UNIVERSITÀ DEGLI STUDI DI PARMA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Dottorato di Ricerca in Tecnologie dell'Informazione
XX Ciclo

Alessandro Negri

SISTEMI MULTI-AGENTE E WORKFLOW PER LA COMPOSIZIONE DI SERVIZI

DISSERTAZIONE PRESENTATA PER IL CONSEGUIMENTO
DEL TITOLO DI DOTTORE DI RICERCA

Gennaio 2008

Ringraziamenti

Ringrazio di cuore la mia famiglia per essermi stata sempre accanto ed avermi supportato e sopportato anche durante quest'esperienza post-laurea.

Ringrazio il mio tutor di dottorato Prof. Ing. Agostino Poggi per l'assistenza, la disponibilità e le doti scientifiche ed umane dimostrate in questi anni.

Ringrazio con affetto i colleghi del laboratorio AOT: Lorenzo, Marco, Federico, Michele e Paola.

Indice

Elenco delle Figure

Prefazione

L'utilizzo di sistemi distribuiti da parte di aziende ed organizzazioni no profit è aumentato esponenzialmente negli ultimi anni, spinto da fattori quali lo sviluppo di Internet, la maturazione e diffusione del protocollo HTTP e l'aumento della sicurezza nelle comunicazioni. Esistono numerosissime versioni di tali sistemi, a partire da gruppi di applicazioni omogenee che funzionano all'interno di un cluster fino ad arrivare a Grid su scala globale, composti da un vasto numero di sistemi forniti da diversi produttori e supportati da ambienti operativi eterogenei. Tali applicazioni spesso comunicano utilizzando protocolli proprietari e metodi d'invocazione progettati ad hoc dai loro sviluppatori. Numerosi standard sono stati implementati nel corso degli anni, molti dei quali con scarso successo. Al giorno d'oggi, alcune delle tecnologie chiave per quel che riguarda i sistemi distribuiti sono le Service Oriented Architectures (SOA), i Web Services e, nel caso specifico della ricerca effettuata durante questo lavoro di tesi, i sistemi ad agenti.

Con il termine SOA non si fa riferimento alla definizione di tecnologie e paradigmi per l'implementazione di sistemi distribuiti, ma si intende tutto l'insieme delle regole che ne definiscono la progettazione ed il design. Quando si parla di SOA, l'enfasi è posta sull'implementazione di componenti sotto forma di servizi atomici e modulari che vengono scoperti ed invocati dagli utenti. Questi servizi, essendo spesso distribuiti in modo eterogeneo e sviluppati da diversi produttori a livello mondiale, fanno riferimento a Web Services. Un Web Service è definito come un componente software distribuito che fornisce informazioni ad un'applicazione software piuttosto che ad un essere umano. I Web Services pubblicano i dettagli delle loro funzioni o interfacce, mantenendone però privata

l'implementazione; in questo modo un client che supporta lo stesso protocollo di comunicazione può interagire con il servizio indipendentemente dalla piattaforma sulla quale stanno funzionando o dal linguaggio di programmazione con il quale tale servizio è stato sviluppato. La semplicità con la quale i Web Services possono essere implementati e la capacità di accedervi da qualunque piattaforma o ambiente, ha portato ad una loro vasta adozione all'interno di sistemi distribuiti come oggetti virtuali che forniscono interfacce per l'accesso e la gestione di risorse.

Il lavoro di ricerca descritto in questo progetto di tesi, ha riguardato lo studio e la realizzazione di un sistema distribuito ad agenti per la gestione e la composizione di servizi utilizzando i workflow. Un sistema ad agenti, infatti, può essere considerato come un esempio di sistema distribuito arricchito da funzionalità avanzate, quali l'autonomia, la mobilità, la capacità di ragionamento ed il supporto di una piattaforma distribuita per la loro gestione. Tale piattaforma è stata identificata in JADE, un framework per lo sviluppo di sistemi ad agenti sviluppato da TILab di Torino in collaborazione con il Dipartimento di Ingegneria dell'Informazione dell'Università di Parma. Esso è distribuito da TILab come software open source sotto la licenza LGPL (Lesser General Public License Version 2). JADE è stato interamente realizzato in linguaggio Java, in particolare l'architettura del sotto-sistema di comunicazione sfrutta la moderna tecnologia ad oggetti distribuiti di Java per comportarsi come un camaleonte, scegliendo al tempo di esecuzione, in modo trasparente all'applicazione, il miglior protocollo di trasporto per ogni situazione.

