo invio? Inviando un disco per posta o negoziando la chiave all'occorrenza? Entrambe le soluzioni sono valide.

La crittografia asimmetrica, invece, offre un meccanismo di distribuzione delle chiavi molto semplice, reso possibile dai certificati X.509. L'endpoint che riceve i dati può inviare pubblicamente il suo certificato e consentire a chiunque di crittografare le informazioni utilizzando la chiave pubblica. Solo il destinatario conoscerà la chiave privata. Per questo motivo, solo il destinatario potrà convertire i dati crittografati in dati leggibili.

Vediamo ora com'è un messaggio crittografato. Se si utilizza Triple-DES, sia il mittente sia il destinatario devono scambiarsi la chiave in modo sicuro. La chiave simmetrica può essere

nascosta in un ticket Kerberos oppure scambiata fuori banda. Un messaggio basato su WS-Security con informazioni crittografate in base allo standard XML Encryption è analogo al seguente:

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <soap:Header
        xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
        xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
        <wsu:Timestamp>
            <wsu:Created
                wsu:Id="Id-3beeb885-16a4-4b65-b14c-0cfe6ad26800"
                >2002-08-22T00:26:15Z</wsu:Created>
            <wsu:Expires
                wsu:Id="Id-10c46143-cb53-4a8e-9e83-ef374e40aa54"
                >2002-08-22T00:31:15Z</wsu:Expires>
        </wsu:Timestamp>
        <wsse:Security soap:mustUnderstand="1" >
            <xenc:ReferenceList>
                <xenc:DataReference
        URI="#EncryptedContent-f6f50b24-3458-41d3-aac4-390f476f2e51" />
            </xenc:ReferenceList>
            <xenc:ReferenceList>
                <xenc:DataReference
        URI="#EncryptedContent-666b184a-a388-46cc-a9e3-06583b9d43b6" />
            </xenc:ReferenceList>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <xenc:EncryptedData
            Id="EncryptedContent-f6f50b24-3458-41d3-aac4-390f476f2e51"
            Type="http://www.w3.org/2001/04/xmlenc#Content">
            <xenc:EncryptionMethod Algorithm=
                "http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                <KeyName>Chiave simmetrica</KeyName>
            </KeyInfo>
            <xenc:CipherData>
                <xenc:CipherValue
                >InmSSXQcBV5UiT... Y7RVZQqnPpZYMg==</xenc:CipherValue>
            </xenc:CipherData>
        </xenc:EncryptedData>
    </soap:Body>
</soap:Envelope>
```

Il messaggio precedente contiene informazioni su quali dati sono stati crittografati e sul metodo utilizzato per la crittografia. Per chiunque non abbia accesso alla chiave, il testo cifrato contenuto nell'elemento soap:Body non è decifrabile.

Se si utilizza la crittografia asimmetrica, per poter decrittare il messaggio il destinatario deve conoscere la chiave privata. Per quanto riguarda lo scambio della chiave pubblica, è necessario individuare il sistema in anticipo.

Conclusioni

WS-Security consente a un messaggio SOAP di identificare il chiamante, firmare il messaggio e crittografare il contenuto del messaggio. Quando possibile, vengono riutilizzate le specifiche esistenti, allo scopo di sfruttare i meccanismi complessi che consentono di trasmettere i messaggi SOAP in sicurezza. Dal momento che tutte le informazioni sono incluse nel messaggio stesso, il tipo di trasporto utilizzato è indifferente. Il messaggio è ugualmente protetto, indipendentemente dal fatto che venga trasmesso via HTTP, tramite posta elettronica o su CD-ROM.

----------fine Addendum----------

GXA

GXA (Global XML Web Services Architecture) un'insieme di specifiche e linee guida che seguono lo sviluppo dell'infrastruttura interna dei Web Services nei prodotti Microsoft. Tale progetto fonda le sue origini nel 2001 quando Microsoft e IBM stabilirono un documento intitolato "Web Services Framework" nel quale le due società definirono una serie di funzionalità di base in grado di consentire alle applicazioni di interagire in modo sicuro, nonostante l'appartenenza a d infrastrutture diverse. Nel 2001 questo documento fù sottoposto al W3C per la standardizzazione.

Concettualmente, l'architettura GXA può essere suddivisa in due livelli: il primo livello rappresenta i moduli SOAP, ovvero quell'insieme di specifiche che descrivono la struttura di un unico messaggio SOAP che contiene determinate indicazioni.

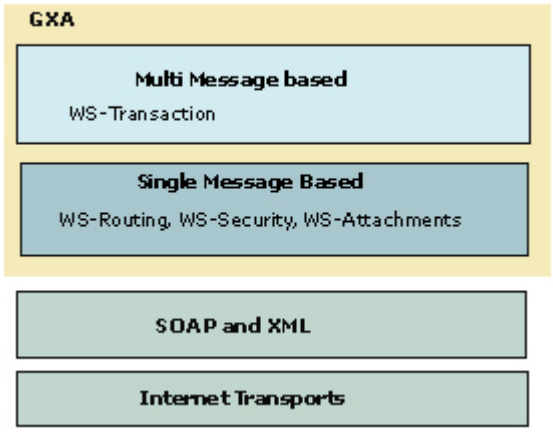
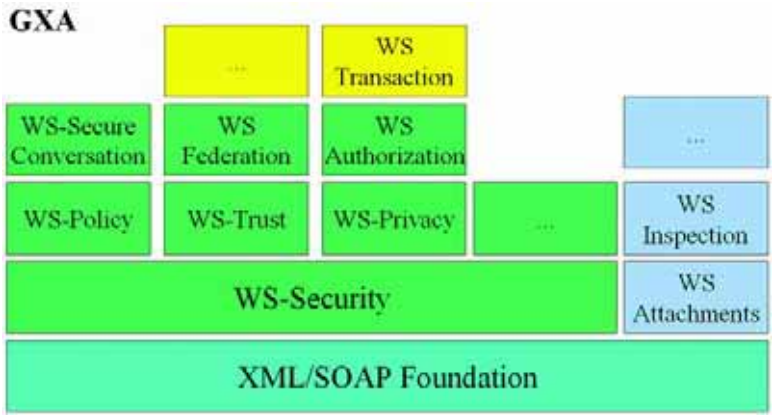


Immagine: Global XML Architecture

Ad esempio, in WS-Security si descrive come il messaggio SOAP possa contenere informazioni sui security token e dati crittografati mentre con WS-Routing viene descritto come referenziare l'instradamento dei messaggi tra zero e n intermediari. Il secondo livello invece raggruppa tutte quelle specifiche che descrivono l'interazione tra più messaggi per definire un'azione comune come ad esempio una transazione. Sebbene nel primo livello siano già disponibili alcune

specifiche, nel secondo livello le informazioni si limitano oggi alle sole specifiche WS-Coordination e WS-Transaction. Oltre a WS-Security, Microsoft e IBM, insieme ad alcuni partner, stanno lavorando in comune ad altre sei specifiche riguardanti la sicurezza: WS-Policy, WS-Trust, WS-Privacy, WS-Secure-Conversation, WS-Federation e WS-Authorization.



I principi che governano la definizione dell'intera architettura GXA e delle singole specifiche sono riassumibili in quattro punti: modularità, federazione, utilizzo di basi standard, generico. Il primo principio, la modularità, determina che tutta l'architettura deve essere costituita da moduli perfettamente integrati e facilmente estendibili. Con la parola federato si indica un'architettura per l'interconnessione tra utenti, applicazioni e servizi, appartenenti anche a infrastrutture diverse per le quali però vi sia una relazione di fiducia. Queste infrastrutture possono essere implementate con tecnologia differente, con diverse modalità di autenticazione e possono essere basate su diversi modelli di programmazione. Il terzo principio, utilizzo di standard, indica da un lato l'esigenza di basare le proprie specifiche su standard di mercato già ampiamente adottati, dall'altro la volontà di Microsoft di lavorare con altre società e clienti per sottoporre queste specifiche alle autorità di standardizzazione. L'ultimo punto, generico, indica che GXA si fa carico dei protocolli orizzontali, cioè non legati a nessun dominio di applicazioni. Un framework deve poter offrire i medesimi servizi di base ad applicazioni B2C, B2B come EAI e P2P.
Come già detto il protocollo base utilizzato in GXA è SOAP. Tutte le specifiche ad oggi rilasciate adottano il medesimo approccio, ovvero estendono gli header SOAP per aggiungere nuove funzionalità. Secondo la specifica SOAP 1.2 un qualsiasi messaggio SOAP è costituito da tre elementi. L'elemento `<soap:Envelope>` che al suo interno può contenere altri due elementi figli : `<soap:Header>` e `<soap:Body>`. L'header di un messaggio SOAP può essere facoltativo mentre il body deve essere sempre presente. La filosofia adottata da tutte le specifiche di GXA è quella di utilizzare il tag `<soap:Header >` per inserire le nuove informazioni sfruttando, solo in alcuni casi, il tag `<soap:Body>` privilegiando l'uso di tale header da parte delle applicazioni dell'utente.

```
<s:Envelope xmlns:s="http://www.w3.org/2001/12/soap-envelope">
   <s:Header>
      <!—Qui vengono inserite le informazioni di infrastruttura (specifiche GXA)
-->
```

```
   </s:Header>
   <s:Body>
       <!—Qui vengono inserite le informazioni applicative -->
  </s:Body>
</s:Envelope>
```

I valori presenti nel tag `<soap:Header>` forniscono i dettagli sulle modalità di elaborazione dei dati presenti nel tag `<soap:Body>`. In questo modo viene rappresentato un sistema "elegante" per distinguere tra i dati di infrastruttura (quelli presenti nell'header) dai dati applicativi (presenti nel body), all'interno dello stesso messaggio.

Come già accennato, GXA è un insieme di specifiche e non un prodotto.

Le specifiche di GXA

GXA attualmente è composta da una decina di specifiche molte delle quali ancora in fase di studio mentre altre già avviate al processo di standardizzazione. Sulla base di WS-Security le principali specifiche che compongono l'architettura GXA sono: *WS-Privacy* e *WS-Policy* indicano gli aspetti di privacy e le security policy da applicare a tutti gli "attori" di una infrastruttura distribuita siano essi endpoint o intermediari. *WS-Trust* si occupa invece della definizione delle relazioni e dei modelli di trust alla base dell'interoperabilità sicura tra i Web Services mentre *WS-Federation* indica come gestire tali relazioni di fiducia in un ambiente eterogeneo.
Al livello superiore troviamo *WS-SecureConversation* e *WS-Authorization* che ricoprono rispettivamente gli aspetti di security context e le policy di autorizzazione e di accesso ai dati. *WS-Inspection* permette di esplorare un sito per conoscere quali servizi sono disponibili mentre *WS-Coordination* definisce un modello consistente per coordinare delle attività complesse. WS-Attachments consente di referenziare attachment dal messaggio SOAP evitando un dispendiosa operazione di encoding e deconding; infine, *WS-Transaction*, basandosi sul modello di WS-Coordination, definisce due tipologie di coordinamento : AT (Atomic Transaction) e BA (Business Activity). Anche in questo caso lo scopo della specifica non è di sostituire il modello e le teorie sulle transazioni, ma al contrario di espandere tali modelli fra infrastrutture diverse.

WS-Routing e WS-Referral

WS-Routing è la nuova versione della specifica già conosciuta come SOAP-RP. Il nome è stato modificato per mantenere una certa consistenza nel referenziare tutte le specifiche di GXA. WS-Routing, insieme a WS-Referral è alla base del concetto di virtualizzazione della rete; uno degli aspetti più importanti e innovativi di GXA. Tramite queste due specifiche è possibile avere una visione astratta dell'intera rete sulla quale viaggiano i messaggi SOAP indipendentemente dalla topologia e dai protocolli di trasporto sottostanti. Un ruolo importante in questo modello è quello ricoperto dagli intermediari SOAP, più comunemente chiamati SOAP router, definiti nella specifica

SOAP 1.2. Un intermediario SOAP viene definito come un SOAP receiver e un SOAP sender, identificato all'interno dello stesso messaggio SOAP, con il compito di processare la propria parte dell'header SOAP, eventualmente effettuare delle operazioni che modificano il messaggio, e infine spedirlo al SOAP receiver successivo che può essere un altro intermediario oppure il destinatario finale. Poichè le informazioni di instradamento risiedono nell'header del messaggio non è necessario, in caso in encryption applicativo, decifrare le parti codificate nel body per estrarre le informazioni del router e di conseguenza non è richiesta l'implementazione di una policy per la distribuzione delle chiavi crittografiche agli intermediari. E' buona norma firmare le informazioni di routing contenute all'interno dell'header per evitare possibili manomissioni durante il tragitto.
WS-Referral è la specifica che permette di gestire le informazioni dei SOAP router presenti in un network tramite una struttura basata su XML chiamata WS-Referral statement, delle funzionalità di interrogazione, e con la possibilità di registrare e cancellare nuovi intermediari.
Questa architettura basata su intermediari SOAP dove ogni hop può appartenere ad architetture differenti e con diversi protocolli può garantire solamente un modello di invio-ricezione messaggi di tipo "datagram" dove i messaggi vengono instradati in modo indipendente e non ne viene garantito l'ordine di arrivo. L'esigenza di definire un protocollo end-to-end in questo scenario è trasferita ad una futura specifica diversa da WS-Routing.
Si noti però che le due specifiche non hanno il compito di sostituire i protocolli di routing standard esistenti in Internet. Al contrario, l'intera architettura dei Web Services utilizza i protocolli di Internet e i DNS come qualsiasi altra entità all'interno della rete. Ciò che giustifica queste due specifiche è la volontà di creare un sistema di messaggistica di base asincrono, "architecture independent", valido per tutte le altre specifiche GXA. In questi nuovi scenari di integrazione il semplice utilizzo del TCP/IP come unica forma per il routing dei messaggi SOAP diventava molto restrittivo per due ragioni: la prima è la già accennata volontà di indipendenza dal protocollo utilizzato, la seconda nasce dalla necessità di gestire applicativamente le informazioni di routing senza dover intervenire e modificare impostazioni all'interno dello stack TCP/IP, aprendo un nuovo scenario e nuove potenzialità alle applicazioni e limitando al massimo l'impatto sull'infrastruttura già esistente.

RSA BSAFE Secure-WS

Il 29 Aprile 2003 RSA Security ha annunciato la disponibilità del nuovo software BSAFE Secure-WS, una delle prime soluzioni a implementare gli standard di sicurezza per i web service basati sulle specifiche WS-Security (Web Services Security) supportate da OASIS (Organization for the Advancement of Structured Information Standards) e volte a integrare numerosi modelli e tecnologie di sicurezza per consentire a più sistemi di interagire in modo indipendente rispetto alla piattaforma e al linguaggio utilizzato. BSAFE Secure-WS è progettato per consentire agli sviluppatori di implementare funzioni di sicurezza basate sugli standard per l'abilitazione dei web service.

Lo scopo di RSA Security è offrire il proprio contributo al perfezionamento degli standard e fornire i componenti di sicurezza e il middleware necessari per favorire l'adozione dei web service quali strumenti per l'estensione delle infrastrutture di identity management.

RSA BSAFE Secure-WS è una soluzione software ad alte prestazioni indirizzata a chi desidera implementare funzioni di sicurezza dedicate ai web service, che consente di crittografare/decrittografare, firmare e verificare i messaggi SOAP secondo le specifiche WS-Security. BSAFE Secure-WS è in grado di accettare e validare i principali metodi di autenticazione degli utenti all'interno dell'Identity Management System di RSA Security (annunciato a metà aprile alla RSA Conference): da username/password ai certificati X.509. Se necessario, il software può inoltre essere esteso per supportare altri metodi di autenticazione quali i token per l'autenticazione a due fattori RSA SecurID, i ticket Kerberos e SAML. BSAFE Secure-WS è basato sugli standard e garantisce pertanto l'interoperabilità tra i web service implementati da un'organizzazione e quelli dei suoi partner, clienti e terze parti.

La sicurezza: visione generale

La sicurezza è uno degli aspetti chiave da risolvere per favorire l’adozione dei Web Services. Tutto è legato al fatto che SOAP, uno dei protocolli chiave dei Web Setivces, nonoffre abbastanza garanzie di security, ma senza forme adeguate di protezione della riservatezza delle transazioni, i Web Services non possono essere una piattaforma adatta per realizzare sistemi commerciali business-to-businness e rimangono, come oggi tipicamente accade, dentro le mura sicure delle aziende. In un certo senso, per alcuni analisti, si sta verificando nel campo dei Web Services ciò che è già accaduto nel più classico ambito dell’EDI: realizzare comunicazioni commerciali sicure tra due pater adottando approcci standard è concettualmente semplice, ma difficile da implementare nella realtà.

Proprio per non minare alla base lo sviluppo diìei Web Services, si stà facendo molto per affrontare il problema affiancando a SOAP altri protocolli che ne aumentino la sicurezza, ed è in questo senso che si stanno muovendo tre organismi: IETF (Intenet Engineering Task Force), Oasis (Organization for the Advancement of Structure Information Standard) e W3C (World Wide Web Consortium).

Messi insieme, questi organismi hanno delineato tredici protocolli, che dovrebbero potenziare e rafforzare i Web Services e che spaziano dalla cifratura alle firme digitali, dal non ripudio a “vecchie conoscenze” come Kerberos.

Oasis e W3C stanno in parte cooperando per armonizzare i propri sforzi: la collaborazione è iniziata con WS-Security. Tale protocollo permette ai Web Services di scambiarsi messaggi SOAP sicuri e firmati digitalmente ed è basato su due protocolli del W3C: XML Encryption e XML Digital Signature. Uno degli obiettivi che ci si è posti con la creazione di WS-Security è aggiungere nuove funzioni a SOAP, come la gestione delle policy di sicurezza e di autorizzazione.

Da parte sua il W3C sta sviluppando autonomamente lo standard XKMS e, più in generale, un'architetture per i Web Services che comprenda un framework per la sicurezza.

Oasis segue invece una strada che comprende SAML per l'autenticazione e l'autorizzazione, XACML, SPML e XCBF. L'operato dell'IETF si concentra infine sugli standard per la sicurezza a livello di trasporto, sul quale Oasis e il W3C stanno realizzando i protocolli per garantire la sicurezza a livello applicativo.

Tutti insieme i tre organismi devono raggiungere un medesimo obiettivo: delineare un modello concettuale di un vero e proprio stack della security per i Web Services, in cui si spieghi come i vari protocolli sinora sviluppati collaborano (e non competono) tra loro.

Nel Dicembre scorso un gruppo di aziende (BEA System, Microsoft, IBM, RSA Security, SAP e VeriSIgn) hanno presentato alcune specifiche indirizzate alla sicurezza e alla gestione del flusso operativo dei Web Services. La base tecnologica è sempre SOAP,a cui si affiancano sei nuove proposte divise in due gruppi: il primo dedicato alla sicurezza vera e propria e il secondo alla implementazione di procedure di gestione dei processi in un ambiente Web Services.

Alla prima serie appartengono WS-Trust, WS-SecureConversation e WS-SecurityPolicy; alla seconda WS-Policy, WS-PolicyAttachment e WS-PolicyAssertions.

IBM e Microsoft hanno lavorato alla definizione di tutti i protocolli, RSA Secutity e Verisign hanno collaborato per le defnizioni di quelli dedicati specificamente alla sicurezza, BEA e SAP per la creazione degli altri.

Un modello del genere è necessario, ma c'è anche da dire che oggi suscita un interesse relativo tra gli utenti che hanno già adottato i Web Services e pensano di estenderli oltre il firewall già nel breve termine. La strada seguita da tali utenti comporta naturalmente approcci proprietari sviluppati in casa, magari adottando allo scopo altri standard del mondo Internet o combinando fra loro vari "pezzi" di software di diversi fornitori. Una volta definito chiaramente il panorama degli standard ufficiali, bisognerà vedere quanto di queste soluzioni potrà essere utilizzato così comè e quanto andrà, invece, modificato nuovamente per essere compatibile e interoperabile con il "resto del mondo".

WS-Security sembra essere un buon punto di partenza accettato da tutti: specifica come rendere sicura la comunicazione tra Web Services e descrive come utilizzare protocolli di sicurezza all'interno di SOAP. L'approccio seguito è quello di inserire informazioni legate alla sicurezza negli header dei messaggi SOAP utilizzando XML Encryption, XML Digital Segnature e altri protocolli di identità come SALM o il "vecchio" standard IETF Kerberos.

Capitolo 9 – WSDK

Con Web service Development Kit (WSDK) Technology Preview, Microsoft fornisce il suo primo strumento per l'implementazione della protezione all'interno di un messaggio SOAP. I servizi Web non sono più costretti ad utilizzare le funzionalità di protezione del trasporto sottostante. Adesso il messaggio SOAP stesso può essere autenticato, la sua integrità verificata e può anche venire crittografato all'interno dell'envelope SOAP utilizzando il meccanismo definito nella specifica WS-Security. Ora vediamo come utilizzare WSDK per usufruire di WS-Security nell'autenticazione e nella firma digitale dei servizi Web.

L'ambiente WSDK

WSDK si colloca al vertice del supporto Microsoft .NET Framework nella campo della scrittura e dell'utilizzo di servizi Web. Al centro del supporto WSDK risiede la classe *Microsoft.WSDK.SoapContext* che fornisce un'interfaccia per il controllo dell'intestazione WS-Security e di altre intestazioni per i messaggi SOAP in arrivo, aggiungendo WS-Security ed altre intestazioni ai messaggi SOAP in uscita. E' stata creata una classe wrapper che migliora le funzionalità di Framework aggiungendo un SoapContext per le richieste e le risposte SOAP. Sul server, è stato creato un HTTPModule Microsoft ASP.NET che convalida i messaggi SOAP in entrata e crea una richiesta ed una risposta SoapContext accessibili dall'interno dei metodi Web.

L'impostazione dell'ambiente WSDK di base implica la configurazione dell'applicazione ASP.NET per utilizzare HTTPModule di WSDK. È possibile effettuare l'impostazione per l'intero computer aggiungendo una voce al file Machine.config. Per cominciare a sperimentare con WSDK, la aggiungiamo solo ad una particolare directory virtuale. È sufficiente aggiungere un elemento HTTPModules all'interno dell'elemento system.web del file Web.config per la directory virtuale. L'elemento aggiunto dovrà essere del tipo della voce sottostante con la differenza che l'attributo type dovrà essere disposto su di una stessa linea (qui è messo su più linee per maggiore chiarezza).

