Implementare Web Service tramite AXIS2

-Architettura-

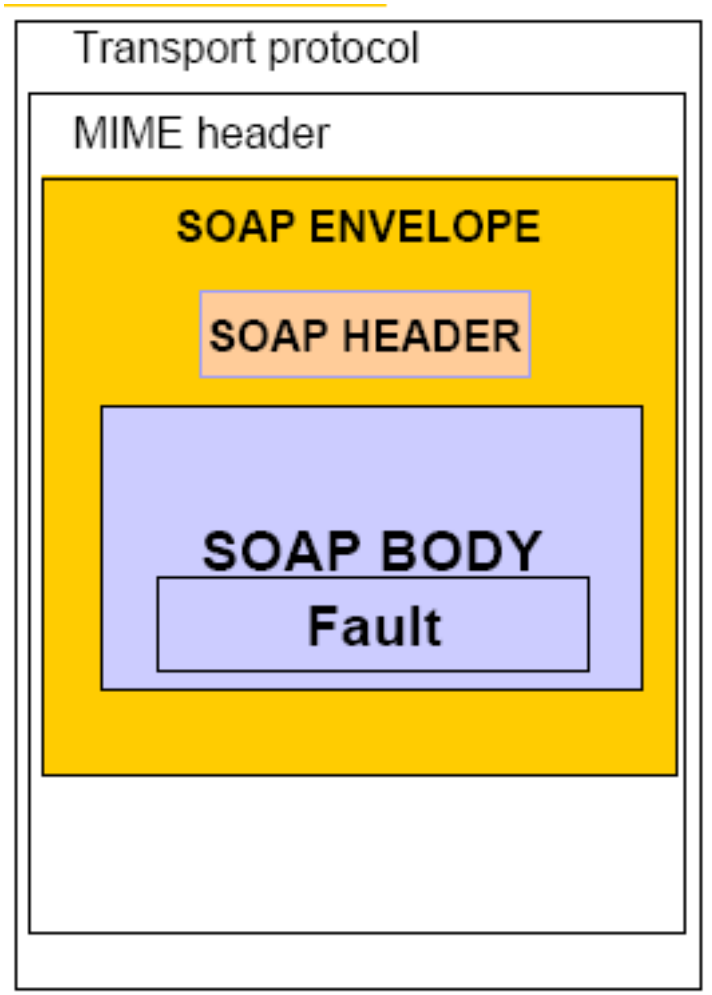
Ing. Buda Claudio
claudio.buda@unibo.it

Outline

- Stumenti
  - SOAP
  - AXIS
  - AXIS2
- Modello Axis2
  - Core Modules
  - Other Modules
- Tutorial

SOAP - Simple Object Access Protocol

SOAP provvede a fornire un framework per lo scambio di messaggi XML. I messaggi SOAP possono essere trasportati su diversi protocolli di rete. Un messaggio SOAP rappresenta sia l'informazione che necessita per invocare e rispondere un servizio, sia (nel caso di utilizzo RPC dei servizi) invocazione di metodi remoti.



Come nasce AXIS

AXIS è un implementazione del “Simple Object Access Protocol” (SOAP), nasce come suo proseguimento, alla fine del 2000, specificato dal W3C.

