Università degli studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Sede di Modena
Corso di Laurea in Ingegneria Informatica
Nuovo Ordinamento

Sperimentazione di un ambiente Oracle per lo sviluppo di processi BPEL

Relatore:
Prof. Sonia Bergamaschi

Candidato:
Andrea Ricchi

Anno Accademico 2003–2004

Indice

Elenco delle figure

⋆⋆⋆

1

Introduzione

La gestione delle informazioni è la base per la crescita di qualsiasi tipo di azienda. Di fronte a una crescita veloce e costante della complessità dei workflow dei Business Process e a un aumento della domanda di servizi inerenti all'IT, le imprese operanti in questo campo hanno sentito la necessità di tecnologie e architetture che permettessero soluzioni portabili, affidabili, interoperabili, economiche e soprattutto flessibili all'evoluzione delle richieste del mercato.

Tre sono i principali strumenti che si sono o si stanno affermando per la realizzazione di questi obbiettivi:

SOA (Service Oriented Architecture): è una architettura orientata ai servizi. Nel nostro caso SOA viene implementata attraverso i Web Service che garantiscono il disaccoppiamento tra le risorse coinvolte, permettendo maggiore flessibilità e scalabilità. Il disaccoppiamento (Loose Coupling) si realizza attraverso sistemi di comunicazione interprocesso standard, che abbassano le cosiddette *"dipendenze artificiali"* esistenti tra due o più servizi che interagiscono tra loro.

Web Services: rappresentano un meccanismo interprocesso e interoperabile basato

su standard W3C. Nel nostro caso si sono affermati come la tecnologia principalmente utilizzata per l'implementazione di architetture SOA. Lo scambio dei dati avviene attraverso *XML* all'interno di messaggi *SOAP*, il cui recapito è affidato ai protocolli di trasporto internet (HTTP, FTP, SMTP, etc...). Un'ulteriore importante caratteristica dei Web Service è quella di possedere una grammatica XML che ne fornisce una descrizione astratta in termini di operazioni consentite, endpoint, tipo di dato, ...

BPEL4WS (Business Process Language for Web Service) o più semplicemente BPEL: nasce per sopperire alla mancanza di uno strumento efficace per la descrizione dei Business Process e per il coordinamento delle transazioni che avvengono all'interno di essi. Le funzionalità principali di BPEL risiedono nella:

- *Choreography*: ovvero la descrizione (operazioni consentite, porte per l'invio e la ricezione dei messaggi,...) di tutti i servizi che andranno a comporre il Business Process;
- *Orchestration*: la capacità di coordinare l'ordine di esecuzione dei servizi coinvolti nel processo business e le transazioni compiute tra i servizi stessi;

La storia di BPEL4WS è estremamente recente: il primo sviluppo risale al 2002, quando IBM, Microsoft e BEA convogliarono in questa specifica i punti di forza di *WSFL* (Web Service Execution Language, sviluppato da IBM e incentrato sulla Choreography) e di *XLANG* (Web Service for Business Process Design, sviluppato da Microsoft e dedicato all'Orchestration). Nel 2003 fu pubblicata la versione 1.1 di BPEL4WS tuttora in uso. Attualmente l'evoluzione della specifica è affidata all'*OASIS Web Service Business Process Execution Language (WSBPEL) Technical Commitee*.

Molti sono i software dedicati all'implementazione di Business Process attraverso BPEL. Per la realizzazione del progetto di tesi la scelta è ricaduta sui software forniti da Oracle dedicati. Nonostante si tratti di prodotti relativamente giovani (il server per i processi BPEL è uscito in Agosto 2004) si basano sull'esperienza acquisita in questo campo da Collaxa Inc., che fu una delle prime aziende a creare un motore (il Collaxa BPEL Server) esclusivamente dedicato all'esecuzione di processi BPEL. Il sever **BPEL Process Manager** di Oracle si avvantaggia, inoltre, di alcuni validi strumenti per lo sviluppo e il monitoraggio dei processi: **Oracle BPEL Designer** e **Oracle BPEL Console**.

1.1 Scopo della Tesi

Lo scopo della tesi è quello di valutare le potenzialità di BPEL4WS come strumento per la gestione dei servizi in una architettura di tipo SOA, e di stabilire pregi e difetti dell'ambiente di sviluppo fornito da Oracle. E' poi stata effettuata una analisi approfondita sulla specifica BPEL4WS e sull'architettura di BPEL Process Manager.

In particolare è stato descritto un processo interno a **Ebilling S.p.A.** come un insieme di risorse SOA e poi implementato attraverso la specifica BPEL4WS. Ebilling S.p.A. nata nel 2001 e leader in Italia nel campo dell'electronic billing, presente nei mercati banking, telecomunicazioni e public utilities.

1.2 Struttura della Tesi

La tesi si svolge su un totale di 6 capitoli.

Nei *capitoli 2 e 3* vengono trattati l'architettura SOA, i Web Services, e la

specifica BPEL4WS con particolare attenzione alla struttura delle activities preposte all'orchestration.

All'interno del **capitolo 4** vengono delineate le carateristiche proprie dell'ambiente Oracle.

Nel *quinto capitolo* viene descritta l'implementazione del processo interno a Ebilling S.p.A..

Il *sesto* e ultimo capitolo espone le conclusioni su BPEL e sull'ambiente di sviluppo fornito da Oracle.

2

Web Service & SOA

In questo capitolo verranno trattati in maniera più approfondita l'architettura SOA e la tecnologia dei Web Service.

2.1 SOA

SOA descrive un'architettura orientata ai servizi. Il problema principale nell'implementazione di questo stile architetturale è sempre stato quello di gestire l'integrazione tra i servizi che lo componevano. In passato, una prima soluzione, venne proposta tramite strumenti quali DCOM (Distributed Component Object Model) * o ORBs (Object Request Brokers) basato sulla specifica CORBA (Common Object Request Broker Architecture) †, che imponevano, però, grossi vincoli di interoperabilità in quanto l'implementazione del servizio era fortemente legata al tipo di dato. Con l'avvento della tecnologia dei Web Service è stato possibile

* Implementato per sistemi Windows utilizza le RPC (Remote Procedure Call) per la comunicazione tra componenti COM di una stessa rete.

† CORBA si pone l'obiettivo di fornire le regole per la realizzazione di una architettura a oggetti distribuiti. Al cuore di CORBA è presente un bus per la trasmissione dei messaggi che gli oggetti distribuiti si scambiano tra loro (ORBs: Object Request Brokers). La struttura dei messaggi è descritta all'interno del protocollo GIOP (General Inter-Orb Protocol). IIOP rappresenta una implementazione di GIOP che sfrutta TCP/IP per il trasporto e la consegna dei messaggi.

implementare SOA attraverso un sistema standard di descrizione dei servizi e di scambio delle informazioni che ha permesso di ottenere elevati livelli di flessibilità e scalabilità nella gestione delle risorse. In particolare è stato creato un sistema di tipizzazione dei dati comune a tutti i servizi e indipendente dal linguaggio utilizzato per la loro implementazione, che ha reso molto più semplice il dialogo tra il Service Consumer che effettua la richiesta per una operazione e il Service Provider che la evade. Inoltre le risorse sono state fornite di una interfaccia che ne permette una più efficace localizzazione e accessibilità.

Queste caratteristiche hanno permesso di realizzare gli obiettivi che si prefigge una architettura orientata ai servizi:

- **Disaccoppiamento**: ovvero rendere i servizi il più possibile autonomi l'uno dall'altro cercando di abbattere le dipendenze che esistono tra di essi. Queste dipendenze possono essere di due tipi:

    1. *Dipendenze Reali*: ovvero l'insieme delle funzionalità che dipendono da risorse esterne al servizio. Questo tipo di dipendenza esiste sempre e non può essere ridotta;
    2. *Dipendenze Artificiali*: sono le caratteristiche che un servizio deve possedere per poter interagire con un altro sistema o un'altra risorsa network. Tipiche dipendenze artificiali sono quelle che derivano dai linguaggi di implementazione o dal tipo di piattaforma utilizzata. Sulle dipendenze artificiali si può agire per aumentare il livello di disaccoppiamento tra i servizi;

- **Condivisione dei tipi di dato e delle operazioni offerte**: i servizi che compongono l'architettura SOA devono rendere publiche le proprie funzionalità e devono possedere un sistema di scambio delle informazioni comune e che

astrae i tipi di dato interni al servizio. I dati dovranno essere correlati solamente al tipo di operazione a cui sono destinati e il loro tipo di trasmissione dovrà garantire univocità e stabilità di comportamento.

2.2 Web Service

I Web Service sono un meccanismo di comunicazione interprocesso e interoperabile basato su standard W3C, il cui scopo è quello di offrire le proprie funzionalità in modo indipendente dal protocollo, dalla piattaforma e dall'architettura di implementazione. Questa tecnologia ci permettere di realizzare il disaccoppiamento tra le risorse SOA tramite un set di standard che ne descrivono la struttura. In particolare vengono regolati il sistema di comunicazione e di trasporto delle informazioni (SOAP su HTTP, SMTP,...), l'interfacciamento con le altre risorse (WSDL)e il meccanismo di pubblicazione e di ricerca del servizio (UDDI). A questi, negli ultimi anni, si è aggiunta la necessità di uno strumento che garantisse un'orchestration efficace dei Web Service, ovvero una specifica che permettesse di assemblarli in un nuovo processo. L'insieme di queste specifiche rappresenta il Web Service Protocol rappresentato in figura 2.1.

Figura 2.1: Web Service Protocol Stack

Trasporto: i protocolli internet di trasporto forniscono il supporto per lo scambio di messaggi sincroni e asincroni e sono, quindi, alla base del sistema di comunicazione dei Web Service. I protocolli principalmente utilizzati sono:

- *SMTP*: solo per conversazioni asincrone in cui il messaggio SOAP è allegato o come testo della E-Mail o come attachment in formato MIME (Multipurpose Internet Mail Extension);
- *HTTP/HTTPS*: permettono sia conversazioni sincrone che asincrone, sono questi i protocolli che si stanno affermando come lo standard de-facto per la comunicazione tra i Web Service;
- *IIOP*: è l'acronimo per Internet Inter-Orb Protocol e permette ai Web Service di supportare anche le architetture CORBA (Common Object Request Broker Architecture);

Comunicazione: i Web Service inviano e ricevono informazioni tramite messaggi che seguono le specifiche di SOAP (Simple Object Access Protocol) uno standard W3C basato su XML. La struttura dei dati che vengono scambiati è rappresentata attraverso schemi XSD (XML Schema Definition) *. Un messaggio che segue le norme imposte da SOAP presenta un meccanismo di incapsulamento dei dati costituito da una parte principale che descrive come processare la struttura del messaggio SOAP e da una o più parti secondarie, in relazione con la prima, che contengono una risorsa identificata tramite una URI. Il messaggio SOAP può poi contenere informazioni su come deve avvenire la consegna del messaggio, ad esempio se utilizzare o meno il modello RPC oppure specificare il protocollo per il trasporto del messaggio.