```
<httpModules>
  <add name="WSDK" type="Microsoft.WSDK.HttpModule, Microsoft.WSDK,
    Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
</httpModules>
```

Dal punto di vista di WSDK, tutto è pronto per iniziare. Ulteriori impostazioni di configurazione da effettuare per alcuni aspetti di WS-Security verranno discusse al momento opportuno. A questo punto, tutto quello che resta da fare è aggiungere un riferimento alla Microsoft.WSDK.dll nel progetto Microsoft Visual Studio .NET.

Provider di password

Abbiamo detto in precedenza, riguardo alle specifiche di WS-Security nell'ambito di scambio di messaggi SOAP, che sia il mittente che il destinatario devono conoscere la password dell'utente; Dal lato del client, ci si aspetta che agli utenti venga richiesta la loro password. Dal lato del server, tuttavia, c'è la necessità di una tabella che contenga le coppie utenti e password valide. WSDK dispone di questa funzionalità attraverso un meccanismo di estensione chiamato *provider di password.*

WSDK ha definito un'interfaccia, Microsoft.WSDK.Security.IPasswordProvider, che può essere implementata da una classe in modo che possa registrarsi come provider di password. L'interfaccia dispone di una funzione GetPassword che prende un nome utente come input. La funzione GetPassword restituisce la password per l'utente dato. L'idea è che si possa utilizzare qualsiasi meccanismo per memorizzare le coppie nome utente/password valide e quindi fornire una classe che implementi l'interfaccia IPasswordProvider per consentire a WSDK di accedere al proprio particolare meccanismo di memorizzazione della password.

È necessario eseguire un'impostazione di configurazione WSDK appropriata per informare WSDK del proprio provider di password. Questo comporta l'aggiunta di un elemento microsoft.wsdk all'interno dell'elemento di configurazione per il file di configurazione della propria applicazione. Per un'applicazione ASP.NET, nel file Web.config si può aggiungere la voce seguente come figlio dell'elemento di configurazione padre:

```
<microsoft.wsdk>
  <security>
    <passwordProvider
      type="WSDK_Security.PasswordProvider, WSDK-Security" />
  </security>
</microsoft.wsdk>
```

L'attributo type dell'elemento passwordProvider indica la classe scritta per implementare l'interfaccia IPasswordProvider. In questo caso, la classe è chiamata PasswordProvider, è definita nello spazio dei nomi WSDK_Security e si trova nell'assembly WSDK-Security.dll. Per semplicità, la classe restituisce una password "opensezme" per ciascun nome utente. Nella maggior parte dei casi, qui si implementa la logica che legge la password appropriata da un database SQL o da un altro archivio. Di seguito viene riportato il codice per questa classe semplice:

```
namespace WSDK_Security
{
   public class PasswordProvider : Microsoft.WSDK.Security.IPasswordProvider
   {
      public string GetPassword(string username)
      {
            return "opensezme";
      }
   }
```

```
}
```

Creazione di un metodo Web che utilizza WS-Security

Passiamo ora alla creazione del servizio Web. Si tratta di un semplice servizio Web del tipo "Hello world", ma si potrà accedere al SoapContext della classe in modo da poterne personalizzare la risposta. Di seguito viene riportato il codice relativo al metodo Web.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using Microsoft.WSDK.Security;
using Microsoft.WSDK;
namespace WSDK_Security
{
   public class Service1 : System.Web.Services.WebService
   {
      [WebMethod]
      public string PersonalHello()
      {
         string response = "";
         SoapContext requestContext = HttpSoapContext.RequestContext;
         if (requestContext == null)
         {
            throw new ApplicationException("Richiesta non SOAP.");
         }
         foreach (SecurityToken tok in requestContext.Security.Tokens)
         {
            if (tok is UsernameToken)
            {
               response += "Ciao" + ((UsernameToken)tok).Username;
            }
         }
         return response;
      }
   }
}
```

Prima di poter utilizzare WSDK nel metodo Web, è stato necessario aggiungere al progetto un riferimento alla Microsoft.WSDK.dll. Si sono aggiunte al codice due clausole "using" per Microsoft.WSDK e Microsoft.WSDK.Security. È quindi stato possibile creare il metodo Web PersonalHello.

La prima azione della funzione PersonalHello consiste nel recuperare l'oggetto SoapContext per la richiesta utilizzando il membro statico HttpSoapContext.RequestContext. Si può ora accedere a tutte le informazioni delle intestazioni WS-Security. In mancanza di contesto, il metodo Web restituisce un errore, altrimenti è possibile enumerare tutti i token di protezione inviati con la richiesta e verificare che si tratta di UsernameTokens. Se si trova un UsernameToken, viene creata una stringa di risposta con un saluto che si basa sul nome utente.

Creazione di un client che utilizza WS-Security

Per capire come utilizzare WS-Security con WSDK dal lato client, è stata creata un'applicazione Windows Form per chiamare il servizio Web alla selezione di un pulsante. Così come è stato fatto per la classe del servizio Web, è stato aggiunto un riferimento alla Microsoft.WSDK.dll e sono state incluse le stesse istruzioni "using" del codice relativo al server.

Per ciò che riguarda il client, WSDK fornisce una classe Microsoft.WSDK.WSDKClientProtocol che eredita dalla classe System.Web.Services.Protocols.SoapHttpClientProtocol. La classe SoapHttpClientProtocol viene utilizzata quando si seleziona l'opzione Add Web Reference all'interno di Visual Studio .NET, oppure quando si utilizza l'utilità WSDL.exe per creare il codice del client basato su un WSDL. Per generare le classi proxy per il client, è possibile utilizzare sia l'opzione Add Web Reference di Visual Studio .NET che l'utilità WSDL.exe, quindi cambiare la classe proxy generata per ereditarietà da SoapHttpClientProtocol in WSDKClientProtocol. La classe proxy dispone ora delle proprietà RequestSoapContext e ResponseSoapContext, da utilizzare per accedere alle intestazioni WS-Security da inviare e ricevere. Se si utilizza l'opzione Add Web Reference, il codice per la classe proxy generata si troverà nella directory Web References. Nei prog etti C#, un file chiamato Reference.cs si trova nella directory il cui nome corrisponde al nome host dove risiede WSDL. Nei progetti Microsoft Visual Basic® .NET, il file si chiama Reference.vb. Cambiamo ora la dichiarazione di classe da:

```
public class Service1 :
    System.Web.Services.Protocols.SoapHttpClientProtocol {
a:
public class Service1 : Microsoft.WSDK.WSDKClientProtocol {
```

Se si utilizza l'opzione di Visual Studio .NET Add Web Reference, occorre fare attenzione nell'apportare modifiche al file che contiene il codice generato. Se si sceglie l'opzione Update Web Reference, Visual Studio .NET genererà il codice e sovrascriverà le modifiche che sono state apportate.

Per poter creare un UsernameToken per la richiesta, il codice del client sarà di questo tipo:

```
localhost.Service1 proxy = new localhost.Service1();
UsernameToken clearTextToken
    = new UsernameToken("Joe",
                        "opensezme",
                        PasswordOption.SendHashed);
proxy.RequestSoapContext.Security.Tokens.Add(clearTextToken);
label1.Text = proxy.PersonalHello();
```

L'unica differenza tra questo ed un normale codice per chiamare un normale servizio Web consiste nel dover creare un oggetto UsernameToken e aggiungerlo all'insieme Tokens per la richiesta. Il costruttore per l'oggetto UsernameToken prende tre parametri: il nome utente, la password e PasswordOption. In questo caso si invia la password hash. Di seguito viene mostrata una versione

abbreviata della richiesta SOAP generata da questo codice. Si potrà notare che un'intestazione Security è inclusa con la richiesta che include a sua volta un elemento figlio UsernameToken contenente il nome utente, la password hash, un parametro nonce casuale da utilizzare per identificare la particolare richiesta e la data di creazione.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
      ...
    <wsse:Security soap:mustUnderstand="1"
        xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext">
      <wsse:UsernameToken
          wsu:Id="Id-24a67c72-c412-46f4-85b1-dd99c66190d2">
        <wsse:Username>Joe</wsse:Username>
        <wsse:Password Type="wsse:PasswordDigest">
          TPd+eP29FfJLbHpUnAyKGLiTa10=
        </wsse:Password>
        <wsse:Nonce>FHi7zbHpTmfsk/1f2xV3ow==</wsse:Nonce>
        <wsu:Created
            xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
          2002-08-14T17:33:27Z
        </wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <PersonalHello xmlns="http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>
```

Il messaggio può essere ora inviato al servizio Web dove HttpModule di WSDK potrà convalidare il formato generale della richiesta, confrontare la password hash con la password recuperata dal provider di password, e quindi, se tutto andrà bene, la richiesta arriverà al metodo Web. Il metodo Web riceve la richiesta in entrata con SoapContext compilato, cerca un UsernameToken nell'intestazione Security e crea una stringa di risposta basata sul nome indicato. La risposta torna indietro come una normale risposta del metodo Web e l'applicazione client visualizza la stringa restituita. L'applicazione WS-Security è stata eseguita.

Convalida di UsernameToken sul server

Sebbene WSDK verifichi la sintassi dell'intestazione Security e confronti la password hash con quella del provider di password, sarà necessario eseguire ulteriori verifiche sulla risposta. Ad esempio, se si riceve un UsernameToken che non include un elemento password, WSDK non eseguirà una chiamata al provider di password. Se non c'è una password da controllare, non c'è motivo di chiamare il provider di password. Occorre quindi verificare indipendentemente il formato di UsernameToken.

Un'altra possibilità e che ci sia più di un elemento UsernameToken incluso con la richiesta. WS-Security fornisce un supporto per includere un qualsiasi numero di token in una richiesta utilizzata

a scopi diversi. Il fatto che il codice del client abbia aggiunto un UsernameToken all'insieme dei Tokens indica che è possibile includere più token in un solo messaggio. Infatti, se si modifica il codice del client per creare un secondo UsernameToken da aggiungere all'insieme, il codice corrente restituirà una stringa simile alla seguente:

```
Ciao Bob.Ciao Alice.
```

Si deve quindi modificare il codice del metodo Web per verificare che UsernameToken include una password hash e per accettare solo richieste in arrivo con un solo UsernameToken. Il codice modificato è riportato di seguito.

```
[WebMethod]
public string VerifiedPersonalHello()
{
   SoapContext requestContext = HttpSoapContext.RequestContext;
   if (requestContext == null)
   {
      throw new ApplicationException("Richiesta non SOAP.");
   }
   if (requestContext.Security.Tokens.Count == 1)
   {
      foreach (SecurityToken tok in requestContext.Security.Tokens)
      {
         if (tok is UsernameToken)
         {
            UsernameToken UserToken = (UsernameToken)tok;
            if (UserToken.PasswordOption
               == PasswordOption.SendHashed)
            {
               return "Ciao " + UserToken.Username;
            }
            else
            {
               throw new SoapException(
                   "Tipo di password UsernameToken non valido.",
                   SoapException.ClientFaultCode);
            }
         }
         else
         {
            throw new SoapException(
               "È necessario un token di protezione UsernameToken.",
               SoapException.ClientFaultCode);
         }
      }
   }
   else
   {
      throw new SoapException(
          "La richiesta deve contenere uno e un solo token di protezione.",
          SoapException.ClientFaultCode);
   }
    return null;
}
```

Come menzionato in precedenza, esistono altri problemi legati all'improprio utilizzo dell'elemento UsernameToken da parte di un altro utente. Per proteggersi dagli attacchi, sarebbe necessario del codice in aggiunta a quello mostrato nell'ambito di questo articolo. Ci occuperemo invece di un'opzione più sicura.

Invio di certificati X.509 come token WS-Security

Oltre a UsernameTokens, WS-Security definisce un elemento BinarySecurityToken per contenere un paio di tipi di token ben noti con i propri formati proprietari. La specifica definisce i tipi BinarySecurityToken per certificati X.509 v3 e ticket Kerberos v5. WSDK Technology Preview supporta i certificati X.509 che vengono trattati come UsernameToken.

Archivi certificati

Per poter includere un certificato con la propria richiesta, occorre stabilire dove trovarlo. Nel tradizionale sviluppo del software di Microsoft® Windows, i certificati sono contenuti in una posizione centrale chiamata archivio certificati. Ciascun utente dispone di un archivio di certificati privato da utilizzare per eseguire la firma digitale di messaggi di posta elettronica o per fornire autenticazioni utente con server Web su SSL. I certificati stessi non sono privati poiché costituiscono un tipo di firma digitale per la gestione di chiavi pubbliche. Ciò che importa è che la chiave privata corrispondente alla chiave pubblica del certificato costituisca un archivio di chiavi sicuro per l'utente di cui si utilizza il certificato. Ciò consente all'utente di firmare digitalmente un'entità con una chiave privata mentre un altro utente potrà verificare la firma con una chiave pubblica nel certificato.

La versione iniziale di .NET Framework non possedeva classi per accedere agli archivi certificati Windows, sebbene avesse una classe per gestire i certificati X.509. Tuttavia, WSDK fornisce classi che consentono di aprire gli archivi certificati Windows e utilizzare i certificati da questa posizione facilmente gestibile. Il codice seguente utilizza la classe Microsoft.WSDK.Security.Cryptography.X509Certificates.X509CertificateStore per riempire una casella di riepilogo con i nomi comuni di tutti i certificati dell'archivio certificati personale di un utente.