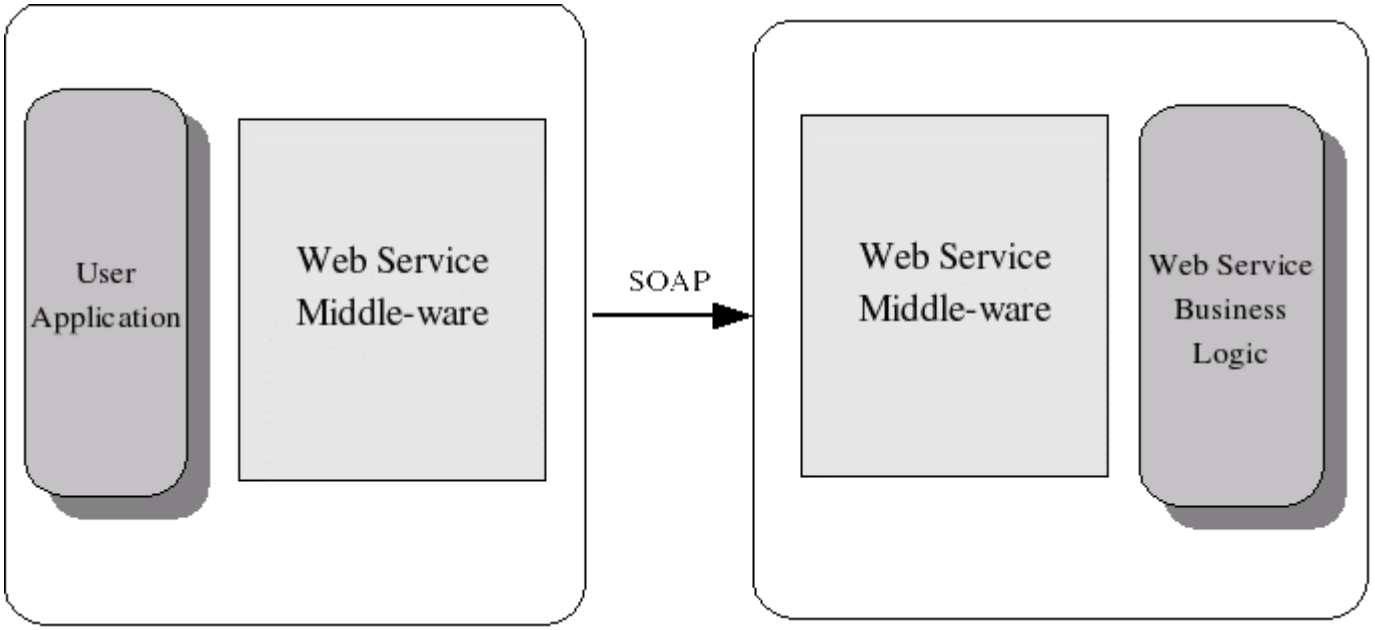
Risulta la base per implementare Java Web Services.

AXIS deriva da “Apache eXtensible Interaction System” ma si potrebbero avere parecchi dubbi sulla pertinenza dell'acronimo.

Qualsiasi cliente che vuole usare un WS scritto con AXIS può farlo poiché fornisce in maniera standard la WSDL.

AXIS è essenzialmente un engine SOAP, viene scritto in JAVA ma in parallelo va avanti anche un progetto in C++.

Modello AXIS come processore SOAP



AXIS

Ma non è solo un motore SOAP. Esso inoltre:

- Include un semplice server stand-alone;
- può inserirsi come servlet plugin dentro TOMCAT;
- è un supporto estensivo per le WSDL;
- può generare il codice dei servizi dalla WSDL;
- contiene altri semplici programmi fra cui un tool per monitorare i pacchetti TCP/IP.

Come nasce AXIS2

Nell' agosto del 2004 durante un Summit di AXIS in Sri Lanka fu introdotta una nuova architettura mantenendo il concetto della catena degli handler già presente in AXIS.

AXIS VS AXIS2

- Migliore architettura, pubblicazione semplificata, e più supporto data binding.
- Il grande obiettivo di AXIS2 è stato quello di fare espandere la piattaforma attraverso i nuovi protocolli che stanno nascendo come WS-Security, WS-Addressing, WS-ReliableMessaging. Ovviamente tutto questo a discapito della compatibilità con AXIS

AXIS2: Features (1)

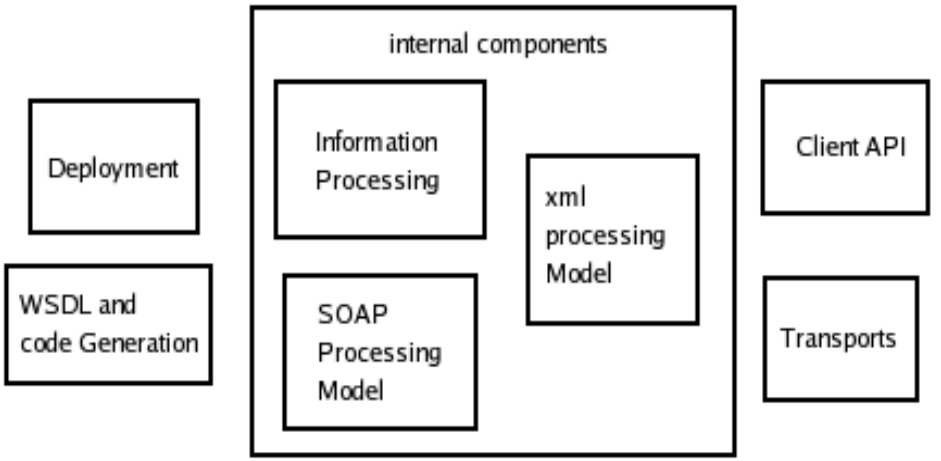
- Asincronia: Supporta la chiamata non bloccante dei client
- Velocità: attraverso il suo Object Model e lo StAX (Streaming API for XML) Parser, questa è aumentata significativamente.
- AXIOM: è l'Object Model creato appositamente per processare messaggi; estensibile e performante.
- Hot Deployment: è possibile pubblicare servizi ed handler mentre il sistema gira.
- MEP (Message Exchacge Pattern) Support