Grazie all'utilizzo dell'add-on JADE WSIG, alla piattaforma ad agenti è stata aggiunta la capacità di invocazione ed esecuzione di Web Services definiti secondo lo standard WSDL e registrati in un UDDI directory. Tale add-on permette un'interazione basilare fra un qualunque agente presente all'interno di una piattaforma standard JADE ed un Web Service definito da terze parti. In questo modo un sistema distribuito ad agenti viene completato con funzionalità simili ad un'applicazione Service Oriented e gli agenti fungono da interfaccia fra il sistema ed i servizi implementati.

In un sistema di questo tipo, però, vengono a mancare capacità di gestione e composizione dei servizi, oltre ad un supporto alla definizione ed all'uso degli stessi. Per questo motivo è stato sviluppato un prototipo di un sistema per la composizione di servizi messi a disposizione da Web Services chiamato W-GAIN. Tale sistema è basato sulla piattaforma JADE e prevede di fornire supporto all'utente sia nella fase di esecuzione che in quella di sviluppo di applicazioni distribuite basate sui workflow. In particolare il sistema permette di definire workflow, componendo i servizi forniti dai diversi nodi del sistema (agenti JADE che interrogano Web Services), distribuire in modo trasparente i lavori ai nodi della rete, eseguire i task, monitorare le risorse e gestire le situazioni di risorse non disponibili. Per la descrizione dei task da effettuare, la loro composizione e la creazione di processi complessi basati sui vari servizi atomici, è stato scelto di utilizzare la tecnologia dei workflow.

Un workflow viene definito da WfMC, un'associazione no profit creata per la definizione e standardizzazione di sistemi a workflow, come l'automazione di una parte o dell'intero processo aziendale dove documenti, informazioni e compiti vengono passati da un partecipante ad un altro per ricevere qualche tipo di azione, seguendo un determinato insieme di regole. Dunque un workflow riassume una serie di procedure che, poste in relazione le une con le altre, definiscono un contesto di lavoro strutturato, dove più unità lavorative, siano esse risorse umane, gruppi di lavoro o sistemi computerizzati (nel caso specifico agenti JADE), lavorano in concomitanza per un determinato fine. Ogni workflow è composto da un certo numero di processi, cioè procedure da seguire per ottenere un certo risultato facendo riferimento a determinate condizioni di partenza. Ogni processo viene descritto dall'insieme di attività e dalla sequenza con cui vengono svolte.

Realizzare un progetto di un workflow è complesso ed oneroso in termini di tempi di sviluppo e vi sono discrepanze fra i reali processi di workflow ed i processi percepiti dal management. Per questo motivo il sistema W-GAIN si pone l'obiettivo di semplificare la progettazione e la rappresentazione di un processo di business attraverso l'uso di workflow e, grazie al sistema distribuito ad agenti, ne gestisce l'esecuzione.

Per realizzare un primo prototipo del sistema W-GAIN è stato necessario sviluppare una serie di tool che ne permettessero un facile accesso ed utilizzo all'utente. E' stato realizzato, quindi, un editor visuale per workflow rappresentati tramite lo standard XPDL, un'estensione del linguaggio XML definito e mantenuto da WfMC. Tale editor è stato sviluppato all'interno della piattaforma Eclipse e permette la creazione, visualizzazione, modifica, salvataggio e validazione di un qualunque workflow descritto nello standard XPDL.

Una volta realizzato il workflow componendo l'insieme dei servizi messi a disposizione dagli agenti JADE, è necessario porre in esecuzione tali workflow nel sistema. Per fare ciò è stato realizzato un traduttore dal linguaggio XPDL a codice Java specifico per la piattaforma JADE. In questo modo, dalla semplice composizione grafica dei servizi e delle funzionalità che si vogliono implementare, è possibile controllare un sistema distribuito ad agenti senza la necessità di avere le conoscenze necessarie per programmare direttamente i vari componenti del sistema.

Nel Capitolo 1 si descrivono brevemente gli agenti, i sistemi multi-agente e la piattaforma JADE. Nel Capitolo 2 si trattano i concetti relativi ai workflow ed ai sistemi di gestione per workflow (Workflow Management System). Nel Capitolo 3 si fornisce una panoramica della piattaforma Eclipse e dei tool che sono stati utilizzati per la creazione dei sistemi e delle applicazioni descritte nei capitoli successivi. Nel Capitolo 4 si introduce il sistema W-GAIN per la composizione di servizi attraverso l'uso di un sistema ad agenti distribuito e workflow definiti nel linguaggio XPDL. Nel capitolo 5 si descrive sia a livello progettuale che funzionale l'editor di workflow che è stato sviluppato per descrivere i servizi all'interno del sistema W-GAIN. Nel Capitolo 6 si illustrano le problematiche relative alla traduzione di un workflow, descritto in linguaggio XPDL, in codice Java direttamente eseguibile su di un sistema ad agenti.