```
  ...
using Microsoft.WSDK.Security.Cryptography;
using Microsoft.WSDK.Security.Cryptography.X509Certificates;
  ...
private X509CertificateStore store;
  ...
private void Form1_Load(object sender, System.EventArgs e)
{
    store = X509CertificateStore.CurrentUserStore(
        X509CertificateStore.MyStore);
    store.OpenRead();
    foreach(X509Certificate cert in store.Certificates)
    {
        listBox1.Items.Add(cert.GetName());
    }
}
```

Per un esempio di una finestra di dialogo completa per la selezione di un certificato X.509 da un archivio certificati, vedere "X509CertificateStoreDialog.cs" nella directory del codice di esempio di WSDK Technology Preview.

Aggiunta del token del certificato X.509 ad un messaggio SOAP

Una volta trovati i propri certificati personali, aggiungerli ad una richiesta SOAP è come aggiungere UsernameToken utilizzato in precedenza. Il codice seguente ottiene il certificato dall'archivio certificati già aperto basandosi sulla selezione della casella di riepilogo che è stata riempita in precedenza, quindi aggiunge all'insieme Tokens il token binario creato dal certificato.

```
X509Certificate cert =
    (X509Certificate)store.Certificates[listBox1.SelectedIndex];
proxy.RequestSoapContext.Security.Tokens.Add(
    new X509SecurityToken(cert));
```

Accedere al certificato dal server è anche molto simile ad accedere a UsernameToken. Di seguito viene mostrata la versione modificata della logica del metodo Web precedente che utilizza il nome comune del certificato per costruire una risposta personalizzata.

```
[WebMethod]
public string PersonalHello()
{
    SoapContext requestContext = HttpSoapContext.RequestContext;
    if (requestContext == null)
    {
        throw new ApplicationException("Richiesta non SOAP.");
    }
    if (requestContext.Security.Tokens.Count == 1)
    {
        foreach (SecurityToken tok in requestContext.Security.Tokens)
        {
            if (tok is Microsoft.WSDK.Security.X509SecurityToken)
            {
                X509SecurityToken certToken = (X509SecurityToken)tok;
                return "Ciao " + certToken.Certificate.GetName();
            }
            else
            {
                throw new SoapException(
                    "È necessario un token di protezione X509.",
                    SoapException.ClientFaultCode);
            }
        }
    }
    else
    {
        throw new SoapException(
            "La richiesta deve contenere uno e un solo token di protezione.",
            SoapException.ClientFaultCode);
    }
    return null;
}
```

Firme digitali

L'invio di certificati X.509 con una richiesta non rappresenta un buon metodo di autenticazione. I certificati sono considerati di dominio pubblico cosicché chiunque può includere alla propria

richiesta il certificato di un altro. Il meccanismo di utilizzo dei certificati per l'autenticazione si basa sull'idea riportata in precedenza di firmare una qualche entità con la chiave privata corrispondente alla chiave pubblica del certificato. Nel caso di invio di un messaggio SOAP, se Bob crea una firma digitale per l'elemento SOAP body con la sua chiave privata, potrà includere il certificato corrispondente insieme alla firma nelle intestazioni della richiesta, cosicché chi riceve il messaggio potrà verificare che la richiesta è stata effettivamente mandata da Bob e non è stata alterata dal momento in cui è stata apposta la firma. Non è necessario includere il certificato con la richiesta, ma ciò risulta conveniente a chi deve convalidare la firma.

WSDK supporta direttamente la creazione di firme digitali. Oltre all'insieme Tokens c'è un insieme SoapContext.Security.Elements che consente di aggiungere vari elementi WS-Security, compreso un elemento Signature. Attraverso il codice del client precedente che comprendeva un certificato digitale, è possibile utilizzare lo stesso certificato per firmare la richiesta. Il codice per eseguire ciò è mostrato di seguito.

```
X509Certificate cert =
    (X509Certificate)store.Certificates[listBox1.SelectedIndex];
X509SecurityToken certToken = new X509SecurityToken(cert);
proxy.RequestSoapContext.Security.Tokens.Add(certToken);
proxy.RequestSoapContext.Security.Elements.Add(
    new Signature(certToken));
```

Proprio come in precedenza, si recupera il certificato dall'archivio certificati precedentemente aperto e lo si aggiunge all'insieme Tokens. Si crea quindi un nuovo oggetto Signature utilizzando X509SecurityToken come un parametro del costruttore Signature e si aggiunge l'oggetto Signature all'insieme Elements. Di seguito viene mostrata una versione ridotta del messaggio SOAP generato dal codice.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <wsrp:path
        soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
        soap:mustUnderstand="1"
        xmlns:wsrp="http://schemas.xmlsoap.org/rp">
      <wsrp:action ...
          wsu:Id="Id-729b19f8-23f2-4688-b3ae-f554810d3b91">
                ...
    </wsrp:path>
          ...
    <wsse:Security soap:mustUnderstand="1"
        xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext">
      <wsse:BinarySecurityToken ValueType="wsse:X509v3"
          EncodingType="wsse:Base64Binary"
          wsu:Id="Id-17c380b6-3cd6-4a1d-8a14-2daf1f0be8ae">
        MIIGkzCCBXugAwIBAgI...    ...A39Vmjd20Lw==
      </wsse:BinarySecurityToken>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <CanonicalizationMethod
```

```
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
            <Reference
                URI="#Id-0cd702d0-ccce-4fe6-a694-e9ada83d0a90">
                    ...
            </Reference>
            <Reference
                URI="#Id-729b19f8-23f2-4688-b3ae-f554810d3b91">
                    ...
          </SignedInfo>
          <SignatureValue>
            O6AOmI9IWkouOvPwCRNwH...       ...kcoWQ8OvrwvSQgQ=
          </SignatureValue>
          <KeyInfo>
            <wsse:SecurityTokenReference>
              <wsse:Reference
        URI="#SecurityToken-17c380b6-3cd6-4a1d-8a14-2daf1f0be8ae" />
            </wsse:SecurityTokenReference>
          </KeyInfo>
        </Signature>
      </wsse:Security>
    </soap:Header>
    <soap:Body
        xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
        wsu:Id="Id-0cd702d0-ccce-4fe6-a694-e9ada83d0a90">
      <PersonalHello xmlns="http://tempuri.org/" />
    </soap:Body>
</soap:Envelope>
```

Per una maggiore comprensione dei vari elementi di intestazione inclusi nella richiesta, si consiglia la consultazione delle specifiche WS-Routing e Web service Security Addendum . Ai fini di questo articolo, ci limiteremo ad esaminare l'utilizzo di WS-Security da parte di WSDK per la firma delle richieste

Innanzitutto si può notare un elemento BinarySecurityToken all'interno dell'intestazione Security. I relativi attributi indicano che si tratta di un certificato X.509 v3 codificato in base 64. È il X509SecurityToken aggiunto all'insieme Tokens nei due esempi precedenti. Non è necessario includerlo nel messaggio, ma semplifica il processo di convalida della firma.

Si può notare quindi l'esistenza di un elemento Signature nell'intestazione Security. Vi sono tre elementi figlio per Signature: l'elemento SignedInfo, l'elemento SignatureValue e l'elemento KeyInfo. L'elemento SignedInfo definisce la data esatta in cui la richiesta è stata firmata e quale algoritmo è stato utilizzato per elaborare la firma. Ciò che è importante è l'elenco degli elementi di riferimento che definiscono cosa è stato firmato nel messaggio. Anche se per brevità sono stati rimossi alcuni elementi di riferimento, si noterà che ai due elementi elencati è associato un attributo URI. L'attributo URI corrisponde all'attributo ID dell'elemento incluso nella firma. Ad esempio, il primo riferimento indica un URI che inizia con "#Id-0cd702d0-". Cercando all'interno del messaggio, si scoprirà che questo URI corrisponde all'ID per l'elemento Body all'interno del messaggio. Di conseguenza, Body fa parte dell'informazione firmata relativa alla richiesta. Il s econdo riferimento è relativo all'elemento action dell'intestazione Path che fa parte della specifica WS-Routing. WSDK include automaticamente le intestazioni Path e Timestamp e la maggior parte

delle informazioni di queste intestazioni è inclusa nella firma digitale. Nell'elenco seguente sono inclusi tutti gli elementi a cui WSDK fa riferimento nell'intestazione SignedInfo, che verranno quindi inclusi nella firma digitale.

```
/soap:Envelope/soap:Header/wsrp:path/wsrp:action
/soap:Envelope/soap:Header/wsrp:path/wsrp:to
/soap:Envelope/soap:Header/wsrp:path/wsrp:from
/soap:Envelope/soap:Header/wsrp:path/wsrp:id
/soap:Envelope/soap:Header/wsu:Timestamp/wsu:Created
/soap:Envelope/soap:Header/wsu:Timestamp/wsu:Expires
/soap:Envelope/soap:Body
```

Piuttosto che gli elementi completi path e Timestamp, ad essere firmati sono i sottoelementi delle intestazioni path e Timestamp, dal momento che, in scenari WS-Routing, ci si aspetta che degli intermediari aggiungano elementi alle intestazioni path e Timestamp non appena il messaggio si propaga ai router di livello SOAP. Di conseguenza, se qualcosa nelle intestazioni dovesse venire modificato, la firma digitale si interromperebbe.

Infine si può notare che, per l'elemento Signature della richiesta, KeyInfo fa riferimento a BinarySecurityToken che contiene il certificato. Questo sta ad indicare la provenienza della chiave utilizzata per firmare la richiesta. Ovviamente, per un certificato, è la chiave privata corrispondente alla chiave pubblica ad essere utilizzata per creare la firma digitale. La chiave pubblica potrà quindi essere utilizzata per verificare che i dati sono stati inviati da un utente a conoscenza della chiave privata corrispondente.

Verifica di una firma digitale

HttpModule di WSDK convaliderà la sintassi di una firma digitale, ma la sola conoscenza che una firma valida esista non è sufficiente ad assicurarsi che il messaggio proviene effettivamente dal particolare utente. Le specifiche WS-Security e XML Digital Signature sono abbastanza flessibili da consentire l'inclusione di firme in XML. Ad esempio, ci si aspetta un messaggio da Alice, si può constatare che il messaggio contiene una firma di Alice, ma tale firma potrebbe in realtà riferirsi solo all'informazione di routing nell'intestazione path. Non è garantito che ciò che ci interessa di più (solitamente il corpo del messaggio) venga effettivamente da Alice a meno che non si facciano dei controlli personali.

Il codice seguente controlla la presenza di una Signature all'interno della richiesta, verifica che il corpo del messaggio sia firmato con la proprietà WSDK IncludesSoapBody, quindi, dal momento che sappiamo esattamente chi ha inviato il messaggio e che è arrivato intatto, crea una risposta di saluto personale basata sul certificato utilizzato per firmare la richiesta.

```
[WebMethod]
public string PersonalHello()
{
```

```
    SoapContext requestContext = HttpSoapContext.RequestContext;
    if (requestContext == null)
    {
        throw new ApplicationException("Richiesta non SOAP.");
    }
    foreach (Object elem in requestContext.Security.Elements)
    {
        if (elem is Signature)
        {
            Signature sign = (Signature)elem;
            // Verifica che il corpo della richiesta contenga la firma
            if (sign.IncludesSoapBody)
            {
                // Determina il tipo del token utilizzato
                // con la firma.
                if (sign.SecurityToken is UsernameToken)
                    return "Ciao " +
                        ((UsernameToken)sign.SecurityToken).Username;
                else
                    if (sign.SecurityToken is X509SecurityToken)
                        return "Ciao " +
                            ((X509SecurityToken)sign.SecurityToken)
                                .Certificate.GetName();
            }
        }
    }
    // Non è stata trovata una firma appropriata
    throw new SoapException("Non è stata trovata una firma valida",
        SoapException.ClientFaultCode);
}
```

Conclusioni

WSDK Technology Preview fornisce una prima panoramica sulle funzionalità della specifica WS-Security. Abbiamo visto come utilizzare WSDK per eseguire autenticazioni **UsernameToken** e X.509 mediante la firma digitale di parti del messaggio SOAP, e come WSDK consente di controllare e verificare il supporto WS-Security che è incluso nel messaggio SOAP. Esistono altre funzionalità di WS-Security fornite da WSDK che non sono state prese in considerazione, come ad esempio l'uso di crittografia o l'inclusione in una risposta a un metodo Web di token, firme e crittografie.

Capitolo 10 – Apache SOAP 2.3.1

Implementazioni SOAP

La specifica SOAP è piuttosto recente, pertanto le implementazioni esistenti sono ancora poche e la maggior parte di esse non è fedele completamente a tutte le regole di specifica. In particolare possiamo elencare le seguenti:

- SOAP for Python: implementazione per il linguaggio Python
- Developmentor SOAP for Perl, Java, C++: implementazioni di Developmentor
- Apache SOAP: implementazione totalmente Java
- TRLSOAP: implementazione totalmente Java curata da IBM
- X-SOAP Toolkit: implementazione totalmente Java curata da IBM
- SOAP for BEA WebLogic: implementazione totalmente Java curata da BEA e dedicata appositamente all'application server WebLogic
- Business Integration Platform (Shinka Tecnologies): implementazione Java, VB, C++
- Microsoft SOAP Toolkit: implementazione VB e C#
- SOAP::Lite: implementazione Perl
- SOAP for ADA: implementazione ADA
- PHP SOAP Toolkit: implementazione PHP
- SOAP for TCL: implementazione TCL

Esistono, in ogni caso, e nascono continuamente altre implementazioni di SOAP per i più svariati linguaggi di programmazione oppure come componenti di altri ambienti di sviluppo.

Qui di seguito verranno prese in considerazione soprattutto le due implementazioni, che dal punto di vista della tecnologia e della distribuzione, possono essere considerate le migliori. Si tratta dell'implementazione Java di Apache Group e di quella di VB/C# di Microsoft.

Introduzione ad Apache SOAP 2.3.1

La Apache Software Fondation sta sviluppando da tempo un progetto ambizioso, si tratta di xml.apache.org con l'obiettivo di costruire un supporto XML, che sia robusto, completo, qualitativamente evoluto e gratuito, alla più ampia quantità possibile di esigenze implementative. In questo contesto si inserisce Apache SOAP.

La versione 2.3.1 basata sul prodotto IBM SOAP4J implementation è conforme alla versione 1.1 della specifica SOAP e alla SOAP Messages with Attachments specifications in Java, anche se con alcune limitazioni, ed è, come tutti i progetti della ASF, open source e disponibile attraverso la Apache License.

Grazie alla passione di questo gruppo di lavoro, peraltro aperto alla collaborazione di chiunque, sono nati moltissimi prodotti indipendenti, sempre dotati di XML come piattaforma di base, di ausilio nello sviluppo di applicazioni. Si va dal package java per la lettura e scrittura di documenti XML, al server web Apache, al servlet engine ed application server Tomcat, fino al package SOAP.

Apache SOAP 2.3.1 è un package Java che può essere installato sia client che sever side e consente da una parte lo sviluppo di applicazioni che intendono utilizzare SOAP come protocollo applicativo, dall'altro il supporto lato server alle componenti remote di tali applicazioni.
L'installazione client-side prevede che sia reso disponibile al J2SDK ed alla Java Virtual Machine il package SOAP contenente tutta l'implementazione della chiamata RPC via SOAP.

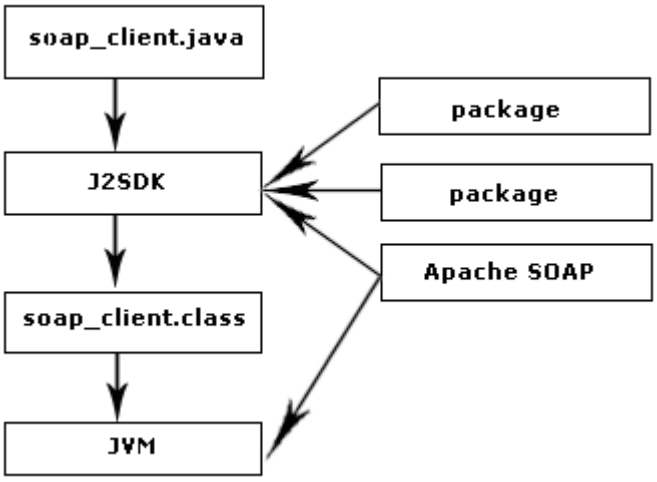


Figura 10.1: Applicazione SOAP lato client

L'installazione server-side prevede che il package SOAP sia reso disponibile al JRE o alla JVM di pertinenza, nel caso specifico alla JVM del server Tomcat.