AXIS2: Features (2)

- Framework di trasporto: E' possibile spedire e ricevere messaggi SOAP sopra diversi protocolli; il cuore dell'engine è completamente indipendente da questo.
- WSDL Support: Supporta il Web Service Description Language che permette di costruire automaticamente stub per richiedere il servizio ed esportare la descrizione di questo per farsi conoscere.
- Componibilità ed Estendibilità: Attraverso i moduli è possibile aggiungere nuove WS-* (specifiche) che però non saranno “hot deployable”.

AXIS 2 Core Modules and Other Modules

- Information Model
- XML Processing Model
- Soap Processing Model
- Deployment Model
- Client API
- Trasports
- Code generation
- Data binding



Architettura principale

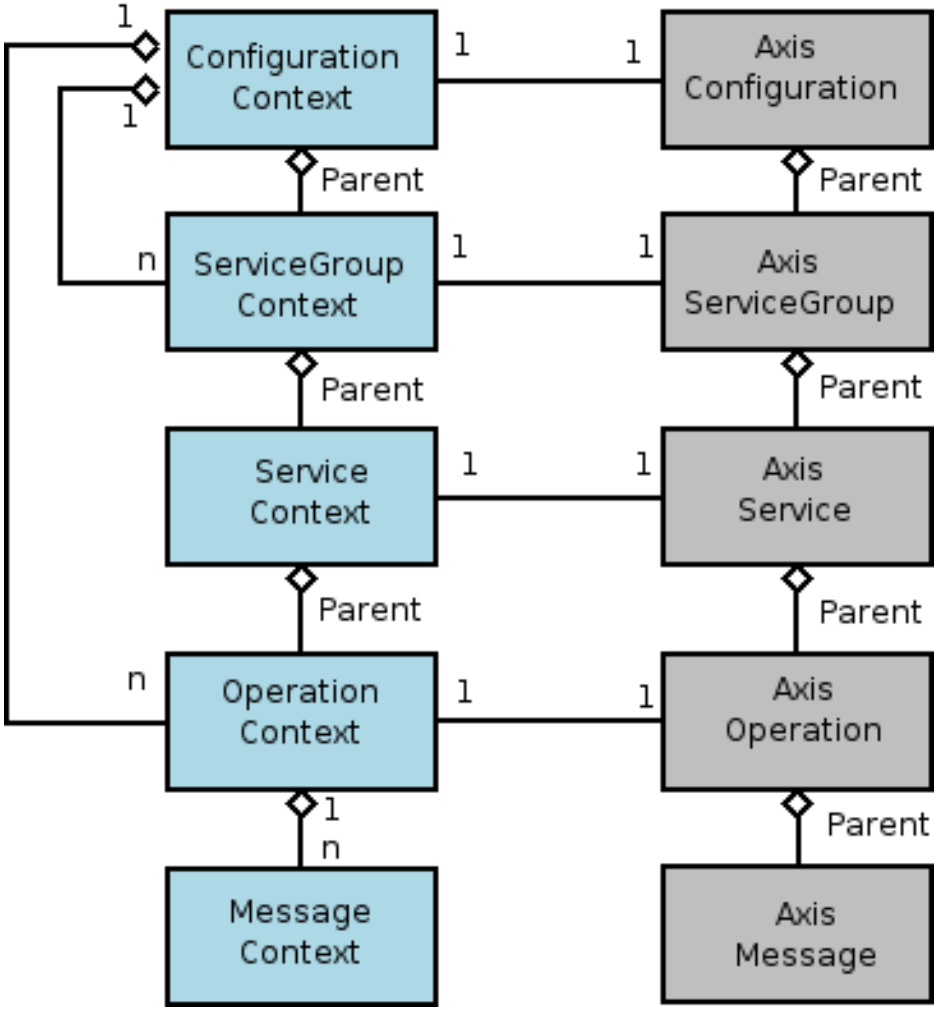
- L'architettura separa logica da stato. Il codice che fa il processamento dentro AXIS2 è senza stato. Il codice è eseguito attraverso liberi threads paralleli
- Tutte le informazioni sono mantenute in un information model per permettere al sistema di fermarsi e ripartire.
- L'architettura è modulare. Il framework è costruito sopra i moduli core, potendone inoltre aggiungerne di nuovi.