1
Sistemi Multi-Agente - JADE

Questo capitolo illustra le caratteristiche principali degli agenti ed introduce il concetto di sistema multi-agente. In particolare si pone l'attenzione su JADE (Java Agent Development Framework), sviluppato da TILab di Torino in collaborazione con il Dipartimento di Ingegneria dell'Informazione dell'Università di Parma.

1.1 Gli Agenti

Il termine "agente" è utilizzato in diversi ambiti, non solo nel mondo dell'informatica. E' possibile comunque fornire una definizione generale che ne illustri le principali caratteristiche.

Un agente è un'entità fisica o virtuale [6]:

- che è capace di agire in un particolare ambiente,
- che può comunicare direttamente con altri agenti,
- che è guidato da un insieme di comportamenti sotto forma di obiettivi individuali o funzioni di soddisfazione che cerca di ottimizzare,
- che può gestire da sé un insieme di risorse,
- che è capace di percepire l'ambiente in cui opera, del quale, però, ha solo una rappresentazione parziale,
- che possiede delle abilità e che può offrire dei servizi,
- che può riprodursi,

- il cui comportamento è teso a soddisfare i propri obiettivi, tenendo conto delle risorse a disposizione, delle proprie capacità, della propria percezione dell'ambiente e delle comunicazioni che riceve.

Dopo aver fornito una definizione generica, vediamo ora di specificare meglio quali sono le caratteristiche peculiari di un agente. Un'entità fisica è qualunque cosa agisca nel mondo reale, per esempio un robot od una macchina. Un componente software o un modulo computazionale, invece, sono entità virtuali e rimangono tali fino a che non hanno un'implementazione fisica.

Gli agenti sono capaci di agire: questo concetto è fondamentale perché implica il fatto che essi possano prendere delle iniziative che modificheranno l'ambiente esterno e tutte le loro decisioni future. Per questo motivo, essi possono comunicare fra di loro: la comunicazione rimane il mezzo principale per l'interazione reciproca.

Gli agenti sono autonomi: non sono diretti da comandi impartiti da un utente, ma hanno un proprio insieme di "comportamenti", sotto forma di obiettivi da raggiungere o di funzioni da ottimizzare. Una volta che sono stati definiti gli obiettivi e le regole di comportamento, l'agente è in grado di operare in modo totalmente indipendente all'interno del proprio ambiente, prendendo decisioni ed interagendo con gli altri agenti.

Un agente, quindi, è una sorta di "organismo vivente" il cui comportamento, che può essere riassunto in termini di comunicazione, azione e riproduzione, mira a soddisfare i propri bisogni e raggiungere i propri obiettivi, sulla base di tutti gli elementi che gli sono disponibili (percezioni, comunicazioni, rappresentazioni, azioni e risorse) [22].

1.1.1 Lo Standard FIPA

L'organizzazione internazionale FIPA (Foundation for Intelligent Physical Agents [16]), fondata nel 1996, ha come scopo quello di definire degli standard utili alle industrie per la realizzazione di applicazioni commerciali. FIPA ha studiato in modo sistemico il problema dell'interoperabilità fra agenti, formulando uno standard per tutti i livelli necessari: il trasporto dei messaggi, la semantica del

linguaggio di comunicazione fra agenti intelligenti, i servizi di gestione di base quali le pagine bianche e le pagine gialle [13]. FIPA fornisce solamente un modello di riferimento e non include alcun tipo di implementazione fisica. I dettagli dell'implementazione delle varie caratteristiche e degli agenti dipendono dalle scelte progettuali dei singoli sviluppatori.