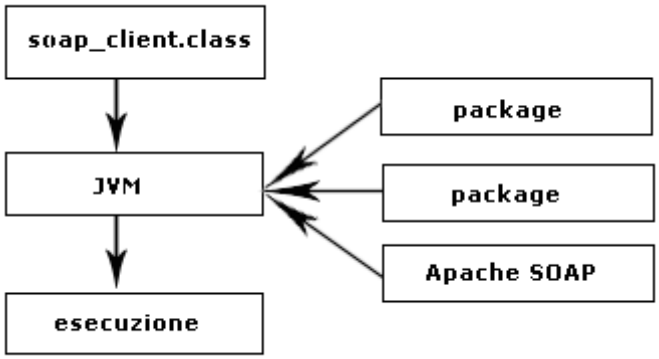


Figura 10.2: Applicazione SOAP lato server

Requisiti

In ognuno dei due casi, vi sono alcuni requisiti che devono essere soddisfatti prima di poter iniziare l'installazione di Apache SOAP. Un'installazione Client di Apache SOAP consiste nell'arricchire un ambiente di sviluppo Java con il package soap.jar, il cuore dell'implementazione. Un'installazione Server, invece, consiste nel rendere disponibile ad un application server, la componentistica giusta per permettergli di rispondere a chiamate SOAP, nel caso specifico si tratta dello stesso package soap.jar. Il requisito minimo, trattandosi in ogni caso di applicazioni scritte in Java, è la presenza, sulla macchina che si intende utilizzare per lo sviluppo, del Java2 Software Development Kit.

Le altre componenti essenziali sono:

- Il package mail.jar del pacchetto JavaMail di Sun
- Il package activation.jar del pacchetto JavaBeans Activation Framework, anch'esso di Sun
- Il package Xerces di Apache per la parificazione dell'XML

Client RPC

I passi basilari per creare un client, il quale interagisca con un servizio SOAP RPC-based, sono i seguenti:

- Ottenere la descrizione dell'interfaccia del servizio SOAP, così da conoscere la struttura del metodo che si desidera invocare. In genere tale descrizione è fornita dal file WSDL associato al servizio.
- Accertarsi che la serializzazione in XML dei parametri inviati e la deserializzazione da XML dei parametri ricevuti in risposta sia registrata/supportata. Apache SOAP fornisce un

numero predefinito di serializzatori/deserializzatori. Per trasmettere o ricevere un tipo di dato non registrato è necessario registrarsi un proprio serializzatore/deserializzatore

- Creare un nuovo oggetto *Call* (org.apache.soap.RPCMessage.Call object). Questo è l'oggetto chiave per una chiamata SOAP, infatti i suoi metodi vengono utilizzati per settare caratteristiche della comunicazione
- Settare, all'interno dell''oggetto *Call,* l'URI del servizio web da utilizzare attraverso il metodo *setTargetObjectURI*
- Settare il metodo che si desidera invocare all'interno dell'oggetto *Call* attraverso il metodo *SetMethodName*
- Creare gli oggetti *Parameter* necessari per la chiamata RPC e settarli all'interno dell'oggetto Call attraverso il metodo *setParams*. Ovviamente i parametri devono essere in numero, ordine e tipo coincidenti con quelli richiesti dal servizio. Inoltre il serializzatore/deserializzatore per gli oggetti da trasmettere/ricevere deve essere registrato
- Eseguire sull'oggetto *Call* il metodo *invoke* e catturare, con un oggetto *Response*, l'oggetto ritornato dal metodo *invoke*. Il metodo *invoke* prevede due parametri: il primo è l'URL che identifica l'endpoint sul quale il servizio risponde, il secondo è il valore che sarà posizionato nell'header SOAPAction (presente in un messaggio http).
- Verificare se l'oggetto *Response* contiene un errore attraverso il metodo *generatedFault.* Tale metodo ritorna un valore false se non vi sono stati errori: true altrimenti. In caso d'errore attraverso il metodo *getFault*, che ritorna un oggetto *Fault*, è possibile indagare sul tipo di errore.
- Nel caso non vi siano errori, è possibile estrarre il risultato (i parametri) di ritorno dall'oggetto *Respons*e attraverso il metodo *getReturnValue* (*getParams*).

Server RPC

La pubblicazione di un servizio RPC si articola di due passi:

- Creare il codice che descrive il servizio attraverso un linguaggio supportato da Apache SOAP, ad esempio utilizzare Java. Tale sorgente Java non ha nulla di speciale che lo renda in qualche modo assimilabile ad un servizio web utilizzabile in remoto. La potenza di questa implementazione è anche questa, non dover utilizzare codice particolare per definire i servizi web, sarà il cliente a richiamare il servizio utilizzando SOAP.
- La pubblicazione vera e propria richiede dei creare un Apache SOAP deployment descriptor del servizio. Tale deployment descriptor fornisce tutte le informazioni necessarie per maneggiare una richiesta ed offrire il servizio; solitamente si crea un documento XML contenente le specifiche della pubblicazione. Ad esempio, per un implementazione Java, il

file contiene principalmente il nome del servizio, il nome della classe che fornisce il servizio e il nome del metodo che fornisce il servizio.

Per effettuare fisicamente il deploy del servizio si utilizza la seguente istruzione:

```
java org.apache.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter
deploy file_deployment_descriptor.xml
```

Si utilizza la classe Java org.apache.server.ServiceManagerClient che si occupa di registrare i servizi, ad essa si passa come parametri il nome della servlet da utilizzare per effettuare il deploy (cioè la rpcrouter), il comando deploy ed il documento XML contenente il deployment descriptor.

In questo modo si rimanda alle funzionalità specifiche dell'ambiente Apache SOAP il compito di accettare le richieste SOAP provenienti dai client, passare i parametri al metodo destinatario ed, infine, rispondere al cliente con un messaggio SOAP contenente il risultato dell'invocazione del metodo. Quindi la gestione dei messaggi SOAP, con relative specifiche, è del tutto trasparente all'utente che ha scritto il codice del server.

Esempio HelloWorld2

L' obiettivo dell'applicazione HelloWorld2 è avere un servizio web che esporti un metodo chiamato sayHello() che prenda in input un parametro di tipo String e restituisca la stringa "Hello xxx Welcome to SOAP", con xxx valorizzato come il parametro che è stato passato al metodo. Il servizio web sarà poi utilizzato da un'applicazione java e da una pagina JSP in running sull'application server Tomcat.

Il server

Vediamo il codice java sorgente del server:

```
package esempi.helloworld2;

import java.io.*;
import java.util.*;

public class HelloWorldServer2
{
```

```
            public String sayHello(String st)
             {
                return ("Hello  "+st+"  !Welcome to SOAP");
             }
}
```

Dando un'occhiata a questo codice sorgente Java appare chiaro che non ha nulla di speciale, nulla che lo renda in qualche modo assimilabile ad un servizio web utilizzabile in remoto. La potenza di questa implementazione è anche questa, il non dover utilizzare codice particolare per definire servizi web, sarà il client a richiamare il servizio utilizzando SOAP.

Come possiamo notare dal codice, viene gestito un parametro di tipo String che il metodo sayhello() si aspetta in input.

A questo punto il servizio web è stato scritto e compilato, ora è necessario pubblicarlo. Per far questo, creiamo un file XML contenente le specifiche per la pubblicazione. Vediamone il contenuto:

```
<isd:service
  xmlns:isd="http://xml.apache.org/xml-soap/deployment"
  id=”urn:HelloWorldServer2”>
  <isd:provider
   type="java"
   scope="Request"
   methods="sayHello">
    <isd:java class="esempi.helloworld.HelloWorldServer2" static="false"/>
  </isd:provider>
</isd:service>
```

Il file in questione contiene tutte le informazioni che sono necessarie per la pubblicazione, in particolare si tratta delle seguenti:

- ID: *urn:HelloWorldServer2*
- Scope: *Request*
- Provider type: *java*
- Provider class: *esempi.heloworld.HelloWorldServer2*

- Use Static Class: *false*
- Methods: *sayHello*

Come si può notare viene indicato con precisione che si tratta di un servizio remoto di nome HelloWorldServer2 sviluppato in Java ed implementato nella classe *esempi.heloworld.HelloWorldServe2r* ed in particolare utilizza il metodo sayHello.

Per eseguire fisicamente il deploy del servizio dobbiamo eseguire dal prompt comandi, la seguente istruzione:

```
java org.apache.soap.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter deploy
helloworld2_deploy.xml
```

Utilizza vale a dire la classe Java *org.apache.soap.server.ServiceManagerClient*, un client SOAP che si occupa di registrare i servizi, ad essa passa come parametri il nome della servlet da utilizzare per effettuare il deploy, cioè la *rpcrouter*, il comando *deploy* ed il file XML contenente tutti i parametri che abbiamo appena visto.

Il client

Vediamo il codice sorgente del client:

```
package esempi.helloworld2;

import java.net.*;
import java.util.*;

import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class HelloWorld2
{
   public final static void main(String args[])
      throws MalformedURLException, SOAPException
   {
```

```
        Call call = new Call();
        call.setTargetObjectURI("urn:HelloWorldServer2");
        call.setMethodName("sayHello");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

        Vector params = new Vector();
        params.addElement(new Parameter("st",String.class,args[0],null));
        call.setParams(params);
        URL url =  new URL("http://localhost:8080/soap/servlet/rpcrouter");

        Response resp = call.invoke(url,"");

        if(!resp.generatedFault())
        {
           Parameter ret = resp.getReturnValue();
           System.out.println(ret.getValue());
        }
        else
        {
           Fault fault = resp.getFault();
           System.err.println("--------------------------------");
           System.err.println("Attenzione: Condizione di Fault");
           System.err.println("Codice: "+ fault.getFaultCode());
           System.err.println("Descrizione: "+ fault.getFaultString());
           System.err.println("--------------------------------");
        }
    }
}
```

Analizzando il codice del cliente SOAP, appare subito evidente l'utilizzo di due oggetti particolari: Call e Response.

L'oggetto chiave della chiamata SOAP è Call, viene creato attraverso il costruttore:

```
Call call = new Call();
```

e poi vengono usati alcuni suoi metodi per settare le caratteristiche della comunicazione:

```
call.setTargetObjectURI("urn:HelloWorldServer2");
```

imposta il servizio web da utilizzare:

```
call.setMethodName("sayHello");
```

imposta il metodo da utilizzare tra quelli messi a disposizione dal server:

```
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
```

imposta l'EncodingStyle del payload XML.

Prima di poter creare l'oggetto Response utilizzando il classico metodo:

```
Response resp = call.invoke(url,"");
```

è necessario pensare ai parametri, in particolare il parametro di tipo String da passare al metodo sayhello(). L'oggetto da utilizzare come veicolo per trasportare i parametri all'interno della request SOAP è un Vector. Dopo aver creato l'oggetto con l'istruzione:

```
Vector params = new Vector();
```

si aggiungono i parametri necessari utilizzando il metodo addElement, al quale è necessario dare in input un oggetto istanza della classe Parameter, anche creato al volo attraverso un costruttore, come nel caso seguente:

```
params.addElement(new Parameter("st",String.class,args[0],null));
```

Al termine della procedura di aggiunta dei parametri è necessario passare l'oggetto istanza della classe Vector al metodo setParam dell'oggetto call:

```
call.setParams(params);
```

Quello che stiamo cercando di ottenere è una chiamata al metodo remoto sayHello() passandogli come parametro una stringa di testo. Facendo un parallelo con una chiamata da un metodo locale siamo nella situazione seguente:

```
stringa = sayHello("parametro stringa");
```

Nel momento in cui sia necessario chiamare un metodo remoto utilizzando SOAP come protocollo RPC e sia necessario passare dei parametri di elaborazione al metodo remoto, quella che abbiamo visto è la procedura da seguire.

Ora si crea l'oggetto Response attraverso la chiamata al metodo invoke dell'oggetto call, utilizzando cioè il codice seguente:

```
Response resp = call.invoke(url,"");
```

L'oggetto Response contiene tutte le informazioni della response SOAP. Quello che ci si aspetta in condizioni di normalità, è la presenza, all'interno dell'oggetto Response, del risultato dell'elaborazione. Il valore ritornato dall'elaborazione remota è contenuto nell'oggetto ret, istanza della classe Parameter ed ottenuto attraverso la chiamata:

```
Parameter ret = resp.getReturnValue();
```

Nel caso specifico, se vogliamo produrre un output contenente il risultato dell'elaborazione, che sappiamo sarà una stringa di testo, potremmo utilizzare l'istruzione:

```
System.out.println(ret.getValue());
```

che restituisce il valore di ritorno del metodo remoto. In generale però, prima di poter tentare di leggere il valore di ritorno di un metodo remoto è opportuno verificare che non ci sia stato un errore di comunicazione oppure, come abbiamo visto nei primi capitoli, una condizione di Fault. Per questi controlli esiste il metodo:

```
resp.generatedFault();
```

che se restituisce un valore true indica che qualcosa non ha funzionato a livello di client o di server. In questo caso sarà possibile utilizzare l'oggetto Fault restituito dal metodo getFault() dell'oggetto Response. Tutto ciò si realizza attraverso il codice:

```
Fault fault = resp.getFault();
```

L'oggetto Fault, come vedremo, possiede alcuni metodi che chiariscono le dinamiche della comunicazione e descrivono in dettaglio l'errore. In particolare il metodo getFaultCode() restituisce il codice d'errore, mentre getFaultString() restituisce una stringa descrittiva della condizione di errore accorsa. Il codice di esempio, quindi, legge il risultato dall'elaborazione solo se non si sono verificate condizioni di Fault, in caso contrario legge i dati del Fault e li mostra all'utente.

L'esecuzione di questa applicazione prevede il passaggio sulla linea di comando di un parametro che sarà interpretato come una stringa e sarà passato al metodo remoto, pertanto l'esecuzione si ottiene utilizzando il comando:

```
java esempi.helloworld2.HelloWorld2 "Chiara"
```

In questo caso quindi il parametro "lettore" viene passato sulla linea di comando al client Java, il quale lo leggerà attraverso l'istruzione args[0] e potrà usarla per costruire l'oggetto Vector da passare al metodo remoto chiamato attraverso SOAP. Il risultato in output dovrebbe essere il seguente:

```
Hello Chiara!Welcome to SOAP
```

Il traffico TCP/IP

Vediamo cosa accade a livello di call e response SOAP, ovvero cosa veramente viene passato come payload all'interno della chiamata HTTP. Per fare ciò utilizziamo l'utility tcpTrace e spostiamo la porta di ascolto del server dalla 8080 di default alla 8081, in modo che l'applicazione di monitoraggio possa effettuare il tunneling tra client e server in maniera trasparente alle relative implementazioni.

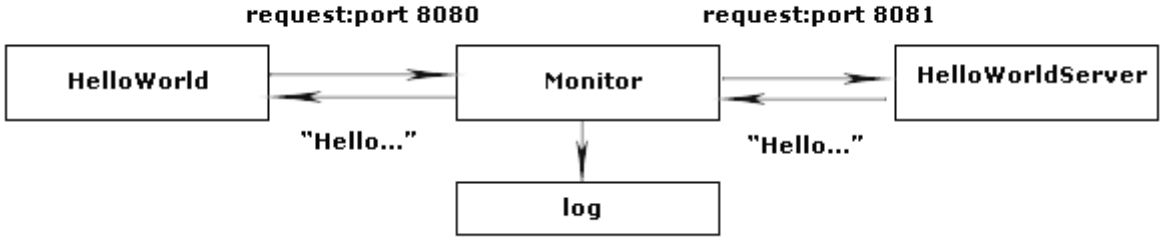


Figura 10.3: Trace della comunicazione HTTP

Non appena avviata l'applicazione server attraverso il comando:

```
java esempi.helloworld2.HelloWorld2 "Chiara"
```

ed ottenuto il messaggio di output che ci si aspettava, possiamo andare a vedere cosa è stato tracciato dall'applicazione di monitoraggio.

Ecco la request SOAP:

```
POST /soap/servlet/rpcrouter HTTP/1.0
```

```
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Lenght: 449
SOAPAction: ""

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:sayHello
       xmlns:ns1="urn:HelloWorldServer2"
       SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <st xsi:type="xsd:string">Chiara</st>
     </ns1:sayHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Come si può vedere si tratta di una chiamata HTTP che utilizza il metodo POST. All'header è stata aggiunta la riga relativa alla SOAPAction che però non contiene valori in quanto non è stata riempita dal codice dell'applicazione client. L'host di destinazione è localhost, come indicato nel codice dell'applicazione client ed il contenuto della chiamata HTTP è un payload XML che segue le specifiche SOAP. Si tratta cioè esattamente di quello che ci si aspettava. Analizzando il payload XML possiamo notare che il servizio remoto di destinazione è la servlet */soap/servlet/rpcrouter* ed in particolare il metodo sayHello che era stato mappato, a livello di deploy dal servizio *urn:HeloWorldServer*, allo stesso metodo implementato nella classe Java *esempi.helloworld2.HelloWorldServer2 "Chiara"*

Come si può notare l'elemento XML <ns1:sayHello> ha come contenuto l'elemento <st xsi:type="xsd:string">Chiara</st>, che serve per trasportare il parametro st="Chiara" di tipo String.

Vediamo ora la response SOAP:

```
HTTP/1.0 200 ok
```

```
Content-Type: text/xml; charset=utf-8
Content-Lenght: 500
Set-Cookie2: JSESSIONID=qqi3ql59v1;Version=1;Discard;Path=”/soap”
Set-Cookie: JSESSIONID=qqi3ql59v1;Path=/soap
Servlet-Engine: Tomcat Web Server/3.2.3 (JSP 1.1; Servlet 2.3; Java 1.3.0;
Windows 2000 5.0 x86; java.vendor=Sun Microsystems Inc.)