Information Model

Serve per mantenere le informazioni dello stato.

Questo modello contiene una gerarchia delle informazioni, più precisamente due, una detta gerarchia di Contesto e l'altra di Descrizione.

La prima rappresenta i dati statici. La seconda mantiene i dati inerenti alle istanze, i contesti dinamici.



XML Processing Model

Permette di gestire i messaggi SOAP. Questo viene fatto attraverso il nuovo Object Model AXIOM. Infatti è qui che vengono stabilite le performance del sistema.

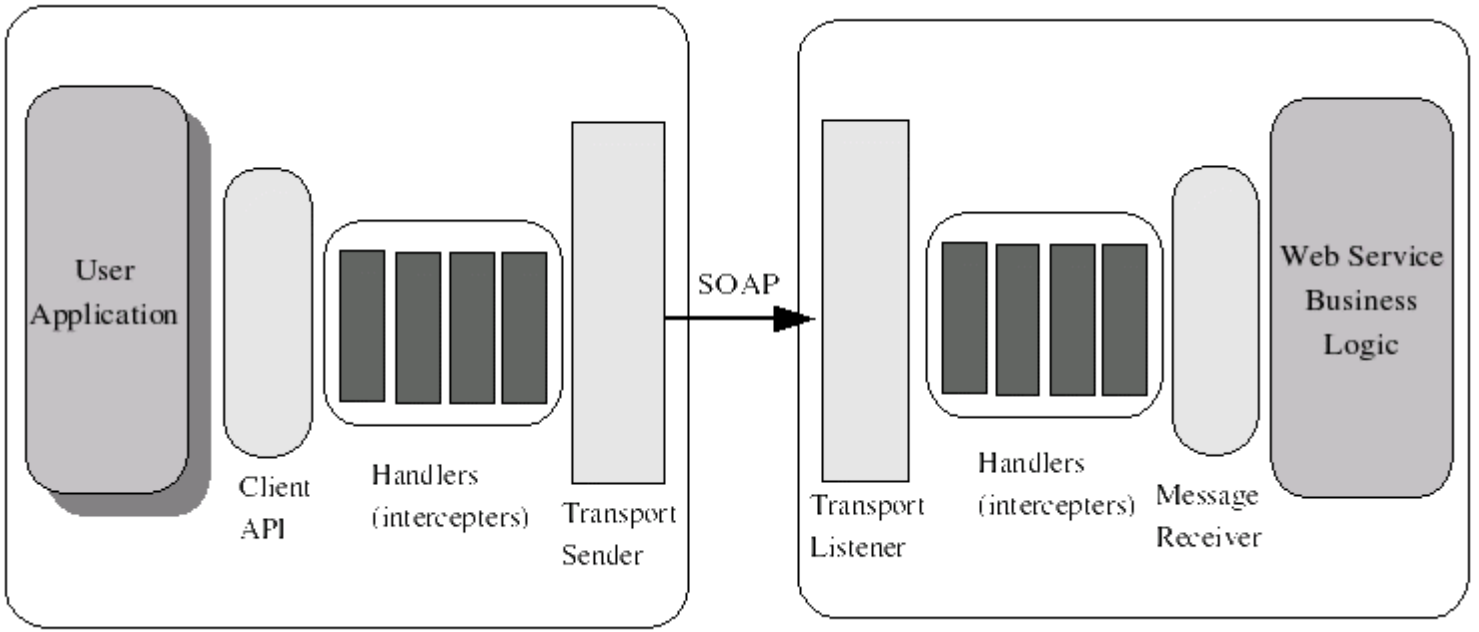
Da qui si sviluppa un sottoprogetto di AXIS2 chè è quello di AXIOM (Axis Object Model) per fornire della API per SOAP e le sue informazioni in formato XML.