Di seguito è riportato un esempio di un messaggio SOAP:

* E' il modello consigliato dal consorzio W3C per la rappresentazione dei dati tramite schemi XML.

```
<SOAP:Header>
</SOAP:Header>

<?xml version = '1.0' encoding = 'UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
  <SOAP-ENV:Body>
    <soap:Operazione xmlns:prefix="http://namespace/">
      <prefix:Informazioni>
        <prefix:Dati>12-3456-789-0</prefix:Dati>
      </prefix:Informazioni>
    </soap:Operazione>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Da quanto emerge sono due le parti fondamentali del messaggio:

1. *Header*: è la parte facoltativa che può contenere parametri che specificano meglio il messaggio SOAP (un esempio potrebbe essere la data di invio);
2. *Envelope*: è un nodo obbligatorio che racchiude il messaggio SOAP. Oltre a contenere i riferimenti ai namespace necessari alla descrizione del messaggio SOAP racchiude anche il nodo Body del messaggio costituito dai parametri da inviare all'operazione del servizio. La parte del Body, nel caso di un messaggio di risposta, può essere sostituita dal nodo Fault che descrive gli eventuali errori della fase di elaborazione.

Descrizione: WSDL (Web Service Description Language) è lo standard W3C

utilizzato per definire le funzionalità del Web Service. In altre parole si fornisce una descrizione astratta del servizio che riguarda le possibili interazioni (operazioni) con le altre risorse. L'elemento cardine di un file WSDL è rappresentato dal Port Type che associa un'operazione ai rispettivi messaggi di entrata e di uscita. All'interno del WSDL vengono, inoltre, definiti i tipi di dato utilizzati dal Web Service e l'endpoint del servizio. L'interfaccia WSDL è fondamentale per la fase di ricerca, infatti è attraverso questo file che avviene la pubblicazione del Web Service nel service registry.

Pubblicazione e Ricerca: per rendere accessibili le funzionalità del Web Service a tutte le risorse della rete è necessario che la sua interfaccia WSDL venga pubblicata. La pubblicazione può avvenire in due modi differenti:

- *Diretta*: quando il Service Requester recupera il file WSDL del Web Service direttamente dal Service Provider;
- *Indiretta*: quando la pubblicazione avviene all'interno di un archivio di Web Service;

In particolare nel caso di una pubblicazione indiretta, l'archivio all'interno del quale sarà depositata l'interfaccia WSDL è organizzato secondo le norme definite dalla specifica UDDI (Universal Description, Discovery and Integration), lo standard che regola la pubblicazione e la ricerca del servizio.

Orchestrazione: è il sistema che permette di assemblare ordinatamente più servizi allo scopo di creare una nuova applicazione che rispetti i canoni dei Web Service finora elencati. Ancora non abbiamo visto imporsi uno standard per regolare questa attività, tuttavia al momento, la specifica più utilizzata risulta essere quella di BPEL4WS. Tra gli altri ricordiamo WS-CI (Web Service Choreography Interface, sviluppato dal W3C), EbXML e RosettaNET.

3

La Specifica BPEL4WS

BPEL4WS è l'acronimo di Business Process Execution Language for Web Service ed è una specifica nata per regolare l'orchestrazione dei Web Service. Questo standard offre, infatti, un linguaggio per gestire non solo la sequenza logica del workflow del processo, ma anche lo scambio di messaggi sincroni e asincroni con servizi remoti e il verificarsi di errori ed eccezioni. Un file sorgente BPEL è composto da tre parti:

1. Una prima parte in cui sono descritti tutti i servizi (partner link) partecipanti al workflow del processo;
2. Una sezione che elenca le variabili del processo;
3. Un'ultima parte che comprende l'insieme delle istruzioni che implementano l'orchestrazione dei servizi;

Va poi ricordato che un processo BPEL è a tutti gli effetti un Web Service e come tale si basa sugli standard caratteristici dei Web Service.

Passiamo ora a una descrizione delle istruzioni e dei sistemi di gestione delle informazioni fornite dalla specifica BPEL4WS.

3.1 BPEL4WS & XML

L'orchestrazione dei servizi è completamente basata sullo standard XML, sia per quanto riguarda la struttura delle istruzioni per la coordinazione dei partner link, sia per quanto riguarda la comunicazione con questi ultimi. Per una migliore gestione delle informazioni la specifica BPEL4WS utilizza XPath [10] come strumento per operare interrogazioni sui documenti XML. Vediamo ora come XPath è stato integrato all'interno della specifica BPEL4WS.

Interrogazioni: XPath ci offre la possibilità di accedere al valore di un nodo all'interno di un documento XML. Questa funzionalità è sfruttata da BPEL nelle strutture $< from >$ e $< to >$:

```
<from variable="nomeVariabile" part="partName"
query="/XPathQuery">
```

L'attributo query fa riferimento a un nodo assoluto contenuto nello schema XML che descrive la variabile in questione.

Espressioni: oltre alle funzioni classiche che XPath ci mette a dispozione per operare su stringhe, valori numerici e date, la specifica BPEL ne introduce di nuove. Queste estensioni sono definite all'interno del namespace http://schemas.xmlsoap.org/ws/2003/03/business-process/ e precedute dal prefisso bpws.

```
<from expression= "bpws:getVariableData
('nomeVariabile', 'partName', '/XPathQuery') + 1"/>
```

3.2 Istruzioni Primitive

La specifica BPEL4WS ci mette offre una serie di istruzioni (activity) che implementano le funzionalità di base necessarie all'orchestrazione. Queste riguardano in particolare l'inizializzazione delle variabili, l'interazione con i partner link e la creazione dei casi di errore.

3.2.1 Gestione dei Dati

L'elemento che ci permette di assegnare i valori alle variabili, sfruttando le propietà di XPath, è il costrutto $< assign >$. Di seguito è riportato un esempio della sua struttura.

```
<assign>
  <copy>
    <from expression="concat('Hello ',
    bpws:getVariableData('input','payload','/tns:name'))"/>
    <to variable="output" part="payload" query="/result"/>
  </copy>
</assign>
```

3.2.2 Interazione con i Partner Link

Le attività primitive che ci consentono lo scambio di messaggi con i partner link sono tre. Ognuna di queste presenta una variabile di input o di output dichiarata nella sezione $< variables >$.

Invoke: è l'activity attraverso la quale possiamo invocare un servizio. Oltre alla variabile di input che contiene il documento XML con le informazioni da inviare al partner link, vengono specificati il nome di quest'ultimo, il portType e il tipo di operazione da compiere sui dati inviati. Nel caso in

cui sia abbia una conversazione sincrona con il servizio invocato allora sarà specificata anche una variabile di output che conterrà lo schema XML del messaggio di risposta.

```
<invoke name="invoke" partnerLink="LoanService"
  portType="services:LoanService"
  operation="initiate"
  inputVariable="request"
  outputVariable="onResult"/>
```

Receive: questa istruzione è utilizzata per ricevere messaggi di risposta asincroni dai servizi che andiamo a invocare oppure per inizializzare una nuova istanza di un processo BPEL.

```
<receive name="receive_invoke"
   partnerLink="LoanService"
   portType="services:LoanServiceCallback"
   operation="onResult"
   variable="response"/>
```

Reply: è il costrutto per inviare un messaggio di risposta a un partner link.

```
<reply partnerLink="purchasing
       portType="lns:purchaseOrderPT"
       operation="sendPurchaseOrder"
       variable="POFault"
       faultName="cannotCompleteOrder"/>
```

3.2.3 Segnalazione dei Casi di Errore

Attraverso il costrutto $< throw >$ abbiamo la possibilità di segnalare un errore che si verifica durante l'esecuzione del processo BPEL. L'elemento $< throw >$ è identificato da un nome e può possedere una variabile che fornisce ulteriori informazioni sul tipo di errore occorso.

```
<throw faultName="nomeErrore"
 faultVariable="nomeVariabile" />
```

3.2.4 Wait, Terminate, Empty

Questi sono i costrutti che ci consentono rispettivamente di attendere un certo periodo prima di continuare con il flusso delle istruzioni, di terminare il processo senza che vengano creati messaggi di errore e di non eseguire nulla *

```
<wait (for="espressione" | until="espressione") />
```

3.3 Istruzioni Strutturate

L'insieme di queste activity costituisce il cuore delle funzionalità della specifica BPEL. Attraverso di esse possiamo gestire la logica del workflow del Business Process, gli errori le eccezioni † e gli eventi ‡.

3.3.1 Sequence

Questa activity contiene una serie di istruzioni che sono eseguite sequenzialmente secondo l'ordine in cui sono state inserite all'interno dell'elemento $< sequence >$.

* BPEL prevede questa struttura nel caso in cui non sia necessario eseguire nessuna operazione per il verificarsi di un determinato evento. Il costrutto $< empty >$ diventa molto utile nella sincronizzazione del flusso delle istruzioni.

† Si distinguono dagli errori in quanto riguardano le transazioni che danno un risultato diverso da quello previsto. Ad esempio una richiesta di prenotazione per un volo aereo che non viene approvata.

‡ Riguarda la gestione delle istruzioni che devono essere eseguite all'arrivo di un determinato tipo di messaggio.

Questa istruzione termina quando viene eseguito l'ultimo elemento contenuto in essa.

```
<sequence>
...
Activity
...
</sequence>
```

3.3.2 Switch

L'elemento $< switch >$ ci consente di stabilire delle condizioni per l'esecuzione delle istruzioni.

```
<switch>
  <case condition="espressione Booleana">
    <sequence>
    ...
    Activity
    ...
    </sequence>
  </case>
  <case condition="espressione Booleana">
    <sequence>
    ...
    Activity
    ...
    </sequence>
  </case>
```

```
    ...
  <otherwise>
    <sequence>
    ...
    Activity
    ...
    </sequence>
  </otherwise>
</switch>
```

3.3.3 While

Questa è l'unica istruzione all'interno della specifica BPEL che ci permette di implementare dei cicli.

```
<while condition="espressione Booleana">
  <sequence>
  ...
  Activity
  ...
  </sequence>
</while>
```

3.3.4 Flussi di istruzioni paralleli

Lo standard BPEL ci offre la possibilità di eseguire sequenze di istruzioni in parallelo. Ci viene poi fornito l'elemento $< links >$ che ci permette di sincronizzare l'esecuzione di questi flussi di istruzioni.

```
<flow>
  <links>
```

```
    <link name="XtoY"/>
    <link name="CtoD"/>
  </links>
  <sequence name="X">
    <source linkName="XtoY"/>
    <invoke name="A" .../>
    <invoke name="B" .../>
  </sequence>
  <sequence name"Y">
    <target linkName="XtoY"/>
    <receive name="C" ...>
    <source linkName="CtoD"/>
    </receive>
    <invoke name="E" .../>
  </sequence>
    <invoke partnerLink="D" ...>
      <target linkName="CtoD"/>
    </invoke>
</flow>
```

Attraverso l'elemento $< links >$ possiamo stabilire che una sequenza del $< flow >$ venga eseguita solo al completamento di un'altra sequenza. Nel nostro esempio la sequenza Y verrà eseguita soltanto al termine delle istruzioni contenute nella sequenza X. Allo stesso modo la sequenza D dovrà attendere il completamento delle attività contenute in Y.

3.3.5 Pick

L'elemento $< pick >$ attende il verificarsi di un evento e una volta che questo si verifica esegue le istruzioni che gli sono associate. Questo costrutto può anche essere utilizzato per la creazione dell'istanza di un processo BPEL.

```
<pick createInstance="yes/no" >
  <onMessage partnerLink="buyer"
             portType="orderEntry"
             operation="inputLineItem"
             variable="lineItem">
    <sequence>
    ...
    Activity
    ...
    </sequence>
  </onMessage>
  <onMessage partnerLink="buyer"
             portType="orderEntry"
             operation="orderComplete"
             variable="completionDetail">
    <sequence>
    ...
    Activity
    ...
    </sequence>
  </onMessage>

  <onAlarm for="'P3DT10H'">
```

```
    <sequence>
    ...
    Activity
    ...
    </sequence>
  </onAlarm>
</pick>
```

Ogni elemento $< onMessage >$ è preposto all'intercettazione di un diverso evento, ma la funzione $< pick >$ termina una volta eseguite le istruzioni relative alla prima occorrenza di uno di questi. L'elemento $< onAlarm >$ permette di stabilire un tempo massimo per la validità del costrutto $< pick >$.

3.3.6 Scope

Lo $< scope >$ fornisce l'ambiente di background durante l'esecuzione delle istruzioni in esso contenute. Al suo interno possono essere dichiarate nuove variabili e sono presenti le strutture per la gestione di errori ed eccezioni e per la definizione degli eventi (messaggi e tempi limite per l'esecuzione dello $< scope >$) e dei $< correlationSet >$.

```
<scope>
  <variables>
  ...
  </variables>
  <correlationSets>
  ...
  </correlationSets>
  <faultHandlers>
```

```
  ...
  </faultHandlers>
  <compensationHandler>
  ...
  </compensationHandler>
  <eventHandlers>
  ...
  </eventHandlers>
  <sequence>
  ...
  Activity
  ...
  </sequence>
</scope>
```

Particolare attenzione meritano gli strumenti per la gestione di errori ed eventi:

1. **Fault Handlers**: entrano in funzione nel caso in cui si verifichi un errore che interrompe l'esecuzione della sequenza di istruzioni contenuta all'interno dello $< scope >$. Attraverso l'elemento $< catch >$ possiamo intercettare un determinato tipo di errore segnalato attraverso il costrutto $< throw >$ e caratterizzato da un nome e in generale da una variabile. L'elemento $< catchAll >$ ci permette,invece, di intercettare qualsiasi tipo di errore, senza la neccessità che questo sia stato descritto all'interno di $< throw >$. Se per uno $< scope >$ non sono stati previsti dei $< faultHandlers >$ allora la gestione dell'errore viene affidata allo $< scope >$ padre * e così via fino alla radice. Se, a questo punto, l'errore non è ancora stato intercettato allora abbiamo il verificarsi di

* Gli $< scope >$ possono essere innestati uno nell'altro.

una terminazione anomala del processo BPEL. Molto importante è il fatto che noi potremo inserire istruzioni all'interno degli elementi $< catch >$ e $< catchAll >$ da eseguire una volta che l'errore sia intercettato.