<?xml version=’1.0’ encoding=’UTF-8’?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:sayHelloResponse
       xmlns:ns1=”urn:HelloWorldServer2”
       SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <return xsi:type="xsd:string">
        Hello  Chiara ! Welcome to SOAP
       </return>
     </ns1:sayHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Il server http risponde con un codice 200 OK per informare che la comunicazione ha avuto successo e quindi restituisce alcune righe di header. All’interno del payload XML, più precisamente nella sezione Body, è presente l’elemento *<ns1:sayHelloResponse>* che rappresenta il contenitore delle informazioni per la risposta all’elemento *<ns1:sayHello>* presente nella request. Il valore di ritorno vero e proprio è costituito dall’elemento XML:

```
<return xsi:type=”xsd:string”>
    Hello  Chiara ! Welcome to SOAP
</return>
```

Questa response contiene esattamente quello che ci aspettiamo, cioè la presenza, all'interno della stringa, di ritorno, del valore del parametro che era stato passato in input al metodo remoto. Il codice Java dell'applicazione client, in questo caso, non gestisce autonomamente la condizione particolare della mancanza del parametro in input, di conseguenza provando a lanciare l'esempio nel modo seguente:

```
java esempi.helloworld2.HelloWorld2
```

ci troviamo di fronte all'errore :

```
Exception in thread "main" java.lang.ArrayIndexOutBoundsException: 0
at esempi.helloworld2.HelloWorld2.main(HelloWorld2.java:28)
```

Questo utilizzo inappropriato dell'applicazione client non produce una condizione di Fault, ma un vero e proprio errore applicativo, questo perché quando l'eccezione viene prodotta non è ancora iniziata la comunicazione SOAP.

Dal punto di vista dell'implementazione lo sviluppatore ha dovuto soltanto utilizzare, a livello di sviluppo dell'applicazione client, gli oggetti Call, Response e Fault. A tutto il resto ha pensato l'implementazione SOAP che è stata utilizzata.

Un client JSP

Se immaginiamo di avere un client JSP, le cose sono simili a quanto visto per il client scritto in Java. Nel caso specifico utilizzeremo lo stesso application server (Tomcat) sul quale è in running la servlet rpcrouter e di conseguenza lo stesso sul quale girano i servizi.

Se vogliamo scrivere una pagina JSP da utilizzare all'interno di Tomcat dovremo modificare il file di configurazione, in modo che sia possibile utilizzare il path /helloworld2 dell'application server per raggiungere tutti i file contenuti all'interno del percorso fisico indicato dal parametro docBase. La pagina JSP in questione avrà il seguente codice:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=WINDOWS-1252">
<TITLE>
Hello World in SOAP
</TITLE>
</HEAD>
```

```
<BODY>
<h2>Benvenuto in SOAP, come ti chiami?</h2>
<form action="helloworld2b.jsp" name="hello" id="hello">

<input type="text" name="name" size="20" maxlength="20">
<input type="submit" name="submit" value="Invia">

</form>

</BODY>
</HTML>
```

Questa pagina JSP contiene solo codice HTML, si tratta di un form che permette all'utente l'inserimento di un valore di tipo String. Questo valore sarà passato come parametro, utilizzando indifferentemente GET o POST, alla successiva pagina JSP.

(immagine form)

Proviamo ad inserire un valore nel text-box e a premere il pulsante di Submit del Form. A questo punto quello che richiediamo è una GET alla pagina HelloWorld2b.jsp e passandogli sulla QueryString il parametro che abbiamo inserito nel form. Il risultato della chiamata è il seguente:

(immagine form)

Come si può vedere sulla QueryString il parametro "name" è stato passato alla pagina di destinazione e da questa in qualche modo è stato elaborato. Vediamo in che modo, guardando al codice della pagina:

```
<%@ page contentType="text/html;charset=WINDOWS-1252"%>
<%@ page
import="java.util.*,java.io.*,java.net.*,org.w3c.dom.*,org.apache.soap.util.x
ml.*,org.apache.soap.*,org.apache.soap.encoding.*,org.apache.soap.encoding.so
apenc.*,org.apache.soap.rpc.*" %>

<HTML>
<HEAD>
```

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=WINDOWS-1252">
<TITLE>
Hello World in SOAP
</TITLE>
</HEAD>
<BODY>
<%    Call call = new Call();
      call.setTargetObjectURI("urn:HelloWorldServer2");
      call.setMethodName("sayHello");
      call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

      Vector params = new Vector();
      params.addElement(new
Parameter("st",String.class,request.getParameter("name"),null));
      call.setParams(params);
      URL url =  new URL("http://localhost:8080/soap/servlet/rpcrouter");

      Response resp = call.invoke(url,"");

      if(!resp.generatedFault())
      {
         Parameter ret = resp.getReturnValue();
         out.println("<H2>" + ret.getValue() + "</H2>");
      }
      else
      {
         Fault fault = resp.getFault();
         out.println("-------------------------------");
         out.println("Attenzione: Condizione di Fault");
         out.println("Codice: "+ fault.getFaultCode());
         out.println("Descrizione: "+ fault.getFaultString());
```

```
            out.println("--------------------------------");
        }
%>
</BODY>
</HTML>
```

Come si può vedere il parametro “name” che viene passato alla pagina JSP viene letto attraverso l’istruzione request.getParameter(“name”) e viene utilizzato per creare un nuovo parametro da aggiungere al vettore di parametri che verrà poi associato alla chiamata SOAP.

L’implementazione utilizzata nel codice JSP non è per niente diversa da quella di un classico client scritto in Java.

Il traffico TCP/IP

Vediamo la Request HTTP Get sulla porta 8080 per la pagina JSP (helloworld2a.jsp)

```
GET /helloworld2/helloworld2a.jsp HTTP/1.1
Accept: */*
Accept-Language: it
Accept-Encoding: gzip, defilate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: localhost:8080
Connection: Keep-Alive
Cookie: JSESSIONID=6xbahr6rf1
```

Vediamo la Response:

```
HTTP/1.0  200 OK
Content-Type: text/html; charset=ISO-8859-1
Servlet-Engine: Tomcat Web Server/3.2.3 (JSP 1.1; Servlet 2.3; Java 1.3.0;
Windows 2000 5.0 x86; java.vendor=Sun Microsystem Inc.)
<HTML>
<HEAD>
<META http-EQUIV="Content-Type" CONTENT="text/html; charset=WINDOWS-1252">
```

```
<TITLE>
</HEAD>
<BODY>
<h2>Benvenuta in SOAP, come ti chiami?</h2>
<form action="helloworld2b.jsp" name="hello" id="hello">

<input type="text" name="name" size="20" maxlenght="20">
<input type="submit" name="submit" value="Invia">

</form>

</BODY>
</HTML>
```

Request HTTP Get sulla porta 8080 per la pagina JSP (helloworld2b.jsp)

```
GET /helloworld2/helloworld2b.jsp?name=Chiara HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword,
*/*
Refer: htp://localhost:8080/helloworld2/helloworld2a.jsp
Accept-Language: it
Accept-Encoding: gzip, defilate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: localhost:8080
Connection: Keep-Alive
Cookie: JSESSIONID=6xbahr6rf1
```

Request SOAP

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Lenght: 451
```

```
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:sayHello
       xmlns:ns1="urn:HelloWorldServer2"
       SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <st xsi:type="xsd:string">Chiara</st>
     </ns1:sayHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

Response SOAP

```
HTTP/1.0 200 ok
Content-Type: text/xml; charset=utf-8
Content-Lenght: 5020
Set-Cookie2: JSESSIONID=yzl9257t21;Version=1;Discard;Path="/soap"
Set-Cookie: JSESSIONID= yzl9257t21;Path=/soap
Servlet-Engine: Tomcat Web Server/3.2.3 (JSP 1.1; Servlet 2.3; Java 1.3.0);
Windows 2000 5.0 x86; java.vendor=Sun Microsystems Inc.)

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:sayHelloResponse
```

```
      xmlns:ns1="urn:HelloWorldServer2"
      SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">
       Hello  Chiara ! Welcome to SOAP
      </return>
    </ns1:sayHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Dal quadro completo delle Request e Response HTTP e SOAP appare chiaro che la situazione a livello di comunicazione SOAP è identica nel caso in cui il client sia un'applicazione Java oppure una JSP. Nei due casi infatti, l'implementazione della chiamata SOAP è la stessa in quanto si utilizzano gli stessi oggetti. Il fatto che questi oggetti vengano utilizzati in contesti differenti, cioè applicazione Java di console oppure una pagina JSP, non è influente sul comportamento stesso degli oggetti.

Esempio: Calculator

Questo esempio prevede un servizio dal lato server scritto in Visual Basic ed un client scritto in Java. L'obiettivo dell'applicazione è fornire un servizio remoto che permetta di ottenere il risultato di un'operazione matematica utilizzando il metodo appropriato e passandogli i parametri elaborativi corretti. Le funzionalità supportate sono quattro: somma, sottrazione, moltiplicazione, divisione. Il servizio web, anche in questo caso, verrà poi utilizzato da un'applicazione java in running sull'application server Tomcat.

Server

Vediamo il codice sorgente del server, del progetto Calc.vbp:

```
Public Function getSum(ByVal n1 As Integer, ByVal n2 As Integer) As Integer
getSum = n1 + n2
End Function

Public Function getDifference(ByVal n1 As Integer, ByVal n2 As Integer) As
Integer
getDifference = n1 - n2
End Function
```

```
Public Function getMoltiplication(ByVal n1 As Integer, ByVal n2 As Integer)
As Integer
getMoltiplication = n1 * n2
End Function

Public Function getDivision(ByVal n1 As Integer, ByVal n2 As Integer) As
Integer
getDivision = n1 / n2
End Function
```

Anche qui possiamo vedere che il file sorgente del server scritto in Visual Basic non ha nulla di speciale. Come vediamo il numero dei metodi a disposizione è quattro e abbiamo due parametri di tipo numerici.

Per generare il componente COM, dobbiamo creare la libreria Calc.dll e poi è necessario registrare tale componente in modo che possa essere utilizzato da altre applicazioni; per fare ciò basta digitare dal prompt: `regsvr32 Calc.`

Apache SOAP fornisce il supporto per pubblicare servizi riguardanti componenti COM. Il deploy del servizio, si ottiene digitando dal prompt dei comandi:

```
java org.apache.soap.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter deploy
calc.xml
```

Dove il file XML che definisce il deployment descriptor, calc.xml, ha il seguente codice:

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
             id="urn:Calc">
 <isd:provider type="org.apache.soap.providers.com.RPCProvider"
               scope="Application"
               methods="getSum getDifference getMoltiplication getDivision">
```

```
        <isd:java class="required not needed for COMProvider"/>

        <isd:option key="progid" value="Calc.Calc"/>

      </isd:provider>

<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>

   </isd:service>
```

Questo file sappiamo contenere le informazioni necessarie per la pubblicazione, che sono:

- `id="urn:Calc"` : nome del servizio remoto
- `provider type="org.apache.soap.providers.com.RPCProvider"` : nome della classe Java che fornisce il supporto alla pubblicazione di componenti COM
- `scope="Application"` : indica il tipo di operazione eseguita dal server
- `methods="getSum getDifference getMoltiplication getDivision"` : nome dei metodi pubblicati

Client

Il codice del client è il seguente:

```
package esempi.calc;

import java.net.*;
import java.util.*;

import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class Calc
{
      public static void main(String[] args) {
            if (args.length != 3) {
                  System.err.println("Utilizzo: java esempi.calc.Calculator arg1
operatore arg2");
                  System.err.println("arg1 e arg2 sono numeri");
                  System.err.println("operatore è uno tra PIU MENO PER DIVISO");
                  System.exit(-1);
            }

            Call call = new Call();
            call.setTargetObjectURI("urn:Calc");

            String method="";

            if (args[1].toUpperCase().equals("PIU")) method="getSum";
            if (args[1].toUpperCase().equals("MENO")) method="getDifference";
            if (args[1].toUpperCase().equals("PER")) method="getMoltiplication";
            if (args[1].toUpperCase().equals("DIVISO")) method="getDivision";

            call.setMethodName(method);

            call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

            Vector parms = new Vector();
```

```
            parms.addElement(new Parameter("first", int.class, new
Integer(args[0]),null));
            parms.addElement(new Parameter("second", int.class, new
Integer(args[2]),null));

            call.setParams(parms);

            try {
                  URL url = new
URL("http://localhost:8080/soap/servlet/rpcrouter");

                  Response response = call.invoke(url, "");

                  if (response.generatedFault())
                        {
                         Fault fault = response.getFault();
                         System.err.println("------------------------------");
                         System.err.println("Attenzione: Condizione di Fault");
                         System.err.println("Codice: "+ fault.getFaultCode());
                         System.err.println("Descrizione: "+
fault.getFaultString());
                         System.err.println("------------------------------");
                        }
                        else {
                       Parameter result = response.getReturnValue();
                       System.out.println(args[0] + " " +args[1] + " " +
args[2] + " = " + ((Integer)result.getValue()).intValue());
                        }
            } catch (MalformedURLException mue) {
                  mue.printStackTrace();
            } catch (SOAPException se) {
                  se.printStackTrace();
            }
      }

}
```

Anche qui notiamo l'utilizzo dell'oggetto Call attraverso il codice:

```
Call call = new Call();

call.setTargetObjectURI("urn:Calc");
```

qui avendo più metodi a disposizione , il codice deve avere la possibilità di decidere quale metodo utilizzare e questo viene fatto attraverso la decodifica del secondo parametro passato al client. Il codice che effettua la selezione del metodo è il seguente:

```
String method="";

if (args[1].toUpperCase().equals("PIU")) method="getSum";

if (args[1].toUpperCase().equals("MENO")) method="getDifference";

if (args[1].toUpperCase().equals("PER")) method="getMoltiplication";

if (args[1].toUpperCase().equals("DIVISO")) method="getDivision";

call.setMethodName(method);
```

I parametri da passare sono due e di tipo int, vediamo il codice:

```
Vector parms = new Vector();
parms.addElement(new Parameter("first", int.class, new Integer(args[0]),null));
parms.addElement(new Parameter("second", int.class, new Integer(args[2]),null));
```

Poi, stabiliti il metodo da utilizzare ed i rispettivi parametri, è possibile effettuare la chiamata SOAP ed ottenerne la response, tramite il codice:

```
Response response = call.invoke(url, "");
```

Il resto del codice provvede ad utilizzare il risultato dell'operazione remota per produrre una stringa in output.

La compilazione del codice Java si ottiene attraverso il comando:

```
javac Calc.java
```

L'esecuzione di questa applicazione prevede il passaggio sulla linea di comando di tre parametri secondo la seguente struttura:

```
java esempi.calc.Calc 5 PIU 18
```

In questo caso il secondo parametro viene utilizzato per decidere quale dei metodi remoti debba essere invocato, mentre il primo e il terzo vengono passati direttamente al metodo remoto che restituirà il seguente risultato:

```
5 PIU 18 = 23
```

Il traffico TCP/IP

L'applicazione che stiamo utilizzando ha diverse possibilità di utilizzo, pertanto è importante capire cosa contiene il payload XML della request e della response nei diversi casi.

Vediamo la Request SOAP nel caso si utilizzi la funzione di somma di due numeri (22 e 55), passati sulla linea di comando:

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
```

```
Content-Lenght: 477
SOAPAction: ""

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:getSum
       xmlns:ns1="urn:Calc"
       SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <first xsi:type="xsd:int">5</first>
       <second xsi:type="xsd:int">18</second>
     </ns1:getSum>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Il metodo utilizzato, come si vede è getSum ed i parametri sono stati passati correttamente come interi. Negli altri casi non cambierebbe di molto la Request SOAP, sempre se il passaggio dei parametri avviene in modo corretto, e cioè con numeri interi; cambierebbe ovviamente il nome del metodo utilizzato. Vediamo la Response SOAP:

```
HTTP/1.0 200 ok
Content-Type: text/xml; charset=utf-8
Content-Lenght: 457
Set-Cookie2: JSESSIONID=le72zseuq1;Version=1;Discard;Path="/soap"
Set-Cookie: JSESSIONID=le72zseuq1;Path=/soap
Servlet-Engine: Tomcat Web Server/3.2.3 (JSP 1.1; Servlet 2.3; Java 1.3.0);
Windows 2000 5.0 x86; java.vendor=Sun Microsystems Inc.)

<?xml version='1.0' encoding='UTF-8'?>
```

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:getSumResponse
       xmlns:ns1="urn:Calc"
       SOAP-NV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <return xsi:type="xsd:int">23</return>
     </ns1: getSumResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

All'interno della Response SOAP, troviamo il valore di ritorno della chiamata al metodo remoto.

Condizione di Fault

Uno degli errori applicativi che andrebbe gestito a livello di codice server, trattandosi di un'applicazione che effettua calcolo, è la divisione per zero.