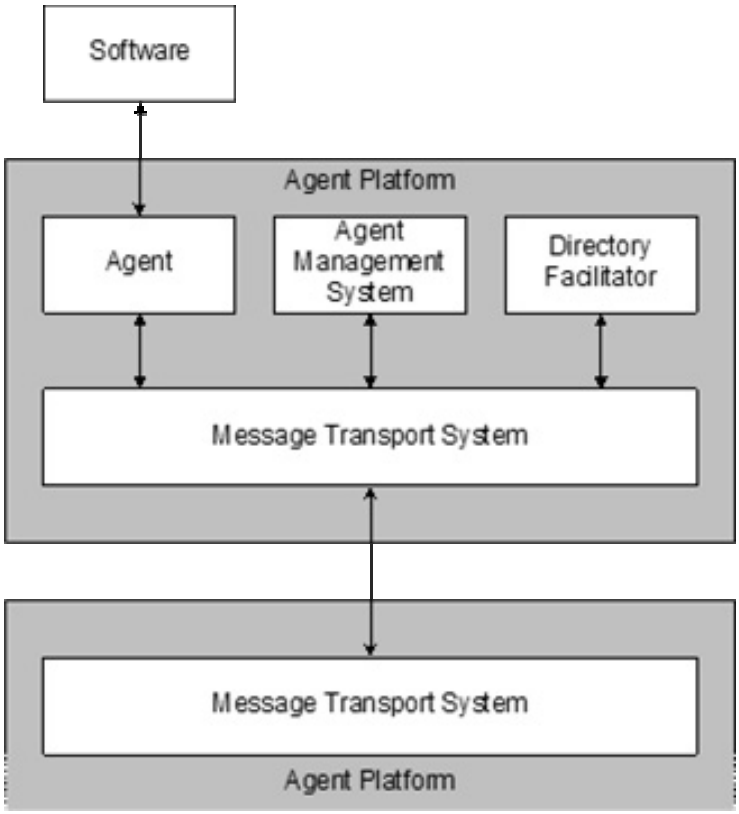

Figura 1 - FIPA Agent Management Reference Model

In Figura 1 è rappresentato il FIPA Agent Management Reference Model, ossia il modello presentato da FIPA per la creazione e la gestione di un sistema multi-agente [12]. Questo modello è formato dai seguenti componenti logici, ognuno dei

quali contiene un insieme di interfacce da sviluppare per ottenere un sistema reale [15]:

- un ***Agente*** inteso come un processo computazionale che implementa le funzionalità di autonomia e comunicazione. Esso deve avere almeno un gestore e deve essere identificato da un proprio `AID` (Agent Identifier). L'`AID` è un'etichetta che viene utilizzata all'interno del sistema multi-agente per identificare in maniera univoca e non ambigua un determinato agente.
- Un ***Directory Facilitator (DF)***, cioè un componente che fornisce un servizio di pagine gialle. Gli agenti possono registrare i propri servizi nel DF oppure possono interrogarlo per conoscere i servizi messi a disposizione dagli altri agenti.
- Un ***Agent Management System (AMS)***, cioè un componente che esercita un controllo di supervisione sull'accesso e l'uso di tutto il sistema. Può esistere un solo AMS per ogni piattaforma che fornisce fra gli altri servizi quello di pagine bianche.
- Un ***Message Transport Service (MTS)***, cioè un metodo di default per la comunicazione fra agenti localizzati su diverse piattaforme.
- Un ***Agent Platform (AP)***, cioè l'infrastruttura fisica nella quale sono sviluppati gli agenti.
- ***Software*** descrive tutto ciò che non è contenuto nella piattaforma ad agenti.

1.1.2 Il linguaggio FIPA ACL

Lo standard FIPA oltre a definire la struttura della piattaforma ad agenti, specifica in maniera completa e precisa anche il linguaggio che viene utilizzato nella comunicazione fra agenti. Questo linguaggio, chiamato ACL (Agent Communication Language [14]), definisce la sintassi e la semantica della

comunicazione fra agenti che necessitano di relazioni complesse per collaborare, competere e negoziare. La semantica di FIPA ACL si basa sulla definizione dell'effetto di ogni messaggio (`rational effect`) e dello stato in cui esso può essere trasmesso (`feasibility preconditions`) [13].

Un messaggio FIPA ACL contiene un insieme di parametri che varia in funzione della situazione a cui deve essere applicato. L'unico parametro obbligatorio, che deve essere presente in tutti i messaggi, è il `performative` che indica il tipo di comunicazione che si vuole effettuare. In una tipica comunicazione sono quasi sempre presenti anche il `sender` (l'agente che invia il messaggio), il `receiver` (il destinatario del messaggio) ed il `content` (il contenuto vero e proprio del messaggio) [13].

1.2 La piattaforma JADE

JADE (Java Agent Development Framework [21]) è un framework software che semplifica lo sviluppo di applicazioni di agenti, fornendo dei servizi di base conformi allo standard FIPA ed interoperabili con altre piattaforme, anch'esse conformi allo standard. JADE è stato sviluppato da TILab S.p.A. di Torino in collaborazione con AOT Lab del Dipartimento di Ingegneria dell'Informazione dell'Università di Parma. Esso è distribuito da TILab come software open source sotto la licenza LGPL (Lesser General Public License Version 2).