```
<faultHandlers>
  <catch faultName="nomeErrore"
         faultVariable="nomeVariabile">
    <sequence>
    ...
    Activity
    ...
    </sequence>
  </catch>
  <catchAll>
    <sequence>
    ...
    Activity
    ...
    </sequence>
  </catchAll>
</faultHandlers>
```

2. **Compensation Handler**: la funzione di compensanzione può essere invocata solo nel caso in cui l'elemento $< scope >$ termini con successo e ci permette di annullare le transazioni avvenute durante la sua esecuzione se una operazione non restituisce la risposta desiderata. Prendiamo per esempio un processo BPEL il cui scopo sia quello di prenotare un hotel, un'auto e un volo per l'organizzazione di un viaggio. Se le operazioni per l'hotel

e l'auto vanno a buon fine, ma venga rifiutata la prenotazione per il volo aereo allora attraverso il $< compensationHandler >$ possiamo eseguire le istruzioni per annullare le transazioni precedenti.

```
<scope name="assessor-scope">
  <target linkName="receive-to-assess"/>
  <compensationHandler>
  ...
  Activity
  ...
  </compensationHandler>
  <invoke name="invokeAssessor"
          partner="assessor"
          portType="asns:riskAssessmentPT"
          operation="check"
          inputContainer="request"
          outputContainer="riskAssessment">
    <source linkName="assess-to-setMessage"
            transitionCondition=
              "bpws:getContainerData
                ('riskAssessment', 'risk')='low'"/>
    <source linkName="assess-to-approval"
            transitionCondition=
              "bpws:getContainerData
                ('riskAssessment', 'risk')!='low'"/>
  </invoke>
</scope>
```

In questo esempio se la condizione per la transazione non viene soddisfatta allora viene invocato il relativo $< compensationHandler >$.

Il $< compensationHandler >$ può anche essere invocato esplicitamente tramite l'elemento $< compensate >$, specificando il nome dello $< scope >$ per cui è necessaria la compensazione. Questo sistema risulta utile all'interno di un $< faultHandlers >$ nel caso si verifichi un errore che obblighi a compensare le istruzioni di uno $< scope >$ precedente.

```
<compensate scope="NomeScopeDaCompensare"/>
```

3.4 Correlation Set

I $< correlationSet >$ sono una particolare struttura per correlare un messaggio alla rispettiva istanza e porta del processo BPEL a cui è destinato. I correlation set sono dichiarati a livello globale del processo oppure all'interno di uno $< scope >$ e possono essere utilizzati assieme agli elementi $< invoke >$, $< receive >$, $< reply >$, negli $< eventHandlers >$ e nel costrutto $< pick >$. Nell'interfaccia WSDL del processo BPEL viene descritta la relazione tra il correlation set e la porta e il messaggio a cui si riferisce.

```
<receive partnerLink="Buyer"
         portType="SP:PurchasingPT"
         operation="AsyncPurchase"
         variable="PO">
  <correlations>
    <correlation set="PurchaseOrder" initiate="yes">
  </correlations>
</receive>
```

```
<invoke partnerLink="Buyer"
        portType="SP:BuyerPT"
        operation="AsyncPurchaseResponse"
        inputVariable="POResponse">
  <correlations>
    <correlation set="PurchaseOrder" initiate="no"
                 pattern="out">
    <correlation set="Invoice" initiate="yes"
                 pattern="out">
  </correlations>
</invoke>
```

All'invocazione il costrutto $< correlation >$ presenta alcuni attributi che definiscono la correlazione. Abbiamo ad esempio l'attributo *set* che ne stabilisce il tipo, *pattern* nel caso dell'$< invoke >$ per decidere se la correlazione si rferisce a un messaggio di input o di output e l'attributo *initiate* per inizializzarla.

⋆⋆⋆

4

L'ambiente Oracle

Nel giugno 2004 Oracle acquisisce Collaxa Inc., un'azienda che da tempo operava nell'ambito dei processi BPEL e che aveva realizzato il primo server completamente dedicato alla loro esecuzione. Nell'agosto 2004 Oracle, basandosi sull'esperienza di Collaxa Inc., effetua il primo rilascio dei software per realizzare workflow di Business Process utilizzando la specifica BPEL. L'ambiente Oracle comprende due software:

- **Oracle BPEL Process Manager**: comprende *BPEL server* per l'esecuzione e la pubblicazione dei processi BPEL e *BPEL Console* per il loro monitoraggio e l'amministrazione del BPEL Server;
- **Oracle BPEL Designer**: è l'editor grafico per l'implementazione dei workflow dei Business Process;

A questi si devono aggiungere un application server J2EE e un database come supporto alle funzionalità del BPEL Server.

In figura 4.1 sono mostrate l'architettura dell'ambiente Oracle e le caratteristiche, delle sue componenti.

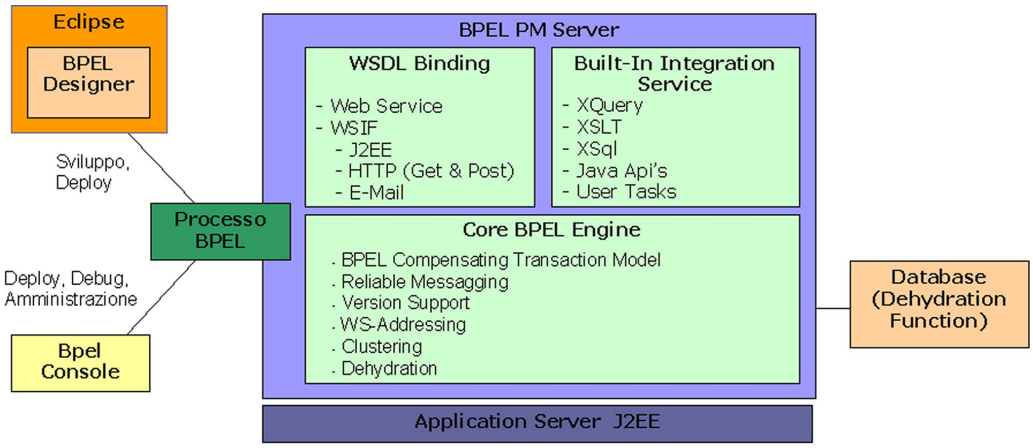

Figura 4.1: Architettura dell'ambiente Oracle

Prima di procedere con l'analisi dei software forniti da Oracle è conveniente spiegare come è strutturato e come avviene il deploy * di un processo BPEL all'interno di questo ambiente.

4.1 Realizzazione di un processo BPEL

Il workflow del Business Process è implementato tramite *BPEL Designer* che viene fornito come plug-in della piattaforma open source Eclipse. Durante la fase di sviluppo un progetto BPEL è costituito da:

- Un project file per la gestione del progetto Eclipse;
- Un file per la descrizione del processo BPEL (bpel.xml) in cui vengono elencati il nome del file sorgente BPEL, l'identificativo (ID) del processo, la locazione di tutti i WSDL dei servizi orchestrati e alcune configurazioni opzionali come ad esempio eventuali input di default per i partner link o per il processo BPEL stesso;

* Riguarda la creazione dell'archivio *.JAR* necessario a BPEL Server per l'esecuzione e la pubblicazione del processo BPEL

- Abbiamo poi un file utilizzato per la compilazione e il deploy del processo (build.xml);

```
<?xml version=1.0"?>
<project name="TravelProcess"
         default="main"
         basedir=".">
  <property name="deploy" value="default">
  <property name="rev" value="1.0">
  ...
</project>
```

- Il file sorgente BPEL e la relativa interfaccia WSDL;

Una volta completato lo sviluppo del workflow tramite BPEL Designer, può avvenire il deploy del processo BPEL in uno dei domini in cui è partizionato BPEL Server. I domini sono una suddivisione logica del server che permettono allo sviluppatore una ordinazione più efficiente dei processi BPEL attivi; possiamo, ad esempio, creare un dominio per i processi di test e uno per i processi operativi. Una volta installato BPEL Server viene creato il dominio default, ma a questo possono venirne aggiunti altri attraverso BPEL Console.

Il deploy del processo BPEL può avvenire utilizzando BPEL Console o BPEL Designer, oppure da linea di comando. In quest'ultimo caso i comandi devono essere lanciati all'interno della cartella contenente il progetto BPEL. Oracle ne mette a dispozione due:

- **BPELC**: è il BPEL Compiler che, in riferimento al file bpel.xml, compila il progetto BPEL sviluppato tramite il BPEL Designer e produce l'archivio

JAR del processo BPEL. Per rendere effettivo il deploy, l'archivio dovrà essere copiato all'interno della cartella . . . \orabpel\domains\nomeDominio\deploy\. BPELC presenta anche alcuni comandi opzionali tra i quali quelli per specificare la versione del processo BPEL (-rev) o per effettuare automaticamente il deploy all'interno di un dominio (-deploy).

```
>bpelc -rev 1.0 -deploy default
```

- **Obant**: questo comando è basato sullo strumento di compilazione Ant di Apache e sfrutta il file build.xml per effettuare il deploy del processo. Per lanciarlo sarà sufficiente eseguire obant dalla linea di comando all'interno della cartella contenente il progetto Eclipse del processo BPEL.

4.2 Oracle BPEL Process Manager 2.0.11

Questo software fornisce gli strumenti necessari all'esecuzione e all'amministrazione dei processi BPEL e offre alcuni servizi che potenziano le caratteristiche della specifica BPEL4WS.

4.2.1 BPEL Server

BPEL Server fornisce l'ambiente in cui avviene il deploy dei processi BPEL e offre il supporto per la loro esecuzione. Come si può notare dalla figura 4.1, è costitutito da tre parti:

- **Core BPEL Engine**;
- **WSDL Bindings**;
- **Integration Services**;

Core BPEL Engine

Il Core BPEL Engine è l'ambiente per l'esecuzione e il deploy dei processi BPEL. Oltre alla specifica BPEL4WS, supporta il *WS-Addressing* che è il protocollo per l'indirizzamento e l'instradamento dei mesaggi SOAP e il *BPEL Compensating Transaction Model* attraverso il quale viene garantito che le transazioni seguano il modello *ACID*, acronimo per le quattro caratteristiche che descrivono una transazione:

- **Atomicità**: la transazione deve concludersi con successo per tutte le informazioni coinvolte, altrimenti deve essere ripristinato lo stato iniziale;
- **Consistenza**: la transazione deve creare le nuove informazioni in modo consistente con l'ambito in cui si trovano (in database non devono, per esempio, essere violate le chiavi primarie);
- **Isolamento**: ogni transazione deve essere portata a termine indipendentemente dalle altre;
- **Durabilità**: una volta compiuta una transazione i suoi effetti devono essere garantiti nel tempo;

BPEL server offre, inoltre, supporto per il deploy di versioni diverse di uno stesso Business Process e per il *clustering* del server. La caratteristica più importante rimane, però, la *dehydration function* che riduce drasticamente la richiesta di risorse hardware durante l'esecuzione dei processi BPEL: quando viene effettuata una invocazione asincrona a un partner link, lo stato del Business Process viene salvato nel database di supporto a BPEL Server e, una volta ricevuto il messaggio di risposta, viene ricaricato continuando l'esecuzione del workflow del processo.

WSDL Bindings

Il WSDL Bindings è il sistema responsabile della comunicazione tra il processo BPEL e i servizi orchestrati. A differenza della specifica BPEL che parla esplicitamente di orchestrazione di Web Service, BPEL Server consente di integrare nel workflow di processo risorse che non rispondono ai protocolli utilizzati dai Web Service. Questo avviene tramite *WSIF (Web Service Integration Framework)* [9], una tecnologia che deriva da Apache e originariamente sviluppata da IBM alphaWorks che estende, nel nostro caso, il modello dei Web Service verso le seguenti risorse:

- **Risorse HTTP (GET e POST)**;
- **Classi Java**;
- **EJB** (Enterprise Java Beans): sono la componente lato server dell'architettura J2EE e permettono lo sviluppo di applicazioni basate sulla tecnologia Java;
- **JCA** (Java Connector Architecture): è la tecnologia Java che permette la connessione dell'application server J2EE verso altri server o altre risorse (ad esempio un database);
- **Code JMS** (Java Messagge Service): è il servizio di messaggistica per la comunicazione tra le componenti dell'architettura J2EE;
- **Apache AXIS** [9]: è un ambiente ideato da Apache per lo sviluppo di Web Service;

Integration Service

Oracle utilizza all'interno di BPEL Server alcune funzioni per potenziare lo sviluppo e la gestione dei processi BPEL.

Manipolazione delle informazioni: Oracle, oltre a XPath, supporta altre tecnologie e introduce nuove funzioni per il trattamento delle informazioni contenute nei documenti XML scambiati tra il processo BPEL e i partner link. Tra queste tecnologie abbiamo *XSLT* (Extensible Stylesheet Language for Transformation) [13] che fornisce supporto per la trasformazione del documento XML in altri formati come ad esempio HTML. La funzione per attivare il motore XSLT prevede due parametri: il modello XSLT per la trasformazione e il documento XML su cui va compiuta l'operazione.