AXIOM è differente dagli altri standard OM (SAX, DOM, JDOM..), principalmente per la possibilità di aumentare la struttura durante la ricezione dell'XML.

Esso dipende direttamente da StAX per input e output.

SOAP Processing Model

L'architettura prevede due “pipe” per compiere spedizione e ricezione. Con la combinazione di queste porta ad avere MEP complessi.



MEP (Message Exchange Pattern)

Un Message Exchange Pattern è un template, sprovvisto di applicazione sematica, che descrive un pattern generico per lo scambio di messaggio fra enti. Questo descrive relazioni (temporali, condizionate, sequenziali) di scambio di messaggi multipli.

SOAP Processing Model

L'estensibilità di questo processo è fornito attraverso gli handlers. Quando un messaggio viene processato ogni handler che fa parte della catena viene messo in esecuzione.

Gli handler sono come intercettori del messaggio e processano parti di esso (solitamente l'header) per fornire servizi aggiuntivi.

Solitamente il percorso effettuato da un messaggio consiste nell'uscire dalla OutPipe del client ed entrare nella InPipe del server ma può capitare al termine di questa la creazione di un nuovo messaggio di risposta da spedire. Si avrà allora un OutPipe lato server e InPipe lato client.

SOAP Processing Model

- Un messaggio all'interno delle pipe attraversa diverse fasi.
- Gli handler sono in esecuzioni all'interno delle fasi, queste infatti prevedono un meccanismo per specificare l'ordine degli handlers.
- Gli handler definiti di default da Axis sono i seguenti:
  - Dispatchers: cerca il servizio e l'operazione a cui è diretto il messaggio
  - Message Receiver: posizionato alla fine della InPipe questo handler consuma il messaggio
  - Transport Sender: alla fine della OutPipe spedisce il messaggio SOAP
  - Transport Receiver: riceve i messaggi in arrivo dal mittente

SOAP Processing Model: InPipe

- Una volta che il Transport Receiver riceve il messaggio viene creato un Message Context per esso. La InPipe viene quindi eseguita con questo contesto.
- Le fasi dei messaggi in arrivo
  - Transport Phase
  - Pre-Dispatch Phase
  - Dispatch phase
  - User phase
  - Message validation Phase
  - Message Processing Phase

InPipe phases: Transport phase

- Nella fase di Transport gli handler processano informazioni specifiche al trasporto come la validazione di messaggi in entrata osservando gli header dei messaggi, aggiungendo dati nel Message Context.

InPipe phases: Pre-Dispatch e Dispatch Phases

- Nella fase di Pre-Dispatch gli handler dovranno riempire il Message Context per la fase successiva. Un esempio di ciò è l'estrarre informazioni dall'header riguardanti l'addressing.
- Durante la fase di Dispatch vengono ricercati i servizi da mettere in esecuzione data la richiesta del messaggio. La post-condizione di questa fase, permette di lanciare un errore nel caso il servizio non sia stato trovato

InPipe phases: User Phases

In questa fase l'utente ha la possibilità di inserire i propri handler. In realtà dalla nuova versione 1.0 di Axis2 l'utente può scegliere di inserire i propri handler in qualsiasi fase, così da poter mettere mano anche a processi quali l'addressing.

Questa è stata anche una nostra necessità.

InPipe phases: Message Validation & Message Processing Phases

- Nella fase di validazione del messaggio viene assicurata la corretta esecuzione della fase successiva.
- La Business Logic del messaggio SOAP è eseguita qui. Ogni operazione associata al messaggio viene messa in esecuzione dal Message Receiver associato.