JADE è stato interamente realizzato in linguaggio JAVA, in particolare l'architettura del sotto-sistema di comunicazione sfrutta la moderna tecnologia ad oggetti distribuiti di JAVA per comportarsi come un camaleonte, scegliendo al tempo di esecuzione, in modo trasparente all'applicazione, il miglior protocollo di trasporto per ogni situazione.

Per farsi un'idea di ciò che è realmente JADE, si elencano di seguito le caratteristiche principali.

- Piattaforma ad agenti distribuita. La piattaforma è costituita da uno o più contenitori di agenti (agent container) a cui si può supporre, per

semplicità, corrisponda una macchina virtuale Java indipendente. La comunicazione tra i contenitori è basata su Java RMI. La composizione della piattaforma può variare al tempo di esecuzione con l'aggiunta o la rimozione di contenitori di agenti.

- Interfaccia grafica per la gestione degli agenti e dei rispettivi container anche da host remoto.
- Piattaforma ad agenti conforme agli standard FIPA. Sono presenti un AMS (Agent Management System), uno o più DF (Directory Facilitator) ed un'implementazione di MTS (Message Transport Service).
- Trasporto efficiente dei messaggi FIPA ACL all'interno della stessa piattaforma. I messaggi, durante il trasferimento, sono codificati come oggetti Java.
- Mobilità intra-piattaforma degli agenti. Gli agenti si possono muovere da un container all'altro all'interno della stessa piattaforma.
- Supporto per la gestione delle attività concorrenti degli agenti, eseguite in parallelo, secondo il modello dei comportamenti o "`behaviour`".
- Strumenti per la gestione della piattaforma e per effettuare la monitorizzazione ed il debug degli agenti contenuti nella piattaforma stessa. Tutti questi strumenti sono implementati come agenti FIPA e non richiedono alcun supporto speciale per svolgere i compiti a loro assegnati, se non in alcuni casi con l'ausilio dell'AMS.
- Registrazione e cancellazione automatica degli agenti presso l'AMS.
- Supporto alle applicazioni per la definizione e l'uso di nuovi linguaggi e ontologie per le comunicazioni.

- Implementazione completa di una libreria di protocolli di interazione FIPA.

2
Modellazione e Gestione dei Workflow

Questo capitolo introduce il termine workflow e realizza una panoramica delle caratteristiche e degli standard che sono presenti sul mercato. Nella parte finale illustra la problematica della modellazione e composizione di servizi distribuiti attraverso Web Services.

2.1 Workflow e Workflow Management System

La gestione del workflow è un aspetto delle realtà aziendali che negli ultimi anni, partendo da un ruolo marginale, ha assunto un'importanza sempre crescente, fino a diventare un punto di forza nella realizzazione di infrastrutture informatiche. Il concetto di Workflow Management deriva dalla gestione dei flussi di lavoro tipici delle produzioni industriali, in particolare si identifica nella gestione dello scambio strutturato di informazioni tra diverse entità dell'azienda (individui, ruoli, applicazioni software), finalizzato all'ottenimento di un certo scopo e all'automazione dei processi per conseguire un determinato risultato economico o di mercato. Con il termine workflow si intende:

"l'automazione di una parte o dell'intero processo aziendale dove documenti, informazioni e compiti vengono

passati da un partecipante ad un altro per ricevere qualche tipo di azione, seguendo un determinato insieme di regole".

Dunque un workflow riassume una serie di procedure che, poste in relazione le une con le altre, definiscono un contesto di lavoro strutturato, dove più unità lavorative, siano esse risorse umane, gruppi di lavoro o sistemi computerizzati, lavorano in concomitanza per un determinato fine. Ogni workflow è composto da un certo numero di processi, cioè procedure da seguire per ottenere un certo risultato da determinate condizioni di partenza. Ogni processo si può definire attraverso la sequenza di azioni che devono essere svolte dai partecipanti ed attraverso l'evoluzione dello stato degli oggetti che lo compongono. Un processo, ad esempio, può essere l'iter per una richiesta di acquisto di un bene o l'insieme delle procedure per attivare un nuovo contratto per una linea telefonica in un'azienda di telecomunicazioni.