```
ora:processXSLT('modelloXSLT','documentoXML');
```

Sono utilizzati anche XQuery [12] e XSQL [11] come linguaggi di interrogazione più avanzati di XPath.

```
ora:processXQuery();
ora:processXSQL();
```

Oracle ha introdotto le seguenti nuove funzioni:

ora:getElement('nomeVar', 'partName', 'queryPath', index);	Per accedere ai valori in un array
ora:countNodes('nomeVar', 'partName', 'queryPath');	Per stabilire la grandezza di un array
ora:addChildNode('nomeArr', 'nuovoVal');	Per aggiungere un valore a un array
ora:mergeChildNode('nomeArr', 'nomeArr');	Per effettuare il merge tra due array
ora:parseEscapedXML(string);	Trasforma una stringa in un documento XML
ora:getContentAsString(ElementoXML);	L'operazione inversa della funzione vista sopra
ora:setNodeValue('nomeVar', 'partName', 'queryPath', 'Val');	Stabilisce un nuovo valore per un nodo
ora:getNodeValue(nodo);	Accede al valore di un nodo
ora:integer(node);	Restituisce il valore di un nodo come integer
ora:addQuotes(string);	Aggiunge gli apici agli estremi della stringa
ora:readFile('nomeFile');	Utilizzata per leggere il contenuto di un file
ora:getCurrentDate();	Restituisce la data corrente
ora:getCurrentTime();	Restituisce l'ora corrente
ora:getCurrentDateTime();	Restituisce ora e data correnti
ora:formatDate('dateTime', 'formato');	Dà in uscita data e ora nel formato richiesto
ora:getProcessId();	Ritorna l'ID del processo BPEL
ora:getProcessURL();	Ritorna l'URL del processo BPEL
ora:getInstanceId();	Ritorna l'ID dell'istanza del processo BPEL
ora:getConversationId();	L'ID che identifica una conversazione asincrona
ora:getCreator();	Identifica chi ha creato l'istanza
ora:generateGUID();	Genera un GUID (Globally Unique ID)

Built-In Service: Oracle ha introdotto alcuni servizi per integrare i server di posta e le code JMS e per l'interazione di un utente umano con il processo BPEL. Questi servizi si comportano come un qualsiasi Web Service e possono essere orchestrati allo stesso modo dei normali partner link.

Il **servizio per le E-Mail** presenta un file XML in cui sono configurati i server di posta IMAP, POP e SMTP da lui utilizzati. Le E-Mail possono essere ricevute tramite l'operazione *onMessage* dopo aver eseguito una sottoscrizione al server di posta utilizzato (*subscribe*, *unsubscribe*). L'operazione *sendMessage* permette invece l'invio delle E-Mail. In figura 4.2 è descritta la struttura di questo servizio.

Molto simile è il funzionamento del servizio per le **code JMS** che presenta le stesse operazioni per spedire e ricevere i messaggi. In questo caso la coda

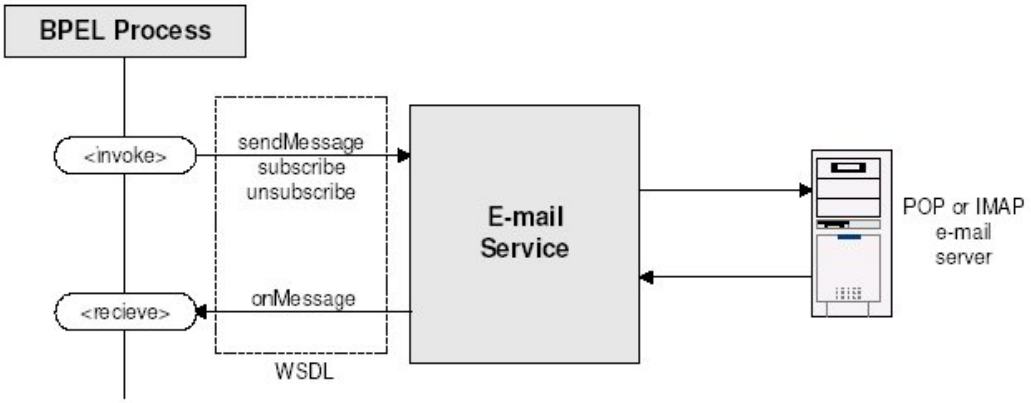

Figura 4.2: Servizio E-Mail

JMS a cui farà riferimento il servizio è descritta all'interno del file WSDL che lo interfaccia al processo BPEL.

Discorso a parte merita il **Task Manager** utilizzato per l'interazione di un utente umano con il processo BPEL, funzionalità non contemplata dalla specifica BPEL4WS, ma molto importante ai fini di un Business Process. Attraverso questo servizio un utente può effettuare delle scelte o confermare transazioni, ma non può modificare il workflow del processo mentre questo è in esecuzione. Il Task Manager è provvisto di due interfacce: una verso il processo BPEL e l'altra verso l'utente.

- L'interfaccia WSDL verso il processo BPEL permette a quest'ultimo l'invocazione del servizio (initiateTask) oppure di completare o aggiornare una sua istanza esistente (updateTask, completeTask). Il Task Manager fornisce una risposta al processo sia quando l'interazione con l'utente termina (onTaskResult) sia quando scade il periodo di attività del servizio (onTaskExpired).
- L'interfaccia utente può essere implementata sia come Java API sia

come WSDL ed elenca tutte le operazioni consentite all'utente per l'interazione con il processo BPEL.

Nello schema che segue è riportata l'architettura di questo servizio.

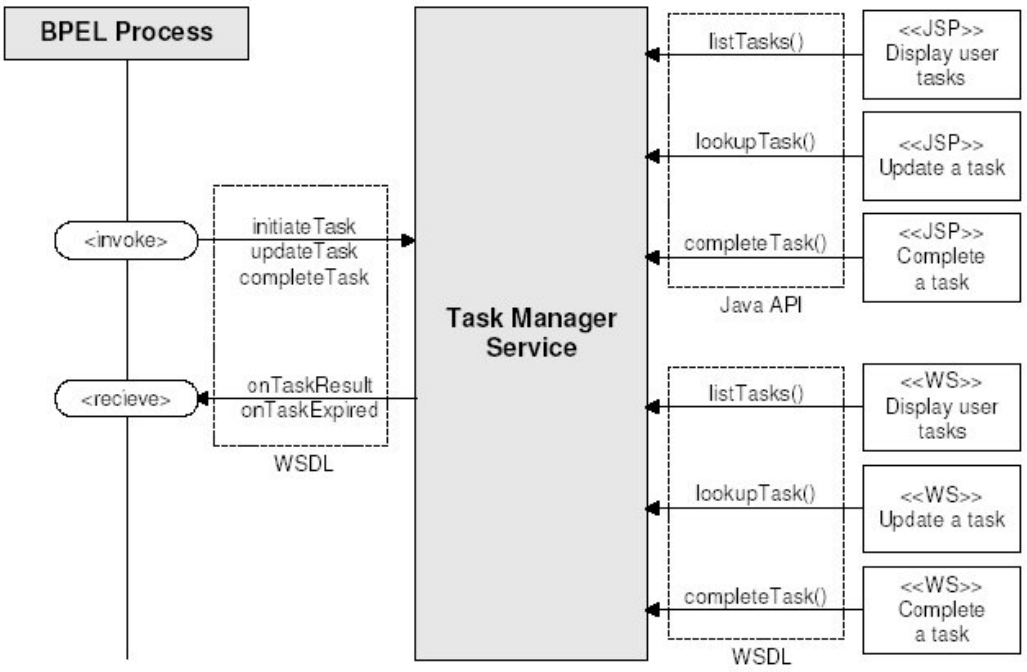

Figura 4.3: Servizio Task Manager

Connector: sfruttando la tecnologia JCA propria dell'architettura dell'application server J2EE, Oracle ha creato una serie di connettori per permettere una più semplice interazione tra il processo BPEL e file system, database, code JMS e code Oracle AQ *.

Java Embedded: Oracle ha aggiunto un nuovo elemento alla specifica BPEL4WS che consente di inserire codice Java all'interno del file sorgente BPEL del processo.

* Oracle Advanced Queue: sistema di code di messaggi proprio del database Oracle 9i [8]

```
<bpelx:exec name="InvokeJavaExec"
      language="Java"
            version="1.4">
  <[!CDATA[
  ...
  Codice
  ...
  ]]>
</bpelx:exec>
```

Questa activity presenta anche i seguenti attributi:

- *import*: per importare package Java;