Se da linea di comando passiamo, per esempio, il numero 17 e il numero zero e invochiamo il metodo della divisione, il client effettua la lettura di tre parametri e di conseguenza invoca il metodo remoto getDivision passandogli come dividendo il valore 17 e come divisore il valore 0. Il server restituirà al client un messaggio di errore, e siccome l'eccezione non è stata gestita a livello di codice sorgente, il server sarà costretto a restituire un errore SOAP. L'output finale è il seguente:

```
. . . . . .
Attenzione: Condizione di Fault
Codice: SOAP-ENV:Server
Descrizione: Exception from service object: / by zero
. . . . . .
```

che, guardando il codice sorgente, ci accorgiamo essere prodotto dal codice client quando si rende conto che c'è stato un errore SOAP, cioè quando si è verificata una condizione di Fault. Le due informazioni:

`fault.getFaultCode()` che restituisce il valore `"SOAP-ENV:Server"`

`fault.getFaultString()` che restituisce il valore `"Exception from service object: / by zero"`

ci permettono di capire che l'errore SOAP che si è verificato è localizzato nel server e la sua descrizione parla di una divisione per zero. Il codice client quindi, è vero che non gestisce le possibili eccezioni applicative che possono verificarsi, però permette di capire che tipo di errore si è presentato ed in quale ambito.

Vediamo la Request SOAP:

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Lenght: 486
SOAPAction: ""

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:getDivision
       xmlns:ns1="urn:Calc"
       SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <first xsi:type="xsd:int">17</first>
```

```
        <second xsi:type="xsd:int">0</second>
      </ns1:getDivision>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Il payload XML dal punto di vista sintattico non ha nessun problema. La riga `<second xsi:type="xsd:int">0</second>` sarà quella che produrrà l'errore sul server.

Vediamo la Response SOAP:

```
HTTP/1.0 500 Internal Server Error
Content-Type: text/xml; charset=utf-8
Content-Lenght: 474
Set-Cookie2: JSESSIONID=2gcyiwz2k1;Version=1;Discard;Path="/soap"
Set-Cookie: JSESSIONID=2gcyiwz2k1;Path=/soap
Servlet-Engine: Tomcat Web Server/3.2.3 (JSP 1.1; Servlet 2.3; Java 1.3.0);
Windows 2000 5.0 x86; java.vendor=Sun Microsystems Inc.)

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
     <SOAP-ENV:Fault>
       <faultcode>SOAP-ENV:Server</faultcode>
       <faultstring>Exception from service object: / by zero</faultstring>
       <faultactor>/soap/servlet/rpcrouter</faultactor>
     </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

La chiamata http ha generato una condizione di 500 Internal Server Error, e nel payload, è presente la sezione: `<SOAP-ENV:Fault>` contenente tutti i dettagli della condizione di errore.

Capitolo 11 – Microsoft SOAP Toolkit

Introduzione

Come detto in precedenza, Microsoft è stata una tra le prime aziende a muoversi nella direzione di SOAP, anche per permettere a tutti gli sviluppatori che realizzano soluzioni distribuite utilizzando COM e COM+, di orientarsi verso uno standard devvero universale quale è SOAP. Il toolkit di Microsoft è composto essenzialmente da alcune utilità che permettono di creare uno strato di comunicazione SOAP attorno a qualsiasi componente COM progettato ad hoc. In particolare il SOAP Toolkit è composto da:

- Alcune librerie ActiveX da utilizzare come oggetti per lo sviluppo di applicazioni client
- Una utilità, il WSDL Generator, che permette la generazione dei file WSDL e WSML
- I filtri ISAPI per poter gestire i file WSDL e WSML da Internet Information Server
- Altre utilità

La filosofia di base di Microsoft Toolkit è la seguente: qualsiasi componente COM può essere utilizzato come fornitore di servizi via SOAP, a condizione che per esso venga predisposta un'opportuna coppia di file WSDL e WSML che contengano la descrizione del servizio da esporre attraverso Internet Information Server. Sarà lo stesso Internet Information Server, attraverso il filtro ISAPI opportuno, a saper gestire i file WSDL attraverso i quali il client richiederà il servizio al server. Il client pertanto, attraverso SOAP, farà una request HTTP al file WSDL generato appositamente per un certo componente COM attraverso il WSDL Generator.

Il filtro ISAPI installato come plugin in Internet Information Server si occuperà di trasformare la chiamata al file WSDL in una reale richiesta di servizio al componente COM sul server ed effettuerà la response SOAP contenente il payload di ritorno.

Anche nel caso di Microsoft SOAP Toolkit, ha senso parlare di installazione lato client ed installazione lato server. L'installazione client ha come obiettivo la copia e la registrazione degli oggetti ActiveX necessari per fare in modo che un'applicazione client che voglia utilizzare servizi SOAP possa farlo, nel caso specifico si tratta di una serie di librerie DLL da utilizzare come riferimenti all'interno degli ambienti di sviluppo che si utilizzano.

L'installazione server invece, ha come obiettivo l'installazione e la configurazione dei filtri ISAPI da utilizzare per fare in modo che Internet Information Server possa ricevere request di file WSDL.

Le caratteristiche supportate da SOAP Toolkit 3.0 sono le seguenti:

- Specifiche del consorzio W3C relative a WSDL 1.1
- Specifiche del consorzio W3C relative a SOAP 1.1
- Specifiche del consorzio W3C relative a XML Schema Part 0 (Primer), Part 1 (Structure) e Part (Datatypes).
- Array di tipo semplice e complesso e array multidimensionali
- Type Complex
- Supporto per operazioni WSDL RPC-encoded e document-literal.
- SOAP Headers

Il toolkit permette di eseguire chiamate a servizi e pubblicare questi attraverso due livelli di API (Application Program Interface): “high-level” oppure “low”. La scelta dipenderà dalle caratteristiche dei messaggi SOAP che si desidera inviare o dal livello di monitoraggio desiderato. Ovviamente le “high-level” API facilitano il lavoro dello sviluppatore il quale può ignorare diversi meccanismi interni come: connessioni, serializzazioni, deserializzazioni, ecc.

Fra le “high-level” API troviamo gli oggetti:

- SoapClient
- SoapServer

Fra le “low-level” API troviamo gli oggetti:

- SoapConnector
- SoapSerializer
- SoapReader

Esempio Codice Fiscale

L’obiettivo dell’applicazione CodiceFiscale è quello di avere un servizio web, si tratterà di un oggetto COM, che esporti un metodo chiamato CalcoloCF (), il quale, dopo aver ricevuto in input alcuni dati anagrafici: Cognome, Nome, Data di Nascita, Sesso e Comune di nascita, calcoli il codice fiscale. Il servizio web sarà un componente COM sviluppato in Visual Basic e registrato sotto forma di ActiveX DLL sul server.

Il server

Il codice sorgente del server sarà un progetto Visual Basic costruito in parte dal wizard di generazione di un nuovo componente ActiveX DLL. La sua unica proprietà Name ha valore CodiceFiscaleServer. Il codice sorgente del servizio è interamente racchiuso all'interno dell'unica classe esistente, il file CodiceFiscale.cls. Vediamone il codice:

```
Option Explicit

Public DB  As Database

Public COMUNI As Recordset


Function CalcoloCF(Cognome$, Nome$, DataNascita As Date, Sesso$, Comune$) As
String



    Dim Temp        As String

    Dim Vocali      As String

    Dim Consonanti  As String

    Dim I           As Integer

    Dim AppoNum     As Long

    Dim TempNum     As Long

    Dim TxtCodFis   As String


    TxtCodFis = ""


    '

    ' RICAVO IL COGNOME (123)

    '

    Cognome$ = StrConv(Cognome$, vbUpperCase)

    Vocali = ""

    Consonanti = ""

    For I = 1 To Len(Cognome$)

        If InStr("AEIOU", Mid(Cognome$, I, 1)) Then

            Vocali = Vocali + Mid(Cognome$, I, 1)

        ElseIf InStr("BCDFGHJKLMNPQRSTVWXYZ", Mid(Cognome$, I, 1)) Then
```

```
            Consonanti = Consonanti + Mid(Cognome$, I, 1)
        Else
            ' E' uno spazio, un apostrfo o altro che non va considerato
        End If
       If Len(Consonanti) = 3 Then Exit For
    Next
    If Len(Consonanti) < 3 Then Consonanti = Consonanti + Left(Vocali, 3 -
Len(Consonanti))
    If Len(Consonanti) < 3 Then Consonanti = Consonanti + String(3 -
Len(Consonanti), "X")
    TxtCodFis = Consonanti

    '
    ' RICAVO IL NOME (456)
    '
    Nome$ = StrConv(Nome$, vbUpperCase)
    Vocali = ""
    Consonanti = ""
    For I = 1 To Len(Nome$)
        If InStr("AEIOU", Mid(Nome$, I, 1)) Then
            Vocali = Vocali + Mid(Nome$, I, 1)
        ElseIf InStr("BCDFGHJKLMNPQRSTVWXYZ", Mid(Nome$, I, 1)) Then
            Consonanti = Consonanti + Mid(Nome$, I, 1)
        Else
            ' E' uno spazio, un apostrfo o altro che non va considerato
        End If
    Next I
    If Len(Consonanti) >= 4 Then
        ' isolo la prima, terza e quarta consonante
        Consonanti = Left$(Consonanti, 1) & Mid$(Consonanti, 3, 2)
    ElseIf Len(Consonanti) = 3 Then
        ' Va bene cosi'
```

```
Else
    ' Aggiungo le vocali
    Consonanti = Left$(Consonanti & Vocali, 3)
    ' se non basta, aggiungo le X
    If Len(Consonanti) < 3 Then Consonanti = Left$(Consonanti & "XXX", 3)
End If
TxtCodFis = TxtCodFis & Consonanti

'
'Anno di nascita (78)
'
TxtCodFis = TxtCodFis + Right(Format$(Year(DataNascita), "0000"), 2)

'
'Mese di nascita(9)
'
TxtCodFis = TxtCodFis & Mid$("ABCDEHLMPRST", Month(DataNascita), 1)

'
'Giorno di nascita(0A)
'
If UCase(Sesso$) = "F" Then
    TxtCodFis = TxtCodFis & Format$(Day(DataNascita) + 40, "00")
Else
    TxtCodFis = TxtCodFis & Format$(Day(DataNascita), "00")
End If

'
'Località di nascita (BCDE)
'
Set DB = OpenDatabase(App.Path & "\" & "Comuni.mdb", False, False)
```

```
    Set COMUNI = DB.OpenRecordset("Comuni", dbOpenSnapshot)
    COMUNI.MoveFirst
    If Not (COMUNI.EOF Or COMUNI.BOF) Then

        COMUNI.FindFirst ("COMU_DESCR = '" & Comune$ & "'")
        Comune$ = COMUNI.Fields(0)

        If COMUNI.NoMatch Then Comune$ = "XXXX"

    Else

        Comune$ = "XXXX"

    End If

    TxtCodFis = TxtCodFis & Comune$

    '
    'Ultima lettera (F)
    '
    'Controllo caratteri pari
    TempNum = 0
    I = 1
    Do
        ' I DISPARI
        AppoNum =
InStr("B1A0KKPPLLC2QQD3RRE4VVOOSSF5TTG6UUH7MMI8NNJ9WWZZYYXX", Mid(TxtCodFis,
I, 1))
        TempNum = TempNum + ((AppoNum - 1) And &H7FFE) / 2
        I = I + 1
```

```
            If I > 15 Then Exit Do

            ' I PARI
            AppoNum =
    InStr("A0B1C2D3E4F5G6H7I8J9KKLLMMNNOOPPQQRRSSTTUUVVWWXXYYZZ", Mid(TxtCodFis,
    I, 1))
            TempNum = TempNum + ((AppoNum - 1) And &H7FFE) / 2
            I = I + 1
        Loop
        TempNum = TempNum Mod 26
        TxtCodFis = TxtCodFis & Mid$("ABCDEFGHIJKLMNOPQRSTUVWXYZ", TempNum + 1,
    1)
        CalcoloCF = TxtCodFis

    End Function
```

Non ci soffermiamo ad analizzare l'algoritmo implementativo che è leggermente complesso, ma consideriamo il fatto che stiamo sviluppando un componente COM in grado di fornirci un codice fiscale a partire dai dati anagrafici di una persona.

Per questa applicazione utilizzeremo un databse locale, sul server, che contiene l'associazione tra i comuni d'Italia e la loro rispettiva codifica da applicare durante la produzione del codice fiscale.

Il metodo che trasformiamo in servizio remoto è CalcoloCF. Una volta generato il componente COM, una libreria DLL, registriamo il componente COM sul serever affinché possa essere utilizzato da altre applicazioni. Per avere la conferma dell'esatto funzionamento dell'oggetto COM, utilizziamo una pagina ASP di test e all'interno del codice inseriamo i parametri (Cognome, Nome, Data di Nascita, Sesso e Comune di nascita) che, a sua volta, li passerà all'oggetto COM il quale produrrà il codice fiscale. Il metodo CalcoloCF restituisce in output una stringa di testo, il codice fiscale richiesto, che la pagina ASP di test stampa a video attraverso l'istruzione response.write.

Ora per poter utilizzare questo componente COM da remoto via SOAP, dovremo procedere alla generazione dei file WSDL e WSML. La generazionedei file di descrizione del servizio può essere fatta manualmente, oppure si può utilizzare il comodo strumento WSDL Generator. Infatti basterà inserire i valori per il nome del servizio e per il nome del componente COM che vogliamo pubblicare, gli oggetti e i metodi da esportare come servizi web, la posizione del listener che conterrà i file di descrizione del servizio che stiamo generando e il tipo.

I file che verranno generati sono i seguenti:

CodiceFiscale.wsdl

```
<?xml version='1.0' encoding='UTF-8' ?>

 <!-- Generated 11/29/01 by Microsoft SOAP Toolkit WSDL File Generator,
Version 1.02.813.0 -->

<definitions  name ='CodiceFiscale'   targetNamespace =

'http://tempuri.org/wsdl/'

    xmlns:wsdlns='http://tempuri.org/wsdl/'

    xmlns:typens='http://tempuri.org/type'

    xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'

    xmlns:xsd='http://www.w3.org/2001/XMLSchema'

    xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'

    xmlns='http://schemas.xmlsoap.org/wsdl/'>

  <types>

    <schema targetNamespace='http://tempuri.org/type'

      xmlns='http://www.w3.org/2001/XMLSchema'

      xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'

      xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'

      elementFormDefault='qualified'>

    </schema>

  </types>

  <message name='CodiceFiscale.CalcoloCF'>

    <part name='Cognome' type='xsd:string'/>

    <part name='Nome' type='xsd:string'/>

    <part name='DataNascita' type='xsd:dateTime'/>

    <part name='Sesso' type='xsd:string'/>

    <part name='Comune' type='xsd:string'/>

  </message>

  <message name='CodiceFiscale.CalcoloCFResponse'>

    <part name='Result' type='xsd:string'/>

    <part name='Cognome' type='xsd:string'/>
```

```
    <part name='Nome' type='xsd:string'/>
    <part name='DataNascita' type='xsd:dateTime'/>
    <part name='Sesso' type='xsd:string'/>
    <part name='Comune' type='xsd:string'/>
  </message>
  <portType name='CodiceFiscaleSoapPort'>
    <operation name='CalcoloCF' parameterOrder='Cognome Nome DataNascita
Sesso Comune'>
      <input message='wsdlns:CodiceFiscale.CalcoloCF' />
      <output message='wsdlns:CodiceFiscale.CalcoloCFResponse' />
    </operation>
  </portType>
  <binding name='CodiceFiscaleSoapBinding'
type='wsdlns:CodiceFiscaleSoapPort' >
    <stk:binding preferredEncoding='UTF-8'/>
    <soap:binding style='rpc'
transport='http://schemas.xmlsoap.org/soap/http' />
    <operation name='CalcoloCF' >
      <soap:operation
soapAction='http://tempuri.org/action/CodiceFiscale.CalcoloCF' />
      <input>
        <soap:body use='encoded' namespace='http://tempuri.org/message/'
           encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </input>
      <output>
        <soap:body use='encoded' namespace='http://tempuri.org/message/'
           encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
      </output>
    </operation>
  </binding>
  <service name='CodiceFiscale' >
```

```
    <port name='CodiceFiscaleSoapPort'
binding='wsdlns:CodiceFiscaleSoapBinding' >
      <soap:address
location='http://localhost/esempisoap/codicefiscale/server/CodiceFiscale.WSDL
' />
    </port>
  </service>
</definitions>
```

CodiceFiscale.wsml:

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!-- Generated 11/29/01 by Microsoft SOAP Toolkit WSDL File Generator,
Version 1.02.813.0 -->
<servicemapping name='CodiceFiscale'>
  <service name='CodiceFiscale'>
    <using PROGID='CodiceFiscaleServer.CodiceFiscale' cachable='0'
ID='CodiceFiscaleObject' />
    <port name='CodiceFiscaleSoapPort'>
      <operation name='CalcoloCF'>
        <execute uses='CodiceFiscaleObject' method='CalcoloCF'
dispID='1610809344'>
          <parameter callIndex='1' name='Cognome' elementName='Cognome' />
          <parameter callIndex='2' name='Nome' elementName='Nome' />
          <parameter callIndex='3' name='DataNascita'
elementName='DataNascita' />
          <parameter callIndex='4' name='Sesso' elementName='Sesso' />
          <parameter callIndex='5' name='Comune' elementName='Comune' />
          <parameter callIndex='-1' name='retval' elementName='Result' />
        </execute>
      </operation>
    </port>
  </service>
```

```
</servicemapping>
```

Una volta prodotti questi file, abbiamo il wrapper per il servizio remoto, in questo modo possiamo utilizzarlo come web service via web attraverso Internet Information Server.