Le azioni che devono essere svolte nel processo sono chiamate attività del processo. Ogni processo viene descritto dall'insieme di attività e dalla sequenza con cui vengono svolte. Un'attività descrive un singolo passo del processo da eseguire: ad esempio prendere una decisione, apporre una firma ad un documento, eseguire una procedura in un nodo di una rete informatica. Il processo ha sempre una sola attività di inizio, il cui input è composto dagli oggetti che devono essere elaborati, ed un'attività di fine il cui output sono i risultati dell'esecuzione del processo. La sequenza di passi di un processo non è limitata ad una semplice sequenza lineare. Si possono, infatti, avere ramificazioni e ricongiungimenti per rappresentare attività che devono essere svolte in parallelo o in alternativa le une alle altre.

Ogni processo è un'entità statica, una sorta di definizione di procedure e iter da intraprendere: può essere eseguito più volte, cioè può avere più istanze, anche contemporanee. Ogni istanza di processo ha una propria identità distinta, anche se condivide con le altre istanze di processo la stessa sequenza di azioni da intraprendere. Un'istanza di processo è dunque un'entità dinamica: ha una data ed

un orario di creazione, dati registrati che cambiano di attività in attività ed una fine, la sua archiviazione.

I Workflow Management System (WfMS) sono applicazioni che gestiscono flussi di lavoro. Qualsiasi azienda, amministrazione pubblica o ente commerciale segue dei protocolli per gestire processi produttivi o amministrativi basati sul passaggio di informazioni o documenti: questi iter possono essere schematizzati come flussi di lavoro a cui si riferiscono i dipendenti e/o gli utenti. Nella gestione tipica di questi flussi di lavoro molte energie vengono spese inutilmente in due ambiti che non portano alcun risultato pratico. Da una parte si deve curare il passaggio del lavoro o dei documenti, perdendo tempo e risorse nel cercare di capire che cosa si debba fare di un certo oggetto o a quale ufficio debba essere mandato un incartamento. Dall'altra parte molte delle attività svolte sono ripetitive e automaticamente gestibili da un'applicazione software anziché da un essere umano. I WfMS si propongono di automatizzare queste fasi dispendiose e non redditizie dei flussi di lavoro con l'aiuto delle tecnologie informatiche. Il guadagno in efficienza portato dai WfMS è proporzionale alla complessità del processo modellato e alla percentuale di attività di cui è composto che possono essere rese automatiche. Un WfMS viene descritto, infatti, come:

> "un sistema che definisce, crea e gestisce l'esecuzione di workflow attraverso l'uso di software, coinvolgendo uno o più motori di workflow e che è in grado di interpretare definizioni di processo, interagire con i partecipanti del workflow e, se richiesto, invocare l'uso di applicazioni e strumenti dell'information technology".

In modo semplicistico si può affermare che i WfMS gestiscono il flusso di lavoro in modo tale che esso sia eseguito al momento giusto e dalla persona/macchina corretta. L'idea base dei WfMS è quella di separare i processi dalle risorse e dalle applicazioni, focalizzando l'attenzione sul processo generico e non sul contenuto del compito individuale. In particolare, in base alle caratteristiche del flusso di lavoro del processo aziendale su cui è applicato, il

WfMS è in grado di definire in modo netto le attività, i ruoli e le procedure, automatizzare il lavoro corrente, aiutare a risolvere situazioni di eccezioni o guasti, assegnare i compiti e le risorse agli utenti (in base ai loro ruoli), supportare i cambiamenti del flusso di lavoro e controllarne la consistenza prima di proseguire con le altre istanze del processo. Quindi, concludendo, i WfMS supportano la definizione, l'esecuzione ed il controllo di un processo aziendale, gestendo gli eventi interni e favorendo l'interoperabilità con sistemi esterni. L'automatizzazione del lavoro e l'assegnazione dei compiti porta a migliorare notevolmente l'efficienza: la prima consente di affidare agli strumenti di esecuzione sotto il diretto controllo del workflow tutte le attività che sono puramente automatiche e meccaniche, che non hanno bisogno di nessuna componente umana nello svolgimento. L'assegnazione dei compiti fa sì che ad ogni utente vengano fornite tutte le informazioni e gli strumenti necessari allo svolgimento della sua attività, in questo modo non gli mancherà niente di indispensabile, aumentando l'efficienza. Il controllo del sistema consente anche di eliminare eventuali errori umani. Quindi i benefici immediati che si ottengono sono:

- riduzione dei costi operativi;
- aumento della produttività;
- riduzione dei tempi di produzione.

Inoltre si assiste a:

- miglioramento dei servizi, poiché il tempo per eseguire un lavoro si riduce e le informazioni richieste sono disponibili in minor tempo.
- Miglioramento delle condizioni di lavoro, poiché gli aspetti più ripetitivi dell'attività del processo sono svolti automaticamente.
- Miglioramento delle comunicazioni tra utenti e impresa, per la possibilità di disporre più facilmente di informazioni sulle attività dell'azienda.
- Supporto manageriale e decisionale, poiché, potendo monitorare in modo più efficiente l'andamento dei processi, è possibile prendere decisioni più idonee.