  ```
  <bpelx:exec import="org.w3c.dom.Element">
  ```

- *language*: per stabilire il linguaggio compreso all'interno del codice BPEL. Attualmente è previsto supporto solo per Java, ma in futuro sarà aggiunto anche C#;
- *version*: denota la versione del linguaggio utilizzato;

Oracle ha introdotto anche numerosi metodi utilizzabili all'interno dell'elemento $< bpelx : exec >$ che migliorano l'interazione del codice Java con il resto del workflow del processo. Abbiamo poi il comando schemac, eseguibile dalla linea di comando, che ci consente la creazione di una classe Java partendo dallo schema XML di una variabile del processo BPEL.

BPEL Server API: Oracle ha provveduto una serie di interfacce Java API per permettere all'amministratore di accedere alle funzionalità del BPEL Server. Queste interfacce, utilizzate anche dalla BPEL Console, sono completamente personalizzabili e fanno riferimento ai seguenti package:

- *com.oracle.services.bpel.task*: utilizzata per le User Tasks;
- *com.oracle.bpel.client*: permette di accedere alle funzionalità del server;
- *com.oracle.bpel.client.auth*: contiene le classi per autorizzare l'accesso ai domini;
- *com.oracle.bpel.client.dispatch*: contiene le classi utilizzate nell'invocazione di un processo e nella creazione dell'istanza;
- *com.oracle.bpel.client.util*: classi per l'interazione con HTML e SQL;
- *com.collaxa.xml*: contiene le funzioni per l'elaborazione dei dati XML;
- *com.collaxa.common.util*: contiene le funzionalità per la generazione delle statistiche sui processi in esecuzione;

4.2.2 BPEL Console

BPEL Console è lo strumento incaricato per il deploy, il debug e la gestione dei processi BPEL e per l'amministrazione di BPEL Server. L'interfaccia utente è costituita da pagine JSP di cui è fornito il codice sorgente e quindi completamente personalizzabili. L'amministratore può accedere a BPEL Console tramite un web browser e può collegarsi o ai domini in cui avviene il deploy dei processi oppure alla parte riservata alla configurazione di BPEL Server. In entrambi i casi sarà necessario fornire credenziali per effettuare il login.

Amministrazione dei processi BPEL

Vediamo ora le funzioni principali di BPEL Console per la gestione dei processi BPEL attivi. In figura 4.4 è mostrata la pagina iniziale di questa sezione.

In questa pagina sono elencati tutti i processi attivi presenti nel dominio a cui siamo connessi ed è fornita una sintesi sullo stato delle istanze dei processi BPEL

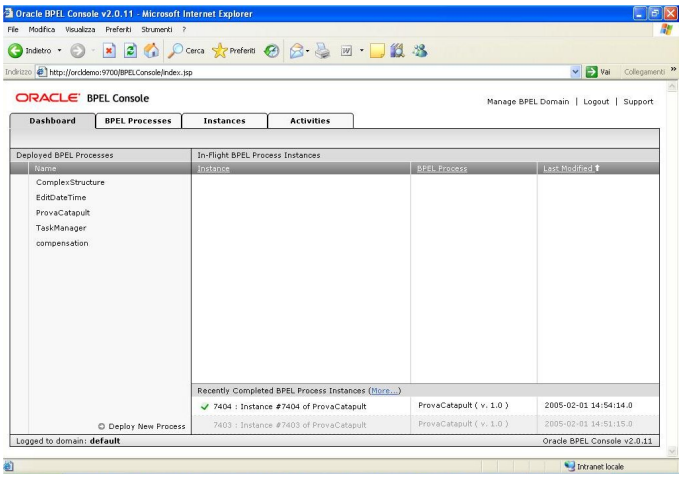

Figura 4.4: BPEL Console: Pagina iniziale

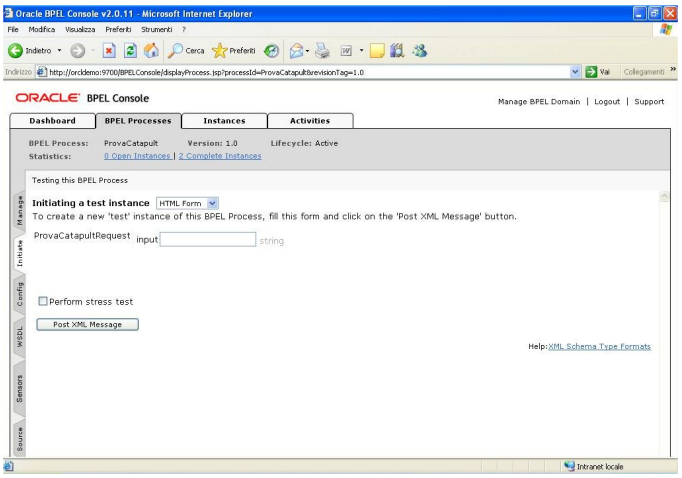

Figura 4.5: BPEL Console: Inizializzazione di una istanza di test

in esecuzione e di quelle completate. Inoltre è possibile effettuare il deploy di un nuovo processo o accedere alla pagina per la gestione del dominio. In figura 4.5 è rappresentata, invece, la pagina per la creazione di una istanza di test per un processo BPEL.

Una volta creata, l'istanza può essere amministrata tramite la pagina mostrata in figura 4.6. *Visual Flow* mostra graficamente le istruzioni eseguite dal processo

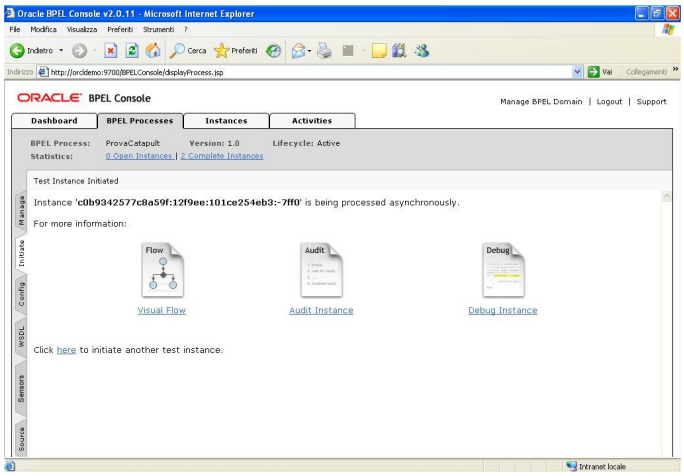

Figura 4.6: BPEL Console: Gestione di una istanza

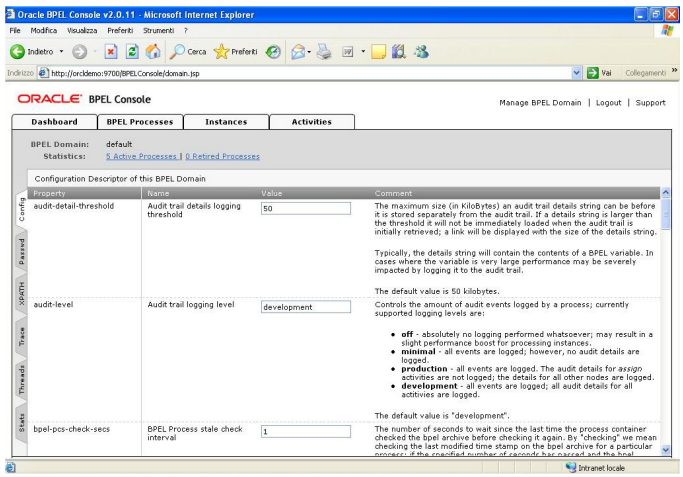

Figura 4.7: BPEL Console: Configurazione del dominio

durante la sua esecuzione. *Audit Trail* le informazioni elaborate da ogni attività svolta dal processo BPEL e in particolare il contenuto dei messaggi scambiati con i partner link. *Debug* ci consente invece il debug sull'istanza del processo BPEL.

Sempre in questa sezione possiamo accedere alla pagina dedicata alla configurazione del dominio a cui siamo connessi (figura 4.7) in cui possiamo:

- Configurare la password di accesso al dominio;

- Consultare la lista delle operazioni XPath;
- Consultare le statistiche sui thread allocati e sull'esecuzione dei processi BPEL;
- Configurare parametri per la gestione dell'istanza di un processo BPEL e per la gestione di alcune funzionalità legate a BPEL Console;

Amministrazione del BPEL Server

Collegandoci a questa sezione della BPEL Console (figura 4.8) potremo configurare parametri di BPEL Server quali l'application server utilizzato o la URL del BPEL SOAP Server, inoltre potremo amministrare i domini esistenti e se necessario crearne di nuovi.