Applicazioni che utilizzano il servizio: client VB

Vediamo un primo client sviluppato in VB, dotato di un Form. Per far funzionare il servizio remoto è necessario affettuare l'inserimento di tutti i dati che servono all'algoritmo di calcolo del codice fiscale. Vediamo il codice sorgente:

```
Option Explicit


Private Client As SoapClient
Private sConnectedWSDL As String
Private WDSLString


Private Sub cmdCF_Click()


    Dim strSesso
    On Error GoTo ErrorHandler
    Me.MousePointer = vbHourglass
    Connect
    If Maschio Then strSesso = "M" Else strSesso = "F"
    txtCf.Text = Client.CalcoloCF(txtCognome.Text, txtNome.Text,
CVDate(txtGiorno + "/" + txtMese + "/" + txtAnno), strSesso, txtComune)
    Me.MousePointer = vbDefault
    Exit Sub


ErrorHandler:


    Me.MousePointer = vbDefault
    MsgBox Err.Description, vbExclamation
    MsgBox Client.faultcode, vbExclamation
```

```
End Sub

Private Sub Form_Load()

    WDSLString =
"http://localhost/esempisoap/codicefiscale/server/CodiceFiscale.wsdl"

End Sub

Private Sub Connect()

    If sConnectedWSDL <> WDSLString Then
        Set Client = New SoapClient
        Client.mssoapinit WDSLString
        sConnectedWSDL = WDSLString
    End If

End Sub
```

Il form contiene diverse text-box per l'inserimento dei parametri necessari per il calcolo del codice fiscale da parte dell'utente, che richiederà tale calcolo al servizio remoto. Partirà quindi la chiamata SOAP sul listener indicato all'interno del codice sorgente, dopodiché l'applicazione rimane in attesa di ricevere la response SOAP. Al termine, una volta ricevuta tale response, il valore di ritorno sarà il codice fiscale.

Traffico HTTP generato tra client e server

Vediamo il traffico HTTP generato tra client e server durante la richiesta del servizio remoto

Request SOAP

```
POST /soap/CODICEFISCALE/CodiceFiscale.wsdl HTTP/1.1
Content-Type: text/xml; charset=utf-8
Host: localhost
```

```
   SOAPAction: "http://tempuri.org/action/CodiceFiscale.CalcoloCF"

   Content-Lenght: 464

   <?xml version='1.0' encoding='UTF-8' standalone=no"?>
   <SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/>
       <SOAP-ENV:Body>
         <SOAPSDK1:CalcoloCF
          xmlns:SOAPSDK1 =" http://tempuri.org/message/"
          <Cognome>Feverati</Cognome>
          <Nome>Chiara</Nome>
          <DataNascita>1977-05-17T22:00:00Z</DataNascita>
          <Sesso>F</Sesso>
          <Comune>Modena</Comune>
         </SOAPSDK1:CalcoloCF>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

Response SOAP

```
HTTP/1.0 100 Continue
Server: Microsoft-IIS/5.0
Date: Thu, 15 Nov 2003 21:05:45 GMT

HTTP/1.0 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 15 Nov 2003 21:05:45 GMT
Content-Type: text/xml; charset="utf-8"
Expires: -1;
```

```
<?xml version='1.0' encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
  SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <SOAPSDK1:CalcoloCFResponse xmlns:SOAPSDK1 ="http://tempuri.org/message/">
          <Result>RSSMRA57H16L219T</Result>
          <Cognome>Feverati</Cognome>
          <Nome>Chiara</Nome>
          <DataNascita>1977-05-17T22:00:00Z</DataNascita>
          <Sesso>F</Sesso>
          <Comune>F257</Comune>
         </SOAPSDK1:CalcoloCFResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Qui si notano alcune differenze di gestione del payload tra Microsoft SOAP Toolkit ad Apache SOAP. In particolare si nota che all'interno del payload di ritorno, cioè all'interno della response SOAP, la presenza di tutti i parametri che erano stati passati in ingresso al metodo remoto oltre che del risultato dell'elaborazione. Poi la request SOAP generata dal SOAP toolkit implementa nativamente l'header HTTP SOAPAction, cosa che non accade per le applicazioni sviluppate con Apache SOAP.

Gestione del Fault

Proviamo a causare una condizione di Fault. Inseriamo nel servizio una data di nascita sbagliata, per esempio con mese=20. In questo caso ci troveremmo nella condizione particolare di cercare di chiamare il metodo remoto passandogli un parametro di tipo diverso da quello che lui si aspetta, cioè una data. Nel caso specifico però la chiamata al metodo remoto non causerà una condizione di Fault, ma una semplice condizione di errore a livello del codice client. Però se tentiamo di effettuare la chiamata, succede che il codice client effettua prima una connect, poi effettua la chiamata al metodo remoto. Poiché la connection è stata fatta utilizzando una GET HTTP sul file WSDL, la request HTTP è la seguente:

```
   GET /soap/codicefiscale/server/CodiceFiscale.wsd1 HTTP/1.1
   Accept: *1*
```

```
Accept-Encoding: gzip, deflate

User-Agent: Mozilla14.0 (compatible; MSIE 5.5; Windows NT 5.0)

Host: localhost

Connection: Keep-Alive
```

E la response HTTP:

```
HTTP/1.1 200 OK

Server: Microsoft-IIS/5.0

Date: Thu, 21 Nov 2003 22:37:09 GMT Content-Type: textlxml

Content-Length: 2726

Expires: -1;

<?xml version='1.0' encoding='UTF-8' ?>

<!-- Generated 11129/01 by Microsoft SOAP Toolkit WSDL File Generator,
Version 1.02.813.0—->

<definitions

   name =’CodiceFiscale'

   targetNamespace = 'http://tempuri.org/wsdl/'

   xmlns:wsdlns='http://tempuri.org/wsdl/'

   xmlns:typens='http://tempuri.org/type'

   xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'

   xmlns:xsd='http://www.w3.org/2001/XMLSchema'

   xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'

   xmlns='http://schemas.xmlsoap.org/wsdl/'>

   <types>

      <schema targetNamespace='http://tempuri.org/type'

       xmlns='http://www.w3.org/2001/XMLSchema'

       xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/’

       xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'

       elementFormDefault='qualified'>

      </schema>
```

```
</types>
<message
 name='CodiceFiscale.CalcoloCF'>
   <part name=Cognome' type='xsd:string'/>
   <part name='Nome' type='xsd:string'/>
   <part name=DataNascita' type='xsd:dateTime'/>
   <part name='Sesso' type='xsd:strìng'/>
   <part name='Comune' type='xsd:string'/>
</message>
<message
 name='CodiceFiscale.CalcoloCFResponse'>
 <part name='Result' type='xsd:string'/>
 <part name='Cognomel type='xsd:string'/>
 <part name='Nome' type='xsd:string'J>
 <part name='DataNascita' type='xsd:dateTime'/>
 <part name='Sesso' type='xsd:string'/>
 <part name='Comune' type='xsd:string'/>
</message>
<portType name='CodiceFìscaleSoapPort'>
  <operation
    name=TalcoloCF
    parameterOrder=Tognome Nome DataNascita Sesso Comune'>
    <input message='wsdlns:CodiceFiscale.CalcoloCF' />
    <output message='wsdlns:CodiceFiscale.CalcoloCFResponse' />
  </operation>
</portType>
<binding
 name='CodiceFiscaleSoapBinding'
 type='wsdlns:CodiceFiscaleSoapPort' >
 <stk:binding preferredEncoding=IUTF-8'/>
 <soap:binding
```

```
 style='rpc'
transport='http://schemas.xmlsoap.org/soap/http' >
<operation name=TalcoloCF >
<soap:operation
   soapAction='http://tempuri.org/action/CodiceFiscale.CalcoloCF' />
    <input>
      <soap:body
       use='encoded'
       namespace='http://tempuri.org/message/'
       encodingStyle='http://schemas.xmlsoap.org/soap/eneoding/' >
    </input>
     <output>
      <soap:body
       use='encoded'
       namespace='http://tempuri.org/message/'
       encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' >
     </output>
</operation>
</binding>
<service name='CodiceFiscale' >
 <port
   name='CodiceFisca1eSoapPort'
   binding='wsdlns:CodiceFiscaleSoapBinding' >
   <soap:address location='http:l/localhost/soap/
        codicefiscale/server/CodiceFiscale.WSDL' />
 </port>
</service>
</definitions>
```

La connect ha letto le impostazioni del servizio remoto usando il file WSDL che troviamo all'interno della response HTTP. In questo modo si è stabilita, all'interno dello spazio di indirizzamento del client, un'interfaccia di utilizzo diretta verso il codice dell'oggetto remoto, pertanto la chiamata allo stesso metodo che utilizzi una forma errata per i parametri verrà individuata come errore lato client senza la necessità di scomodare la chiamata SOAP e pertanto senza provocare condizioni di Fault. Quindi per creare una reale condizione di Fault dobbiamo agire direttamenet sul server. Per non toccare il codice sorgente e dover quindi ricompilare la DLL ed effettuare nuovamente la registrazione, sottraiamo un oggetto che viene usato dal servizio remoto stesso per poter effettuare l'operazione per cui è predisposto.

Allora rinominiamo il database all'interno del quale il servizio remoto và a cercare il codice del comune inserito sul client e passato come parametro elaborativi. Così l'errore si sposta dal client al servizio remoto e quindi dovrebbe causare una reale condizione di Fault.

Facendo la verifica ci accorgiamo che è successo quello che avevamo previsto: il clint richiama il servizio remoto utilizzando una lista di parametri corretta, ma il servizio remoto non è in grado di soddisfare la richiesta del client a causa di un errore interno, in particolare la mancanza del database di appoggio, pertanto quello che viene restituito al client è una response SOAP contenente una condizione di Fault.

Analisi del traffico http

Request SOAP:

```
POST /soap/codicefiscale/server/CodiceFiscale.WSDL HTTP/1.1
Content-Type: text/xml; charset="UTF.8"
Host: localhost
SOAPAction: "http://tempuri.org/action/CodiceFiscale.CalcoloCF"
Content-Length: 464

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
   <SOAPSDK1:CalcoloCF xmlns:SOAPSDK1="http://tempuri.org/message/">
```

```
      <Cognome>Feverati</Cognome>
      <Nome>Chiara/Nome>
      <DataNascita>1977-05-17T22:00:OOZ</DataNascita>
      <Sesso>F</Sesso>
      <Comune>Modena</Comune>
    </SOAPSDK1:CalcoloCF>
  </S0AP-ENV:Body>
</S0AP-ENV:Envelope>
```

Response SOAP:

```
HTTP/1.1 100 Continue
Server: Microsoft-IIS/5.
Date: Thu. 21 Nov 2003 22:18:55 GM

HTTP/1.1 500 Internal Server Error
Server: Microsoft-IIS15.0
Date: Thu, 21 Nov 2003 22:18:65 GMT
Content-Type: text/xml; charset="UTF-8"
Content-Length: 1298
Expires: -l;

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
         <SOAP-ENV:Fault>
               <faultcode>SOAP-ENV:Server</faultcode>
               <faultstring>
               Impossibile trovare il file &apos;C:\Soap\CodiceFiscale\
               ~-> Server\Comuni.mdb&apos;.
               </faultstring>
```

```
                <faultactor>
http://localhost/esempisoap/codicefiscale/server/CodiceFiscale.WSDL
                </faultactor>
                <detail>
                  <mserror:errorInfo
xmlns:mserror="http://schemas.microsoft.com/soap-toolkit/faultdetail/error/">
                  <mserror:returnCode>-2146825264</mserror:returnCode>
                  <mserror:serverErrorInfo>
                    <mserror:description>
Impossibile trovare il file &apos;C:\Soap\CodiceFíscale\
Server\Comuni.mdb&apos;.
                    </mserror:description>
                    <mserror:source>DAO.Workspace<lmserror:source>
                    <mserror:helpFile>jeterr35.hlp</mserror:helpFile>
                    <mserror:helpContext>5003024</mserror:helpContext>
                   </mserror:serverErrorInfo>
                   <mserror:callStack>
                    <mserror:callElement>
                    <mserror:component>WSDLOperation</mserror:component>
                      <mserror:description>
                        Executing method CalcoloCF failed
                      </mserror:description>
                      <mserror:returnCode>-2147352567</mserror:returnCode>
                     </mserror:callElement>
                    </mserror:callStack>
                   </mserror:errorInfo>
                </detai>
          </SOAP-ENV:Fault>
   </S0AP-ENV:Body>
</SOAP-ENV:Envelope>
```

Come si può verificare analizzando il payload della response che ha trasportato la condizione di Fault, sono presenti moltissime informazioni legate all'errore. Tutte queste informazioni possono essere utilizzate dal client, dal momento che le riceve, per mostrare all'utente che utilizza l'applicazione, il maggior numero di dettagli possibile.

Applicazioni che utilizzano il servizio: client ASP

La filosofia di utilizzo di un client ASP è leggermente diversa rispetto a quella di un client scritto in Visual Basic. La motivazione di fondo stà nel fatto che una pagina ASP gira essa stessa su un server applicativo, di conseguenza la definizione di comunicazione client-server non è del tutt precisa. Si tratta a tutti gli effetti della comunicazione tra un sever applicativo ed un altro che si realizza nel momento in cui un'applicazione tra un server applicativo ed un altro che si realizza nel momento in cui un'applicazione in running sul primo ha la necessità di utilizzare un servizio remoto messo a disposizione sul secondo. Una pagina ASP può essere scritta indifferentemente in VBScript oppure in Jscript, cambia la sintassi del linguaggio ma le cose che è possibile realizzare sono più o meno le stesse. Analizzeremo entrambi i casi presupponendo che la chiamata alla pagina ASP sia effettuata sempre attraverso l'utilizzo di una pagina HTML che poi passi i parametri alla pagina ASP per l'eleborazione. E sarà poi la stessa pagina ASP a recuperarli e a passarli al metod remoto.

Il client ASP – VbScript

Vediamo il codice :

```
<%@ LANGUAGE = VBScript %>


<HTML>

<HEAD>

<TITLE>Codice Fiscale</TITLE>

</HEAD>

<BODY>
```

```
<%

Dim soapclient

Const WSDL_URL =
"http://localhost/esempisoap/codicefiscale/server/CodiceFiscale.wsdl"

If IsEmpty(Application("CodiceFiscaleServer")) Then
    set soapclient = Server.createobject("MSSOAP.SoapClient")
    soapclient.ClientProperty("ServerHTTPRequest") = True
    soapclient.mssoapinit WSDL_URL
    Application.Lock
        If IsEmpty(Application("CodiceFiscaleServer")) Then
            set Application("CodiceFiscaleServer") = soapclient
        End If
    Application.UnLock
Else
    Set soapclient = Application("CodiceFiscaleServer")
End If
Dim nome
    cognome = Request.Form("cognome")
    nome = Request.Form("nome")
    datanascita = Request.Form("datanascita")
    sesso = Request.Form("sesso")
    comune = Request.Form("comune")
```

```
        res = soapclient.CalcoloCF(cognome,nome,datanascita,sesso,comune)
        response.write(res)
    %>
    </BODY>
</HTML>
```

Come si può notare la pagina ASP si comporta in modo molto simile al codice Visual Basic dell'applicazione client. Viene creato l'oggetto soapclient e su di esso, dopo aver definito l'interfaccia dei metodi remoti da questo esportati attraverso l'utilizzo del file WSDL, viene utilizzato il metodo CalcoloCF passandogli i parametri ricevuti via HTTP dalla pagina HTML di inserimento. Verificando il traffico HTTP notiamo che la chiamata SOAP è esattamente identica nei due casi di applicazione client Visual Basic oppure di pagina ASP. Questo è dovuto al fatto che entrambe utilizzano gli stessi oggetti, forniti dal Microsoft SOAP Toolkit, per effettuare la comunicazione SOAP.

Il client ASP – Jscript

Vediamo il codice JScript :

```
    <%@ LANGUAGE = JScript %>
    <HTML>
    <HEAD>
    <TITLE>Codice Fiscale/TITLE>
    </HEAD>
    <BODY>
    <%

      var WSDL_URL =
    "http://localhost/esempisoap/codicefiscale/server/CodiceFiscale.wsdl"

       var soapclient
```

```
    if (!Application("CodiceFiscaleServer")) {

        soapclient = Server.CreateObject("MSSOAP.SoapClient")
        soapclient.ClientProperty("CodiceFiscale") = true

soapclient.mssoapinit("http://localhost/esempisoap/codicefiscale/server/Codic
eFiscale.wsdl")

        Application.Lock
            if (!Application("CodiceFiscaleServer")) {
                Application("CodiceFiscaleServer") = soapclient
            }
        Application.UnLock
    } else {
       soapclient = Application("CodiceFiscaleServer")
    }
    var cognome, nome, datanascita, sesso, comune
    cognome = Request.Form("cognome")
    nome = Request.Form("nome")
    datanascita = Request.Form("datanascita")
    sesso = Request.Form("sesso")
    comune = Request.Form("comune")
    res = soapclient.CalcoloCF(cognome,nome,datanascita,sesso,comune)

%>
```

```
<%=res%><P><P>

</BODY>

</HTML>
```

La sintassi ovviamente è diversa ma la sostanza non cambia, tutti i client che utilizzano il Microsoft SOAP Toolkit per effettuare chiamate RPC attraverso SOAP utilizzano le stesse librerie e gli stessi componenti. Questo comporta il fatto che il traffico SOAP sia identico per tutte le chiamate allo stesso metodo remoto che utilizzano gli stessi parametri, anche se sono state effettuate attraverso l'utilizzo di client di tipologie diverse.