- Comunicazioni tra organizzazioni, supportando attività che legano tra loro più strutture.
- Aumento della qualità del servizio offerto.
- Privacy e Sicurezza: solo l'utente autorizzato può accedere al compito e ai dati ad esso associati.

Nella seguente trattazione, si fa ampiamente riferimento alle indicazioni fornite dalla Workflow Management Coalition [33], un'organizzazione internazionale no-profit fondata nell'agosto del 1993 e costituita da produttori di workflow, utenti, analisti e gruppi di ricerca universitari. Essa è sorta con l'obiettivo principale di promuovere lo sviluppo dell'uso del workflow, stabilendo standard e terminologie comuni per consentire la compatibilità e l'interoperabilità tra prodotti eterogenei di workflow. L'interazione fra applicazioni di WfMS di natura diversa è indispensabile poiché ogni situazione richiede uno strumento diverso: lo scopo della WfMC è proprio quello di fornire standard per questa integrazione. Essa inoltre si pone come scopo quello di accrescere il valore degli investimenti nelle tecnologie di workflow, ridurre i rischi legati all'uso dei prodotti di workflow ed espandere il mercato, accrescendo la consapevolezza delle potenzialità dei workflow.

2.1.1 Architettura dei Sistemi

Tutti i WfMS hanno una serie di caratteristiche comuni che garantiscono una base per l'integrazione e l'interoperabilità. Si possono quindi individuare tre macroaree funzionali:

- i servizi di supporto alla costruzione, connessi alla definizione e alla modellazione del processo di workflow e delle attività che lo costituiscono.
- I servizi di controllo della messa in esecuzione, capaci di gestire i processi di workflow in un ambiente operazionale, seguendo la fase runtime delle varie attività.
- Le interazioni tra gli utenti umani e gli strumenti applicativi nell'analizzare i vari passi delle attività.

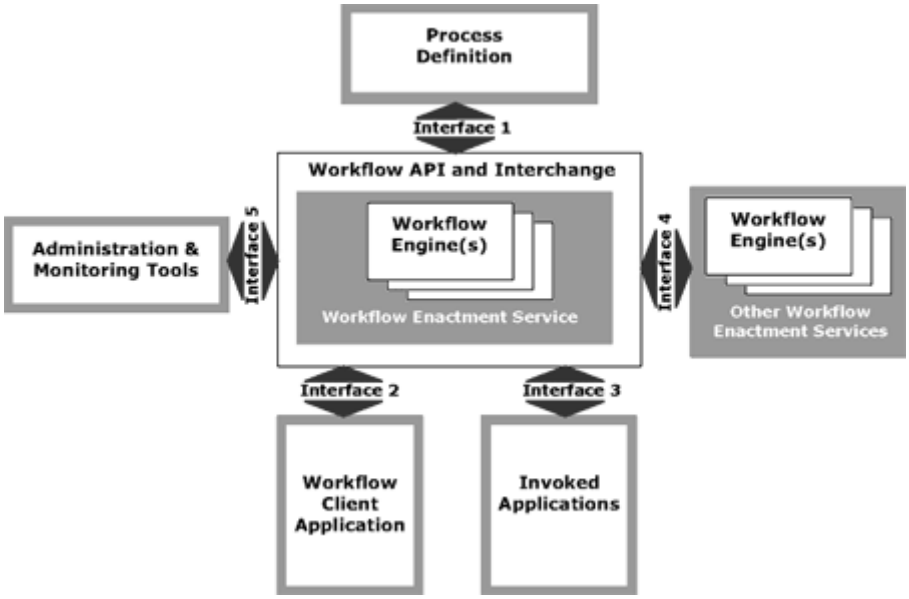

Figura 2 - Workflow Reference Model Diagram

In particolare, la macrofase relativa alla costruzione del workflow parte da un'analisi di Business Process Reengeneering per studiare ed ottimizzare la sequenza dei processi da definire e modellare. La macrofase successiva definisce invece il comportamento evolutivo dei processi, interpretandoli con soluzioni software che si occupano della creazione e del controllo delle istanze operazionali tramite la schedulazione delle attività e l'invocazione di appropriate risorse umane e tecnologiche. L'interazione con tali risorse viene, infine, gestita nell'ambito della terza macrofase, che si occupa delle soluzioni specifiche per gestire il trasferimento del controllo, l'evoluzione dello stato di processi ed attività, il passaggio dei dati e le interfacce.