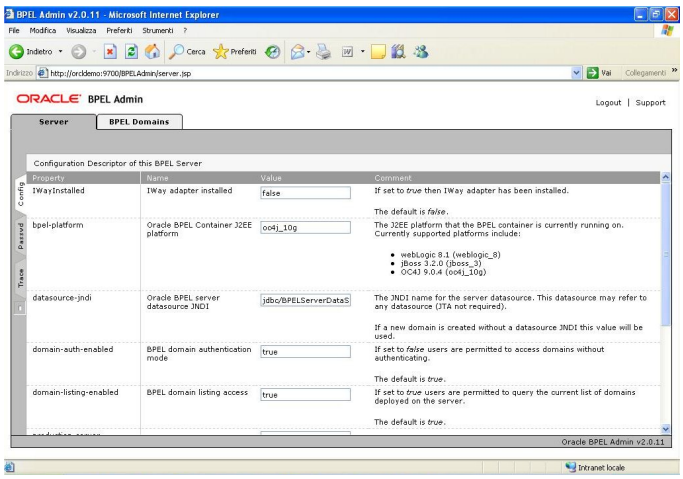

Figura 4.8: BPEL Console: Configurazione del BPEL Server

4.3 Oracle BPEL Designer 0.9.6

Questo è l'editor grafico, messo a dispozione da Oracle, per lo sviluppo di processi BPEL. Il software supporta la specifica BPEL4WS 1.1 e tutte le funzionalità di

BPEL Server.

Una volta creato il progetto BPEL Eclipse possiamo iniziare l'implementazione del workflow del Business Process.

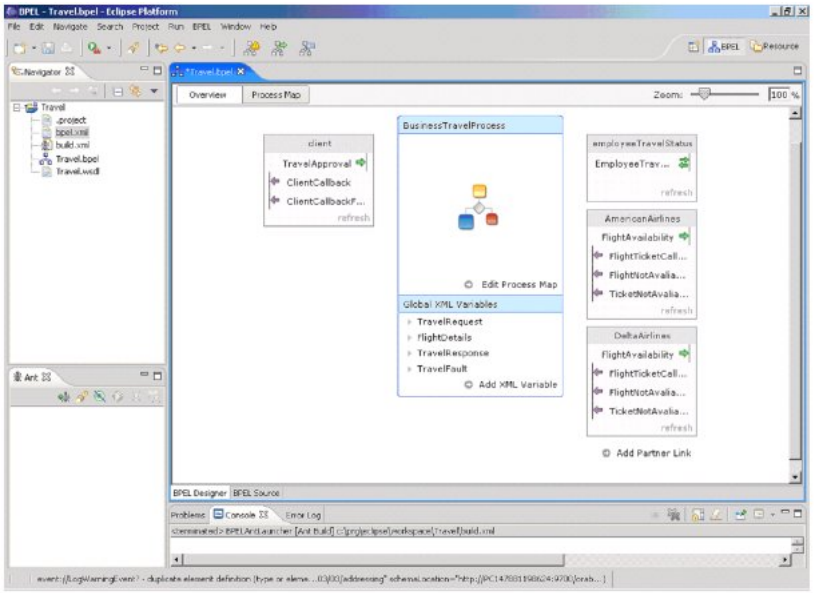

Figura 4.9: BPEL Designer

In questa schermata vengono mostrati il client che inizializza il processo BPEL, le variabili e i partner link coinvolti nell'orchestrazione. All'interno di BPEL Designer abbiamo un wizard per la creazione di una nuova variabile XML (figura 4.10), in cui dovrà essere specificato se questa si riferisce a un messaggio Soap, a un elemento o a un XML Schema.

Sempre dalla schermata iniziale possiamo utilizzare il wizard per la creazione di un nuovo partner link (figura 4.11), in questo caso possiamo specificare la locazione del WSDL del servizio oppure utilizzare il provider UDDI di BPEL Server.

In figura 4.12 è invece mostrata la schermata dedicata all'implementazione del processo BPEL. Sulla destra sono elencate tutte le attività consentite dalla

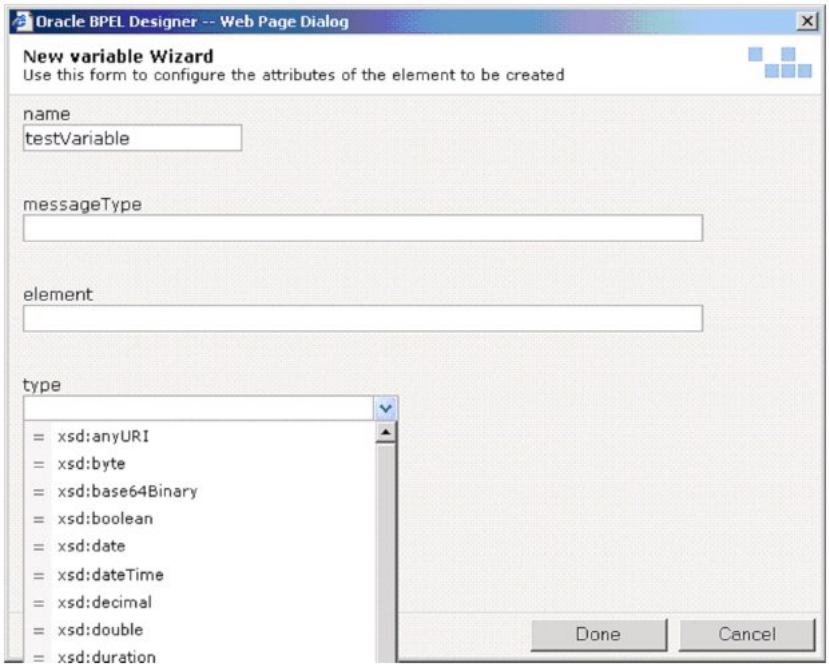

Figura 4.10: BPEL Designer: wizard per la creazione di una nuova varibile XML

specifica BPEL e quelle aggiunte da Oracle, che possono essere selezionate e trascinate nella Business Logic del processo.

Per l'elemento $< assign >$ è stato creato un wizard per creare espressioni XPath e per facilitare le operazioni di inizializzazione delle variabili. In figura 4.13 è mostrata questa funzionalità.

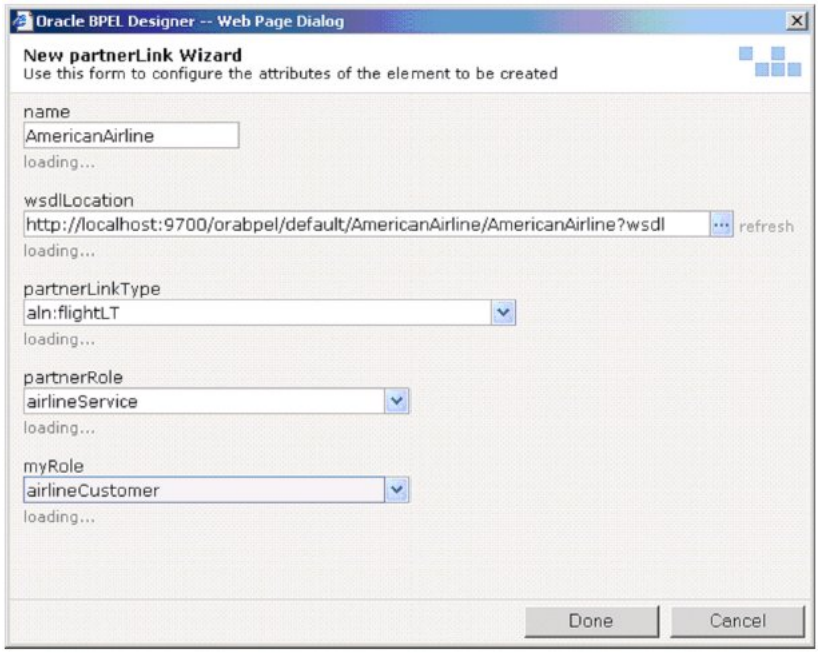

Figura 4.11: BPEL Designer: wizard per aggiungere un nuovo Partner Link

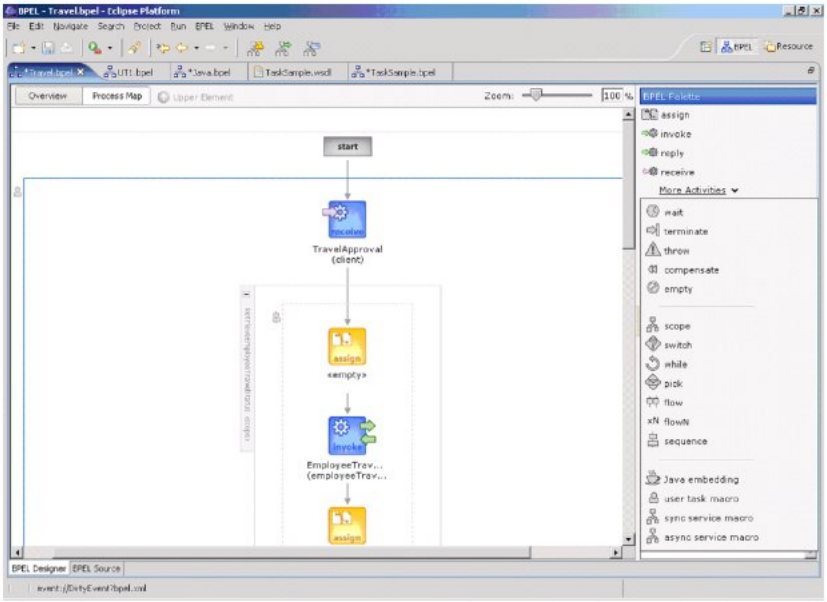

Figura 4.12: BPEL Designer: fase di implementazione

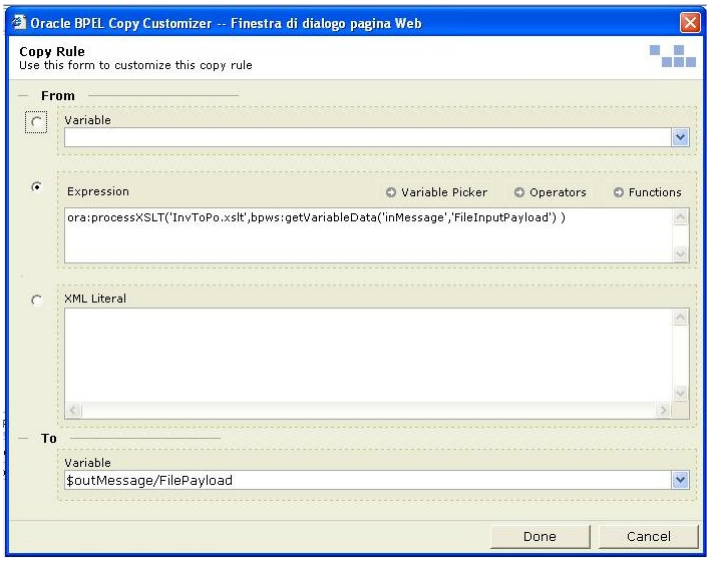

Figura 4.13: BPEL Designer: wizard per l'elemento $< assign >$

⋆⋆⋆

5

Catapult

Catapult è un processo interno a Ebilling S.p.A. implementato in C# in ambiente .NET 1.1. La sua funzione risiede nel muovere i file in arrivo via FTP dall'area di ricezione in DMZ *a quella di elaborazione in back-end. Catapult accede alla DMZ dalla LAN interna dell'azienda, garantendo consistenza al modello DMZ e sicurezza per quanto riguarda violazioni della rete interna da parte di utenti esterni. Per valutare la specifica BPEL4WS e l'ambiente Oracle, Catapult è stato implementato come un insieme di risorse SOA orchestrate tramite un processo BPEL.

5.1 UML Process Diagram

Nel diagramma in figura 5.1 sono mostrate le funzionalità del processo Catapult, il cui compito è quello di spostare i file dall'area in DMZ alla NAS. Una volta che un file arriva nell'area di ricezione FTP, viene notificato dal software *Thor*, che

* DeMilitarized Zone: è un computer host o una piccola rete che rimane tra la rete pubblica e la rete interna di una azienda. In una configurazione tipica la DMZ non può aprire sessioni verso la rete interna: questo per evitare violazioni da parte di utenti esterni non autorizzati, i quali possono accedere solamente ai servizi che risiedono in DMZ e che l'azienda rende disponibili per la rete pubblica. Naturalmente gli utenti della LAN interna dell'azienda potranno inizializzare una sessione verso la DMZ e accedere alla rete pubblica.

invia un messaggio alla *coda MSMQ* * su cui Catapult effettua periodicamente un polling attraverso un Web Service appositamente creato per questo scopo. Il messaggio di notifica inviato da Thor viene letto dal processo che registra l'operazione sul log di testo e sul database, ed effettua lo spostamento del file nell'area di elaborazione all'interno della NAS. Catapult viene inizializzato tramite un file di configurazione in cui sono descritti parametri tra quali il nome della coda MSMQ e l'intervallo di tempo per il polling.

Per l'implementazione del processo Catapult si è scelta la strada appena enunciata sia per la semplicità di realizzazione del sistema di polling, sia per verificare i costi che avrebbe portato in termini di interoperabilità con altri software utilizzati da Ebilling S.p.A..

Di seguito è mostrato un esempio di un messaggio di notifica inviato da Thor:

```
<AE-ENV actor="thor"
        server="nomeServer"
        timestamp="9/7/2004 10:56:32 AM">
  <id value="12345"/>
  <azienda value="1000"/>
  <aziendadesc value="xxx"/>
  <commessa value="1000"/>
  <commessadesc value="test"/>
  <filename value="SourcePath"/>
  <destination value="DestinationPath"/>
  <lastwrite value="15/7/2004 11:44:22 PM"/>
  <size value="25" type="byte"/>
  <direction value="1"/>
</AE-ENV>
```

* MicroSoft Message Queue: è il sistema di comunicazione basato su code di messaggi sviluppato da Microsoft.

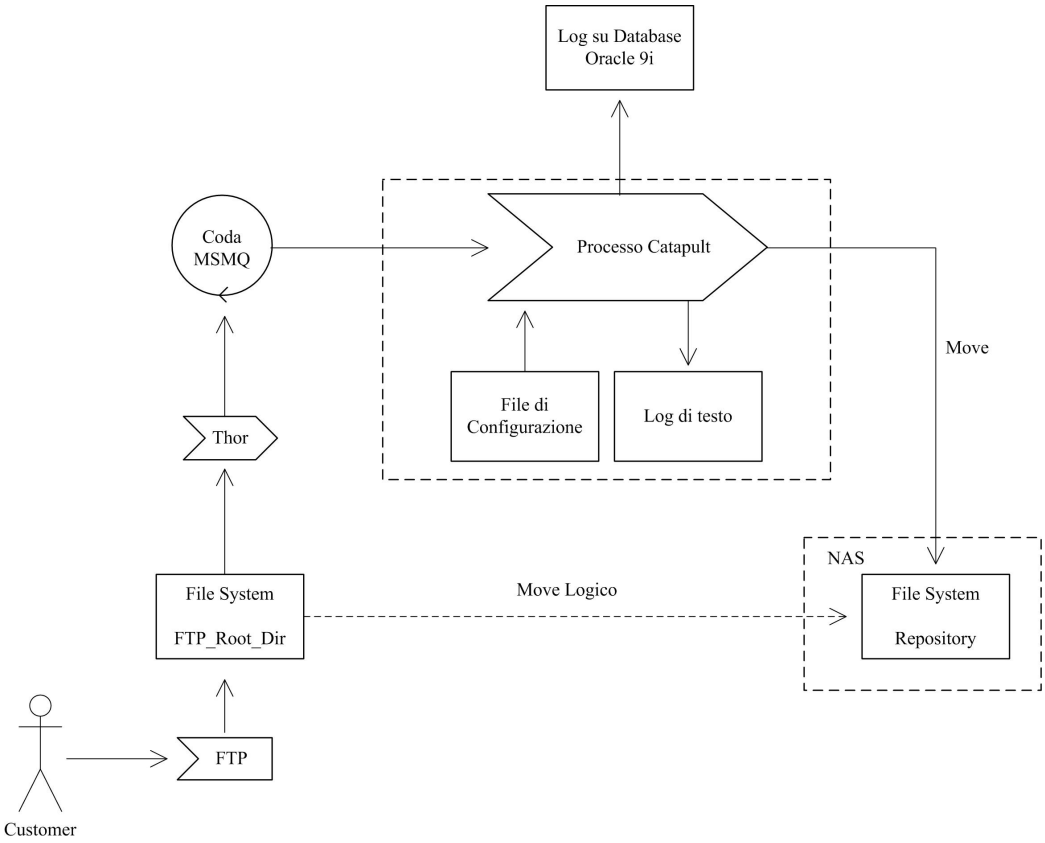

Figura 5.1: Processo Catapult: UML Process Diagram

Come si può notare, il messaggio, fornisce i valori del path di origine e di destinazione del file in arrivo in DMZ.

5.2 UML Deployment Diagram

Come illustrato in figura 5.2 abbiamo Oracle Process Manager 2.0.11 installato su una macchina Linux, come del resto l'application server J2EE * e il database Oracle 9i utilizzati come supporto alle funzionalità del BPEL Server. Sulla stessa macchina è avvenuto anche il deploy del processo Catapult e di tutti i servizi orchestrati, a meno del Web Service per il polling sulla coda MSMQ, che risiede

* Nel nostro caso è stato utilizzato OC4J (Oracle Container for J2EE)

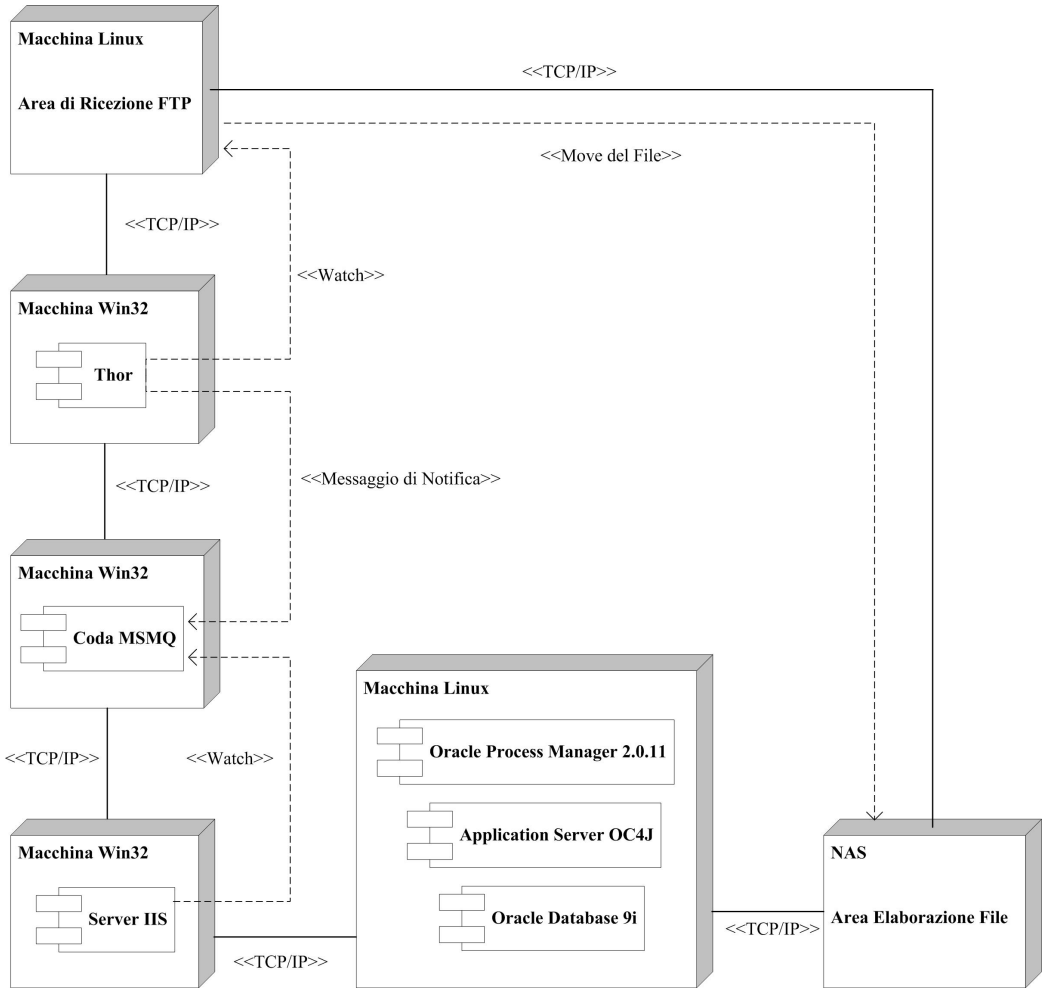

Figura 5.2: Processo Catapult: UML Deployment Diagram

su una macchina Windows32 all'interno di un application server Microsoft IIS. E' importante notare l'elevata interoperabilità consentita dalle tecnologie utilizzate nell'implementazione del processo Catapult, che hanno permesso l'integrazione tra sistemi e piattaforme diverse semplicemente importando l'interfaccia WSDL del servizio partecipante al workflow del Business Process.

5.3 Business Logic del Processo Catapult

Il prcocesso BPEL Catapult viene iniziato tramite la BPEL Console con un messaggio che contiene il path del file di configurazione e orchestra complessivamente sette partner link:

- **ReadConfig**: questo servizio implementato in Java è utilizzato per il caricamento del file di configurazione per il processo Catapult;
- **GetMSMQ**: è il Web Service C# che effettua il polling sulla coda MSMQ. Comprende anche una operazione per segnalare eventuali errori che possono avvenire sulla coda;
- **Move**: è il partner link in Java incaricato del move del file in arrivo in FTP alla NAS;
- **LogTXT**: tramite questo servizio Java scriviamo il log di testo per le operazioni compiute dal processo Catapult;
- **LogDB**: scrive il log sul database Oracle 9i sfruttando i connector disponibili nel BPEL Server;
- **EditDateTime**: questo è un processo BPEL utilizzato per gestire i formati dateTime;
- **E-Mail**: è il servizio di supporto per le E-Mail presente anch'esso all'interno del BPEL Server, utilizzato nel nostro caso per spedire messaggi agli indirizzi presenti nel file di configurazione, nel caso si verifichino errori durante l'esecuzione del processo BPEL;

In figura 5.3 è illustrata l'implementazione del workflow del processo Catapult.

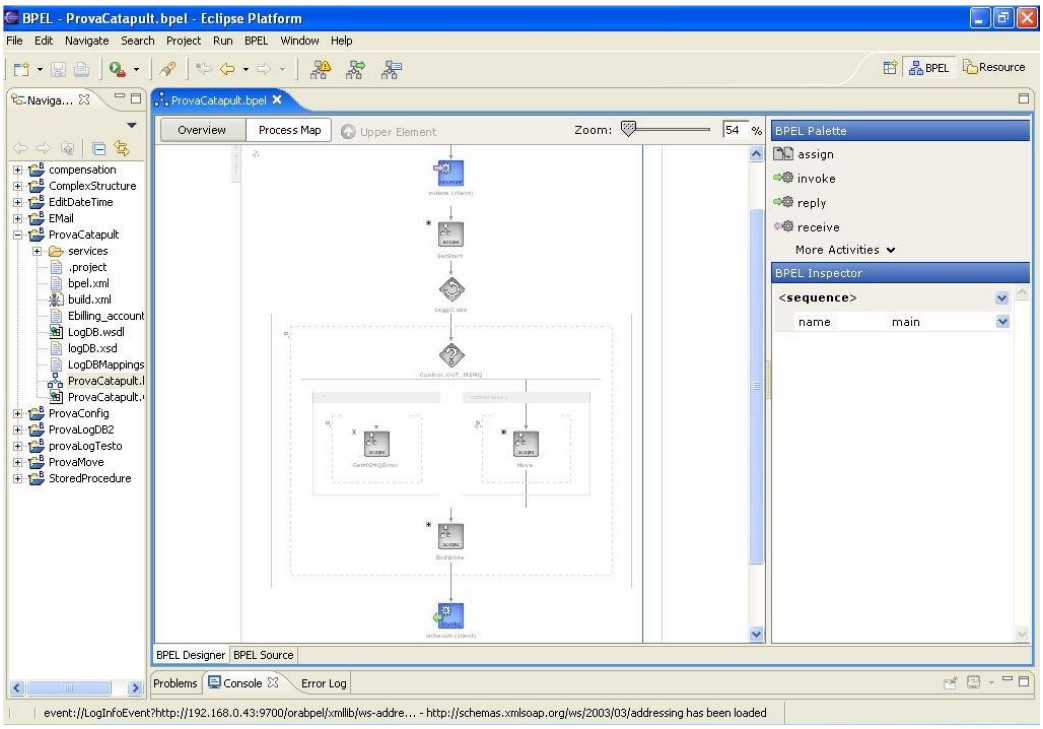

Figura 5.3: Processo Catapult: Business Logic

Una volta inizializzato con il valore del path del file di configurazione, Catapult invoca il servizio *ReadConfig* ed effettua una prima chiamata a *GetMSMQ*. Successivamente inizia del ciclo while, che effettua il polling sulla coda MSMQ e contiene le istruzioni per lo spostamento del file dall'area di ricezione FTP alla NAS. Il processo termina regolarmente quando *GetMSMQ* legge dalla coda la stringa byebye che provoca l'interruzione del ciclo while. In uscita verranno restituiti i risultati delle operazioni di log di testo. Il processo può anche interrompersi a causa della gestione di un errore critico oppure può essere terminato o sospeso tramite BPEL Console.

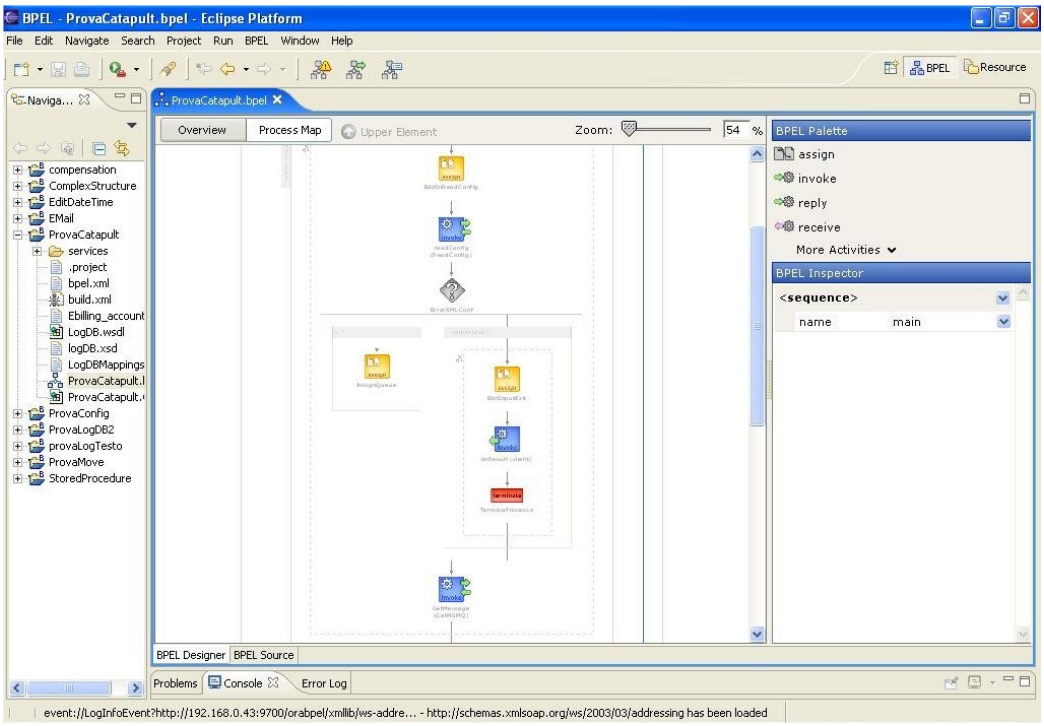

Figura 5.4: Processo Catapult: Caricamento del file di configurazione

5.3.1 Caricamento del file di configurazione

Una volta inizializzato, Catapult carica il file di configurazione XML che contiene informazioni quali:

- Il nome della coda MSMQ;
- L'intervallo di tempo per il polling sulla coda MSMQ;
- Il path di destinazione per il file di log di testo;
- L'elenco degli indirizzi Mail per l'invio dei messaggi in caso di errori durante l'esecuzione del processo Catapult;

Prima che inizi il ciclo while viene effettuata una prima invocazione al servizio *GetMSMQ*. . Se il path del file di configurazione non è valido allora il processo viene terminato.

5.3.2 Ciclo While

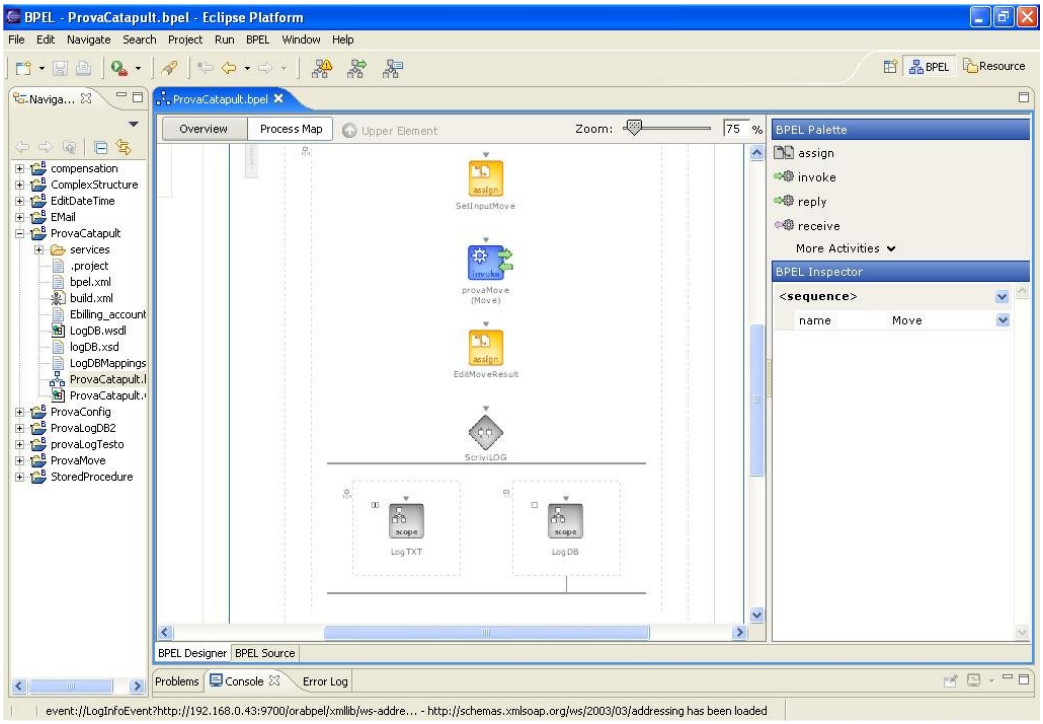

Figura 5.5: Processo Catapult: Invocazione del partner link *Move*

Tramite il ciclo while implementiamo il polling sulla coda MSMQ. Al suo interno abbiamo un elemento $< switch >$ attraverso il quale controlliamo la validità del messaggio restituito dall coda MSMQ. Se questo è corretto verrà invocato il servizio *Move* che effettuerà lo spostamento del file dall'area di ricezione FTP alla NAS (figura 5.5). Verrà poi invocato *EditDateTime* per conformare il formato dateTime utilizzato nel messaggio di Thor a quello utilizzato nella tabella per il log sul database, e quindi in parallelo *LogTXT* e *LogDB*. Prima di un ul-

terirore polling sulla coda MSMQ abbiamo un $< wait >$ per un periodo definito all'interno del file di configurazione.

Nel caso in cui avessimo ricevuto un messaggio vuoto da *GetMSMQ*, Catapult avrebbe invocato l'operazione *GetMSMQError* (figura 5.6) che ci avrebbe informato sulla causa di questa anomalia, che potrebbe essere dovuta a un errore di Timeout (coda MSMQ vuota) oppure a un errore più serio che avrebbe richiesto la sua registrazione nel log di testo e la comunicazione dell'errore tramite E-Mail all'elenco di indirizzi contenuto nel file di configurazione.

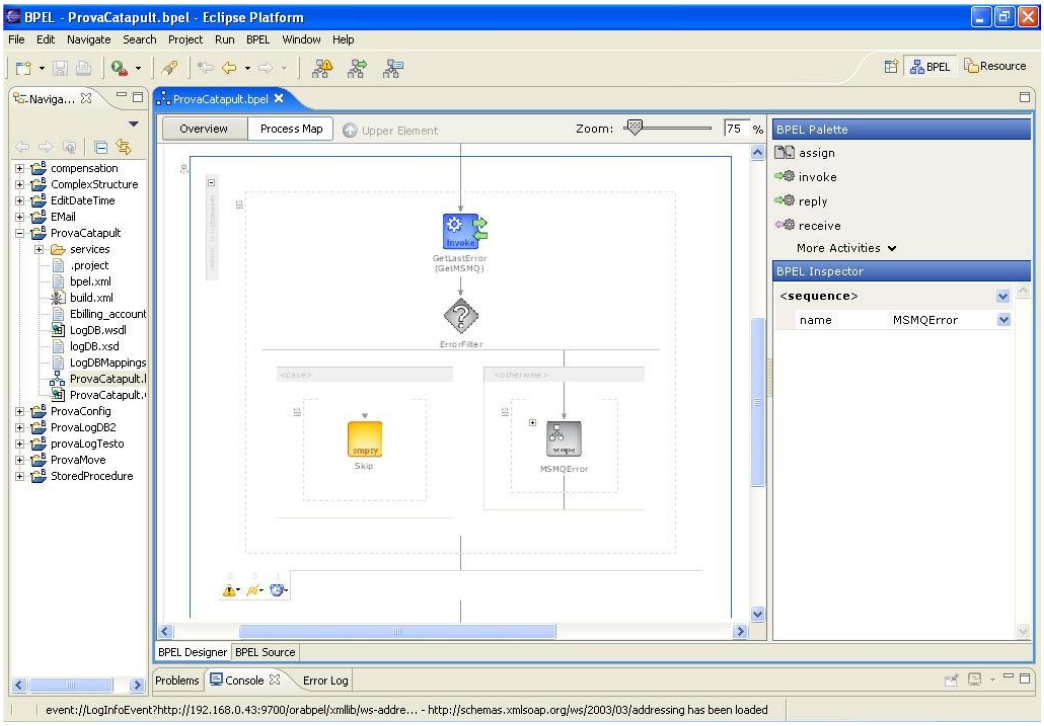

Figura 5.6: Processo Catapult: Invocazione dell'operazione *GetMSMQError*

5.3.3 Gestione degli errori

Ogni servizio orchestrato nel processo Catapult comunica i risultati delle operazioni da lui eseguite e nel caso di errori critici che non consentono la pros-

ecuzione del workflow, vengono inviati messaggi di segnalazione agli indirizzi elencati nel file di configurazione, viene scritto il log di testo e quindi viene terminato il processo. All'interno di ogni $< scope >$ è configurato un $< eventhandlers >$ che stabilisce come tempo massimo per l'esecuzione delle istruzioni contenute nello $< scope >$ un intervallo di cinque minuti. Se questo scade viene segnalato un errore dall'elemento $< throw >$ e poi gestito tramite $< catch >$. Per tutti gli altri tipi di errore è stato implementato un $< catchAll >$. Anche in questi ultimi due casi, Catapult segnalerà l'errore tramite E-Mail, scriverà i relativi log di testo e poi terminerà la sua esecuzione.

6

Conclusioni