SOAP Processing Model: OutPipe

- La OutPipe è più semplice perchè quando viene messa in esecuzione già i servizi e le operazioni definiti nei messaggi sono conosciuti.
- Le fasi dei messaggi in uscita sono le seguenti:
  - Message Initializate Phase
  - User Phases
  - Transport phases

OutPipe Phases: Message Initializate & User Phases

- La prima fase servirà per piazzare gli handler “custom”
- La seconda fase sarà quella dove l'utente può inserire i propri handler. Si ricordi però quanto detto prima sulla versione 1.0 di Axis

OutPipe Phases: Transport phase

Esegue tutti gli hanlder inerenti alla configurazione del trasporto. L'ultimo handler è il Transport Senders che spedisce il messaggio SOAP all'endpoint

SOAP Processing Model: HANDLERS

Una importante peculiarità di Axis2 è la possibilità di inserire facilmente handlers che dovranno essere inseriti in una specifica fase.

Ogni handler deve essere inserito in un proprio modulo per essere configurato in modo da definire la posizione di esecuzione all'interno delle pipe.

SOAP Processing Model: MODULES

Un modulo è un pacchetto che contiene handler e un descrittore di essi (regole per le fasi).

Un modulo può essere “available” o “engaged”. Quando un modulo è available i suoi handler non sono nella catena ma sono disponibili. Una volta che il modulo è engaged i moduli al ricevimento del messaggio vanno in esecuzione. (e.g.: WS-Addressing, SoapMonitor)

Deployment Model

- axis2.xml file: file di configurazione globale
- Service Archive (.aar): Archivio contenente il web service
- Module Archive (.mar): Archivio contenente il modulo con gli handler

Deployment Model: axis2.xml

Il file axis2.xml risiede nella cartella WEB-INF/conf del WS. Esso gestisce:

- Parametri globali
- Trasporti di in & out
- Fasi dell'utente
- Moduli engaged per tutti i servizi
- Message Receivers globali (fanno i due MEP)

Deployment Model: Service Archive

Service Archive (.aar) è un jar da comporre con una determinata struttura, nella quale, dentro la cartella META-INF contiene service.xml. Questo configura:

- Parametri del servizio
- Moduli “engaged” per il servizio
- Message Receivers del servizio
- Operazioni del servizio

Questi archivi sono hot deployable a differenza di quelli dei moduli.

Deployment Model: Module Archive

Anche questo archivio deve essere preparato come un jar e contiene al suo interno il file module.xml dentro la cartella META-INF. Esso contiene parametri e operazioni definite nel modulo.

Quando il sistema parte, viene creato attraverso il Deployment Model un Axis Configuration. Deployment Model definisce le configurazioni globali attraverso il file axis2.xml. Poi cerca archivi di moduli e infine archivi di servizi. Dopo ciò moduli e servizi vengono aggiunti all'Axis Configuration rendendo perciò il sistema abile di funzionare.

Client API

- Fornisce all'utente le API necessarie per utilizzare un servizio, interagire con diversi MEPs fra cui IN-OUT e IN-ONLY (utilizzabili per costruirne altri)
- Ci sono tre parametri che decidono la natura dell'interazione fra client e service:
  - MEP
  - Transport
  - Synchronous/ Asynchronous
- La classe Service Client sarà quella utilizzata per configurare l'interazione ed effettuarla.

I DUE MEP di DEFAULT

- One Way: è una “fire&forget”, che non attende una risposta dal server. Per la sua invocazione si possono utilizzare i protocolli HTTP, SMTP, TCP.
- In-Out (Request Response): è una “sendReceive” e ci sono 4 modi per configurarla
  - Blocking o non blocking
  - Definire il protocollo di trasporto per spedire
  - Definire il protocollo sul quale ricevere la risposta
  - Utilizzo canale serparato: quando la risposta è spedita sopra una connessione di trasporto diversa dalla richiesta