Esempio: Calculator

Questo esempio è stato studiato in precedenza nel caso di un client java e un server Visual Basic. La modalità di pubblicazione del servizio, che avveniva, nonostante fosse un componente COM, attraverso il web server Tomcat che risulta essere completemnete compatibile con Apache SOAP, era limitante. In genere la pubblicazione di un servizio COM avviene sul web server Internet Information Server (IIS), infatti questi due tipi di prodotti sono fortemente legati far loro facendo capo al mondo Microsoft. Inoltre nello studio dell'interoperabilità di SOAP, occorre prevedere che un client (Apache SOAP/Tomcat) si trovi su una piattaforma Unix, mentre il server (Visual Basic/IIS) si trovi su una piattaforma Windows. In questo esempio, in particolare, supponiamo di avere un client Apache SOAP e un server Visual Basic pubblicato su Internet Information Server utilizzando l'utility Microsoft SOAP Toolkit 3.0. Purtroppo in questo contesto non è stato possibile far interagire direttamente il client ed il server come impostati nelle prove precedenti. Questo perché fra le due implementazioni SOAP, quella di Apache e quella di Microsoft, vi sono delle differenze da non sottovalutare. Tra queste il fatto che, mentre Apache SOAP per tutti i parametri di richiesta e risposta specifica sempre la loro tipologia, l'implementazione Microsoft non lo fa.

Server

Come visto nell'esempio precedente, il codice del server è il seguente: (file class1.cls)

```
Public Function getSum(ByVal n1 As Integer, ByVal n2 As Integer) As Integer

getSum = n1 + n2

End Function


Public Function getDifference(ByVal n1 As Integer, ByVal n2 As Integer) As

Integer
```

```
getDifference = n1 - n2
End Function

Public Function getMoltiplication(ByVal n1 As Integer, ByVal n2 As Integer)
As Integer
getMoltiplication = n1 * n2
End Function

Public Function getDivision(ByVal n1 As Integer, ByVal n2 As Integer) As
Integer
getDivision = n1 / n2
End Function
```

Una volta compilato il componente e creata una libreria DLL che ospita il componente COM, il passo successivo consiste nel registrarla opportunamente.
Per la pubblicazione del servizio, utilizziamo lo strumento WSDL Generator di Microsoft SOAP Toolkit, generando i file WSDL e WSML. Nell'utilizzo di questa wizard non è necessario conoscere alcuna specifica per la creazione dei file WSDL e WSML. Le uniche informazioni che vengono richieste sono:

- Nome del servizio da pubblicare : `Calc`
- Posizione del componente COM : `c:\..\ Calc.dll`
- I metodi da esportare: `Class1`
- Posizione del listen del servizio : `http://localhost:80/ Calc`
- Tipologia del listen : ASP
- Il charset da utilizzare : UTF-8
- Il percorso dove memorizzare i file WSDL e WSML

Se il tutto è andato a buon fine, il wizard, nella directory da noi selezionata ha inserito i seguenti file:

- *Nomefile.asp*: questo file contiene le azioni che il listener eseguirà a fronte di una richiesta SOAP.

- *Nomefile.WSDL*

- Nomefile.wsml

Vediamo la servlet ASP, calculator.asp:

```
<%@ LANGUAGE=VBScript %>

<%

Option Explicit

On Error Resume Next

Response.ContentType = "text/xml"

Dim SoapServer

If Not Application("CalculatorInitialized") Then

  Application.Lock

  If Not Application("CalculatorInitialized") Then

    Dim WSDLFilePath

    Dim WSMLFilePath

    WSDLFilePath = Server.MapPath("Calculator.wsdl")

    WSMLFilePath = Server.MapPath("Calculator.wsml")

    Set SoapServer = Server.CreateObject("MSSOAP.SoapServer30")

    If Err Then SendFault "Cannot create SoapServer object. " &
Err.Description

    SoapServer.Init WSDLFilePath, WSMLFilePath

    If Err Then SendFault "SoapServer.Init failed. " & Err.Description

    Set Application("CalculatorServer") = SoapServer

    Application("CalculatorInitialized") = True

  End If

  Application.UnLock

End If
```

```
Set SoapServer = Application("CalculatorServer")

SoapServer.SoapInvoke Request, Response, ""

If Err Then SendFault "SoapServer.SoapInvoke failed. " & Err.Description

Sub SendFault(ByVal LogMessage)

  Dim Serializer

  On Error Resume Next

  ' "URI Query" logging must be enabled for AppendToLog to work

  Response.AppendToLog " SOAP ERROR: " & LogMessage

  Set Serializer = Server.CreateObject("MSSOAP.SoapSerializer30")

  If Err Then

    Response.AppendToLog "Could not create SoapSerializer30 object. " &
Err.Description

    Response.Status = "500 Internal Server Error"

  Else

    Serializer.Init Response

    If Err Then

      Response.AppendToLog "SoapSerializer.Init failed. " & Err.Description

      Response.Status = "500 Internal Server Error"

    Else

      Response.Status = "500 Internal Server Error"

      Serializer.startEnvelope

      Serializer.startBody

      Serializer.startFault "Server", "The request could not be processed due
to a problem in the server. Please contact the system admistrator. " &
LogMessage

      Serializer.endFault
```

```
        Serializer.endBody
        Serializer.endEnvelope
        If Err Then
          Response.AppendToLog "SoapSerializer failed. " & Err.Description
          Response.Status = "500 Internal Server Error"
        End If
      End If
    End If
    Response.End
  End Sub
  %>
```

Così si cerca di creare un Server SOAP con le specifiche definite dai file calculator.WSDL e calculator.wsml definiti con Microsoft SOAP Toolkit. Vediamo il file calculator.WSDL:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- Generated 01/01/04 by Microsoft SOAP Toolkit WSDL File Generator,
Version 3.00.1325.0 -->
<definitions
   name='Calculator'
   targetNamespace='http://tempuri.org/Calculator/wsdl/'
   xmlns:wsdlns='http://tempuri.org/Calculator/wsdl/'
   xmlns:typens='http://tempuri.org/Calculator/type/'
   xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'
   xmlns:dime='http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/'
   xmlns:ref='http://schemas.xmlsoap.org/ws/2002/04/reference/'
   xmlns:content='http://schemas.xmlsoap.org/ws/2002/04/content-type/'
   xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
   xmlns='http://schemas.xmlsoap.org/wsdl/'>
```

```
    <types>
          <schema
                targetNamespace='http://tempuri.org/Calculator/type/'
                xmlns='http://www.w3.org/2001/XMLSchema'
                xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
                xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
                elementFormDefault='qualified'>

                <import
namespace='http://schemas.xmlsoap.org/soap/encoding/'/>
                <import namespace='http://schemas.xmlsoap.org/wsdl/'/>
                <import
namespace='http://schemas.xmlsoap.org/ws/2002/04/reference/'/>
                <import
namespace='http://schemas.xmlsoap.org/ws/2002/04/content-type/'/>

          </schema>
    </types>

    <message name='Class1.getSum'>
          <part name='n1' type='xsd:short'/>
          <part name='n2' type='xsd:short'/>
    </message>

    <message name='Class1.getSumResponse'>
          <part name='Result' type='xsd:short'/>
    </message>

    <message name='Class1.getDifference'>
          <part name='n1' type='xsd:short'/>
          <part name='n2' type='xsd:short'/>
    </message>
```

```
<message name='Class1.getDifferenceResponse'>
      <part name='Result' type='xsd:short'/>
</message>

<message name='Class1.getMoltiplication'>
      <part name='n1' type='xsd:short'/>
      <part name='n2' type='xsd:short'/>
</message>

<message name='Class1.getMoltiplicationResponse'>
      <part name='Result' type='xsd:short'/>
</message>

<message name='Class1.getDivision'>
      <part name='n1' type='xsd:short'/>
      <part name='n2' type='xsd:short'/>
</message>

<message name='Class1.getDivisionResponse'>
      <part name='Result' type='xsd:short'/>
</message>

<portType name='Class1SoapPort'>

      <operation name='getSum' parameterOrder='n1 n2'>
            <input message='wsdlns:Class1.getSum'/>
            <output message='wsdlns:Class1.getSumResponse'/>
      </operation>

      <operation name='getDifference' parameterOrder='n1 n2'>
```

```
            <input message='wsdlns:Class1.getDifference'/>
            <output message='wsdlns:Class1.getDifferenceResponse'/>
        </operation>

        <operation name='getMoltiplication' parameterOrder='n1 n2'>
            <input message='wsdlns:Class1.getMoltiplication'/>
            <output message='wsdlns:Class1.getMoltiplicationResponse'/>
        </operation>

        <operation name='getDivision' parameterOrder='n1 n2'>
            <input message='wsdlns:Class1.getDivision'/>
            <output message='wsdlns:Class1.getDivisionResponse'/>
        </operation>

    </portType>

    <binding name='Class1SoapBinding' type='wsdlns:Class1SoapPort' >

        <stk:binding preferredEncoding='UTF-8'/>
        <soap:binding style='rpc'
transport='http://schemas.xmlsoap.org/soap/http'/>

        <operation name='getSum'>
            <soap:operation
soapAction='http://tempuri.org/Calculator/action/Class1.getSum'/>
            <input>
                <soap:body
                    use='encoded'
                    namespace='http://tempuri.org/Calculator/message/'
    encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                    parts='n1 n2'/>
            </input>
```

```
                <output>
                      <soap:body
                            use='encoded'
                            namespace='http://tempuri.org/Calculator/message/'
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                            parts='Result'/>
                </output>
          </operation>

          <operation name='getDifference'>
                <soap:operation
soapAction='http://tempuri.org/Calculator/action/Class1.getDifference'/>
                <input>
                      <soap:body
                            use='encoded'
                            namespace='http://tempuri.org/Calculator/message/'
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                            parts='n1 n2'/>
                </input>
                <output>
                      <soap:body
                            use='encoded'
                            namespace='http://tempuri.org/Calculator/message/'
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                            parts='Result'/>
                </output>
          </operation>

          <operation name='getMoltiplication'>
                <soap:operation
soapAction='http://tempuri.org/Calculator/action/Class1.getMoltiplication'/>
```

```
            <input>
                  <soap:body
                        use='encoded'
                        namespace='http://tempuri.org/Calculator/message/'
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                        parts='n1 n2'/>
            </input>
            <output>
                  <soap:body
                        use='encoded'
                        namespace='http://tempuri.org/Calculator/message/'
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                        parts='Result'/>
            </output>
      </operation>

      <operation name='getDivision'>
            <soap:operation
soapAction='http://tempuri.org/Calculator/action/Class1.getDivision'/>
            <input>
                  <soap:body
                        use='encoded'
                        namespace='http://tempuri.org/Calculator/message/'
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                        parts='n1 n2'/>
            </input>
            <output>
                  <soap:body
                        use='encoded'
                        namespace='http://tempuri.org/Calculator/message/'
```

```
   encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                            parts='Result'/>
                </output>
          </operation>

   </binding>

   <service name='Calculator' >
          <port name='Class1SoapPort' binding='wsdlns:Class1SoapBinding' >
                <soap:address
location='http://localhost:80/Calculator/Calculator.ASP'/>
          </port>
   </service>

</definitions>
```

Invece il file calculator.wsml è il seguente:

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!-- Generated 01/01/04 by Microsoft SOAP Toolkit WSDL File Generator,
Version 3.00.1325.0 -->
  <servicemapping name='Calculator'
   xmlns:dime='http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/'>
  <service name='Calculator'>
    <using PROGID='Progetto1.Class1' cachable='0' ID='Class1Object' />
    <port name='Class1SoapPort'>
      <operation name='getSum'>
        <execute uses='Class1Object' method='getSum' dispID='1610809344'>
          <parameter callIndex='-1' name='retval' elementName='Result' />
          <parameter callIndex='1' name='n1' elementName='n1' />
          <parameter callIndex='2' name='n2' elementName='n2' />
        </execute>
      </operation>
```

```
      <operation name='getDifference'>
        <execute uses='Class1Object' method='getDifference'
dispID='1610809345'>
          <parameter callIndex='-1' name='retval' elementName='Result' />
          <parameter callIndex='1' name='n1' elementName='n1' />
          <parameter callIndex='2' name='n2' elementName='n2' />
        </execute>
      </operation>
      <operation name='getMoltiplication'>
        <execute uses='Class1Object' method='getMoltiplication'
dispID='1610809346'>
          <parameter callIndex='-1' name='retval' elementName='Result' />
          <parameter callIndex='1' name='n1' elementName='n1' />
          <parameter callIndex='2' name='n2' elementName='n2' />
        </execute>
      </operation>
      <operation name='getDivision'>
        <execute uses='Class1Object' method='getDivision'
dispID='1610809347'>
          <parameter callIndex='-1' name='retval' elementName='Result' />
          <parameter callIndex='1' name='n1' elementName='n1' />
          <parameter callIndex='2' name='n2' elementName='n2' />
        </execute>
      </operation>
    </port>
  </service>
</servicemapping>
```

Analizziamo il file wsml.

- `<servicemapping name='Calculator'>` L'elemento `servicemapping` è la radice del documento wsml e a sua volta contiene tre elementi figli: `service, port` ed `operation.`
- `<service name='Calculator'>`

```
<using PROGID='Progetto1.Class1' cachable='0' ID='Class1Object'/>
```

Il nome del service indentifica a quale servizio (elemento `service`), del corrispondente file WSDL, si fa riferimento. L'elemento `service` a sua volta contiene l'elemento figlio `using`, il quale identifica l'oggetto COM necessario al servizio.

- `PROGID='Progetto1.Class1'`: questo attributo identifica la classe COM che implementa tutti i metodi del servizio in esame
- `cachable='0'` : questo attributo booleano specifica se l'istanza della classe (l'oggetto COM) deve rimanere in memoria per tutto il tempo in cui vi rimane l'oggetto `soapServer` (valore 1) oppure no (valore 0).
- `ID='Class1Object'`: questo attributo specifica il nome con cui l'oggetto COM sarà riferito nel seguito del documento WSML.

- `<port name='Class1SoapPort'>` : Il `name` del `port` identifica a quale elemento `portType`, del corrispondente file WSDL, si fa riferimento.
- `<operation name='getSum'>` : Per ogni operzione definita all'interno del `PortType, c'è` un elemento operation nel file WSML corrispondente.
- `<execute uses='Class1Object' method='getDifference' dispID='1610809345'>` Questo elemento, figlio di `operation`, specifica l'oggetto che ha il compito di eseguire l'operazione in esame.
  - `Uses` : identifica il nome dell'oggetto
  - `Method` : identifica il nome del metodo
  - `dispID` : fornisce il "dispID" del metodo. La presenza di questo attributo è opzionale ma migliora le performance.
- `<parameter callIndex='-1' name='retval' elementName='Result' />` questo elemento, figlio dell'elemento `execute`, descrive un parametro per il metodo in esame.
  - `callIndex` : fornisce il numero del parametro (sono in ordine rescente). Il valore -1 identifica il parametro di ritorno.
  - `Name` : fornisce un nome univoco per il parametro
  - `elementName` : fornisce il nome dell'elemento, nella sezione message del documento WSDL, che contiene il valore del parametro.

Client

Il codice del client è il seguente:

```
package esempi.calculator;

import java.net.*;
import java.util.*;
```

```
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class Calc
{
      public static void main(String[] args) {
            if (args.length != 3) {
                  System.err.println("Utilizzo: java esempi.calc.Calculator arg1
operatore arg2");
                  System.err.println("arg1 e arg2 sono numeri");
                  System.err.println("operatore è uno tra PIU MENO PER DIVISO");
                  System.exit(-1);
            }

            Call call = new Call();
            call.setTargetObjectURI("urn: CalculatorServer ");

            String method="";

            if (args[1].toUpperCase().equals("PIU")) method="getSum";
            if (args[1].toUpperCase().equals("MENO")) method="getDifference";
            if (args[1].toUpperCase().equals("PER")) method="getMoltiplication";
            if (args[1].toUpperCase().equals("DIVISO")) method="getDivision";

            call.setMethodName(method);

            call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

            Vector parms = new Vector();
            parms.addElement(new Parameter("first", int.class, new
Integer(args[0]),null));
            parms.addElement(new Parameter("second", int.class, new
Integer(args[2]),null));

            call.setParams(parms);

            try {
                  URL url = new
URL("http://localhost:8080/soap/servlet/rpcrouter");

                  Response response = call.invoke(url, "");

                  if (response.generatedFault())
                        {
                         Fault fault = response.getFault();
                         System.err.println("-------------------------------");
                         System.err.println("Attenzione: Condizione di Fault");
                         System.err.println("Codice: "+ fault.getFaultCode());
                         System.err.println("Descrizione: "+
fault.getFaultString());
                         System.err.println("-------------------------------");
                        }
                        else {
                        Parameter result = response.getReturnValue();
                        System.out.println(args[0] + " " +args[1] + " " +
args[2] + " = " + ((Integer)result.getValue()).intValue());
                  }
            } catch (MalformedURLException mue) {
                  mue.printStackTrace();
            } catch (SOAPException se) {
                  se.printStackTrace();
            }
      }

}
```

ERRORE!! Impossibilità di far interagire il client e il server.