Durante lo sviluppo del processo Catapult sono state sperimentate molte delle caratteristiche della specifica BPEL e dell'ambiente Oracle. Da questa analisi sono emersi i seguenti punti di forza:

- **Elevata Interoperabilità**: grazie alla tecnologia dei web service e al meccanismo di orchestrazione dei servizi proprio della specifica BPEL è risultata molto semplice l'integrazione tra piattaforme e servizi eterogenei quali piattaforme Linux e Windows, server IIS e application server J2EE, Web Service C# e Java.

- **BPEL Designer**: è risultato una strumento molto efficace nell'implementazione del processo Catapult, rendendo estremamente veloce lo sviluppo di workflow di Business Process complessi;

- **Funzionalità del BPEL Server**: anche se i servizi implementati da Oracle all'interno del BPEL Server sono esclusivi di questo ambiente e quindi il loro utilizzo limita la portabilità del processo BPEL verso altri server BPEL, sono comunque risultati efficaci ai fini della realizzazione di Cata-

pult. Hanno suscitato particolare interesse il servizio per le E-Mail, il Task Manager per l'interazione di un utente umano con il processo BPEL, i connector, WSIF per estendere il modello de web service a risorse che non lo sono e l'elemento $< bpelx : exec >$ per inserire codice Java all'interno del file sorgente BPEL;

- **Oracle BPEL Forum**: Oracle ha creato un Forum come supporto agli sviluppatori in cui i creatori dell'ambiente BPEL rispondono a domande sui problemi riscontrati nell'utilizzo dei software forniti da Oracle;

Mentre la specifica BPEL risulta stabile nelle sue funzionalità il prodotto offerto da Oracle non è ancora maturo e presenta diverse lacune:

- Esistono molti bug che compaiono durante l'utilizzo dell'ambiente Oracle. Purtroppo non esiste una documentazione accessibile agli sviluppatori, in cui questi bug vengano mappati e venga fornito un workaround per una loro soluzione;

- Al momento la documentazione fornita da Oracle sui propri software è scarna se non inesistente, sia per quanto riguarda la fase di sviluppo del processo BPEL sia per quanto riguarda la parte riguardante l'architettura dei software;

- Alcuni servizi non sono ancora ottimizzati, tra questi abbiamo il connector FileAdapter per l'interazione con il file system e l'elemento $< bpelx : exec >$. Il FileAdapter espone ancora troppi vincoli: dobbiamo ad esempio specificare il path di origine e di destinazione di un file a tempo di deploy del processo e non durante la sua esecuzione. Non è invece previsto uno strumento per il debug del codice Java compreso nel costrutto $< bpelx : exec >$ con evidenti difficoltà di implementazione.

- La BPEL Console non è stata ottimizzata per l'utilizzo di Mozilla Firefox come browser per l'accesso alle sue funzionalità, creando difficoltà per utenti Linux/Unix;

- Non esiste ancora una versione Linux del BPEL Designer;

Per risolvere questi problemi, dovuti alla giovane età del prodotto, Oracle sta rilasciando molto velocemente nuove versioni di questi software. Durante il mio periodo di tesi (circa quattro mesi) sono state rilasciate tre diverse versioni dell'Oracle Process Manager e dell'Oracle BPEL Designer, ognuna delle quali ha aggiunto nuove funzionalità e ha migliorato quelle esistenti. Le mancanze che sono state evidenziate sono quindi in via di risoluzione e non rappresentano un grosso ostacolo allo sviluppo e all'esecuzione dei processi BPEL.

⋆⋆⋆

7

Appendice: requisiti e licenze d'uso per l'ambiente Oracle

Purtroppo è necessario precisare che Oracle non fornisce informazioni sui requisiti Hardware per l'installazione dei software, pertanto in questa appendice non vi si farà riferimento.

Oracle BPEL Process Manager 2.0.11

In supporto sono richiesti:

- Internet browser (consigliato Internet Explorer 6.0);
- JDK 1.4.1 (o superiori);
- Application Server J2EE;
- Database;

L'Oracle BPEL Process Manager 2.0.11 è fornito nelle versioni per sistemi Linux e per sistemi Windows 32. Nel pacchetto di installazione di questo software

sono compresi un database (Oracle Lite *) e un application server J2EE (versioni disponibili per OC4J †, WebLogic, JBoss). Sono inoltre supportati database quali Oracle 9i, DB2, Microsoft SQL e Pointbase e application server J2EE forniti da IBM e Sun.

La licenza è necessaria solamente per uso commerciale del software (30.000 euro per CPU compreso application server J2EE e database Oracle Lite).

Oracle BPEL Designer 0.9.6

E' necessaria l'installazione della piattaforma Eclipse nella versione denominata eclipse-SDK-3.0-win32.zip. Oracle BPEL Designer, infatti, è rilasciato per sistemi Windows 32 come plug-in della piattaforma Eclipse.

Il software è completamente open source e non richiede l'acquisto di nessun tipo di licenza.

In futuro verrà fornito come plug-in di Oracle JDeveloper anche nella versione per sistemi Linux/Unix.

* deriva da Oracle 9i e fornisce le funzionalità di base necessarie a BPEL Server.

† Oracle Container for J2EE: è l'application server J2EE sviluppato da Oracle.

Bibliografia

[1] Alessandro Bemporad.
GIOP/IIOP sull'Autostrada CORBA, 1998.
http://www.mokabyte.it/1998/12/iiop.htm.

[2] Hao He.
What is Service-Oriented Architecture?, Settembre 2003.
http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html.

[3] World Wide Web.
Web Service and Service-Oriented Architecture.
http://www.service-architecture.com/index.html.

[4] Rajan Vannin.
Sviluppo e integrazione di applicazioni Web innovative tramite Web Service e Workflows, 2004.

[5] World Wide Web.
Business Process Execution Language for Web Service, Version 1.1, Maggio 2003.
http://www-106.ibm.com/developerworks/library/ws-bpel/.

[6] Matjaz B. Juric, Benny Mathew, Poornachandra Sarang.
Business Process execution Language for Web Service,

Chapter 4 "Oracle BPEL Process Manager.
http://www.oracle.com/technology/books/pdfs/1183_BPEL_Preview_Ch4.zip.

[7] World Wide Web.
Oracle BPEL Web Site.
http://www.oracle.com/technology/products/ias/bpel/index.html.

[8] World Wide Web.
Oracle AQ (Advanced Queuing)
http://www.oracle.com/technology/sample_code/tech/java/codesnippet/aq/MessagePropagation.htm.

[9] World Wide Web.
Apache Web Service Project.
http://ws.apache.org.

[10] World Wide Web.
XML Path Language (XPath), Version 1.0 , Novembre 1999.
http://www.w3.org/TR/1999/REC-xpath-19991116.

[11] World Wide Web.
Oracle XSQL Pages and the XSQL Servlet, Version 9.2.0.6, Luglio 2002
http://www.oracle.com/technology/tech/xml/xdk/doc/production/java/doc/java/xsql/readme.html.

[12] World Wide Web.
XML Query (XQuery).
http://www.w3.org/XML/Query.

[13] World Wide Web.
XSL Transformation (XSLT), Version 1.0, Novembre 1999.
http://www.w3.org/tr/xslt.

[14] World Wide Web.

DMZ.

http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci213891,00.html.