Transport

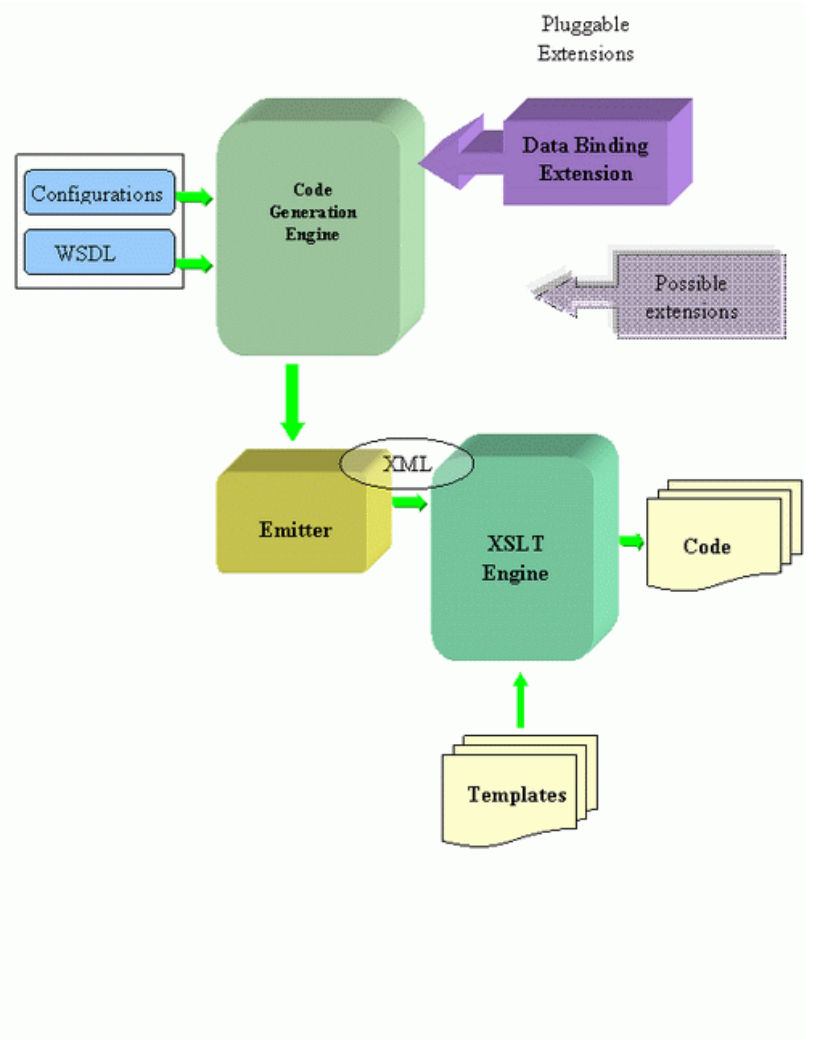
- Il trasporto che sostiene la risposta è deciso attraverso le informazioni di addressing inviate dal mittente.
- Se il trasporto non è specificato, il server risponderà automaticamente sullo stesso trasporto.
- Il client è libero di utilizzare il trasporto che preferisce.
- Il trasporto può avvenire su diversi protocolli:
  - HTTP: protocollo dual channel (two way) che mantiene traccia della sessione
  - TCP: è il più semplice, ma necessita di WS Addressing per essere funzionale
  - SMTP. il transport receiver è un thread che controlla la posta
  - JMS: protocollo di scambio di messaggi asincrono su Java

Other Modules (1)

- Code Generation – Permette di generare sia a lato server che a lato client automaticamente il codice per semplificare la pubblicazione del servizio e la sua invocazione. Le informazioni vengono estratte da una WSDL che descrive il WS e immesse in un XML indipendente dal linguaggio di output. Una volta “parsato”, viene generato il codice nel linguaggio prescelto per fare il servizio.
- Data Binding – Permette di estendere il modulo precedente per avere più controllo sui dati con la generazione automatica del codice. Ce ne sono di diversi tipi, e permettono il passaggio da dati in formato xml al linguaggio voluto. (XMLBeans).

Other Modules (2)

- Il rischio di utilizzare i tool di generazione automatica del codice risulta quello che avendo uno stub a lato client e uno skeleton che rappresenta il servizio a lato server si perda la concezione dei WS come porte (SOA).
- Infatti a questo punto l'utente a lato client richiamando un metodo della classe stub non percepisce lo scambio di messaggi SOAP ma solamente l'invocazione di un metodo remoto.



Link utili

- http://ws.apache.org/soap/
- http://ws.apache.org/axis/
- http://ws.apache.org/axis2/
- http://ws.apache.org/commons/axiom/
- http://today.java.net/pub/a/today/2005/05/10/axiom.html

Nel tutorial...

- Installazione TOM-CAT
- Inserimento del WAR di axis2
- Creazione servizio e relativo archivio (.aar)
- Montaggio SOAP monitor (come reperire file dal loro repository di codice sorgente)
- Creazione clients che usano il servizio
- Creazione Moduli e relativo archivio (.mar)
- Come usare Google WS