Il supporto fisico allo sviluppo di workflow è garantito dai Workflow Enactment Services (WES), che consistono di uno o più motori capaci di creare, gestire ed eseguire le istanze di workflow. Le applicazioni possono interfacciarsi a tali servizi tramite le Workflow Application Programming Interface (WAPI). Nel

modello adottato dalla WfMC (Figura 1.4) si assume una separazione tra l'attività di controllo logico che costituisce il WES e la parte relativa agli strumenti applicativi ed agli utenti finali, che costituisce l'esecuzione associata a ciascuna attività. In un WES distribuito, diversi motori di workflow controllano ciascuno una parte della messa in opera del processo ed interagiscono con un sottoinsieme di utenti e strumenti applicativi relativi alle attività comprese nei processi dei quali sono responsabili.

Il motore di workflow è il componente runtime più importante del sistema di gestione. Solitamente è implementato su di un server e riceve istruzioni dagli altri componenti del sistema. Il suo ruolo principale consiste nello stabilire le priorità dei diversi compiti, porli nel giusto ordine ed assegnarli alle risorse. Per poter fare ciò deve essere in grado di interpretare le regole assegnate in fase di definizione del processo. Ciascun motore di workflow presente nel WES provvede alle funzioni di:

- interpretazione della definizione del processo;
- controllo delle istanze di processo;
- navigazione tra le attività di processo, gestendo parallelismo, sequenzialità, cicli, condizioni, dati e deadline;
- sign-on e sign-off degli specifici partecipanti;
- identificazione delle interazioni sulle interfacce utenti;
- mantenimento e passaggio dei dati connessi al workflow;
- chiamata di applicazioni e collegamento a dati esterni;
- controllo, amministrazione e verifica degli obiettivi.

Le interazioni con le risorse esterne al motore di workflow avvengono attraverso due tipi di interfacce:

- **Client Application**: interagisce con un worklist handler che organizza il lavoro di gestione delle risorse, selezionando e facendo progredire singoli lavori contenuti nella lista.
- **Invoked Application**: abilita il workflow engine a intraprendere una particolare attività, in genere priva di interfaccia utente. Nel caso si

utilizzino strumenti che richiedono interazione con l'utente, tale attività viene in genere invocata tramite un'interfaccia worklist per garantire maggiore flessibilità nella schedulazione delle operazioni.

Il motore di workflow avrà altre interazioni con gli strumenti di definizione e con quelli di monitoraggio dei processi. Sarà poi possibile per un WES interagire con analoghe architetture impegnate su altri workflow. Nel dettaglio, le client application di supporto all'enactment di un workflow offriranno:

- **Ambiente di modellazione**: tool grafici per la definizione del processo, tramite i quali vengono definiti l'inizio e la fine del processo, le attività intermedie con le relative ramificazioni, le attività che richiedono l'interfacciamento con altre applicazioni, le attività manuali. A seconda delle funzionalità del tool grafico, si può disporre di oggetti predefiniti che costituiranno la rappresentazione grafica del processo (ad esempio bottoni per spostare nodi e collegamenti, nodi per trasferimenti condizionati, nodi di rendez-vous, nodi di sottoprocesso). I tool di definizione possono far parte integrante del sistema di gestione di workflow o esserne separati. In quest'ultimo caso l'interfacciamento col motore di workflow avviene attraverso le WAPI.
- **Ambiente di sviluppo**: tool per la rappresentazione della mappa dei ruoli e dei gruppi gerarchici o collaborativi, per la definizione di dati, parametri e condizioni utilizzati nelle singole attività di un processo o rilevanti per il processo globale e per costruire l'applicazione di workflow, cioè programmi per gestire le informazioni che viaggiano attraverso il processo.
- **Ambiente per la gestione**: tool per il monitoring dei processi attivi (controllo dell'avanzamento), per la gestione dei processi e delle attività, per la generazione di statistiche sull'utilizzo delle risorse e sul tempo medio richiesto per completare task.
- **Ambiente runtime delle applicazioni client**: fornisce quello che l'utente vede quando usa un'applicazione di workflow. Comprende

l'accesso trasparente al database, alle applicazioni legacy, al sistema di gestione documentale, incluse le componenti per l'interfaccia della work list dell'utente coinvolto nel processo e tutti gli oggetti che il motore del workflow manipola.

Gli utenti del sistema utilizzano l'applicazione client per interagire col motore di workflow. Questo mette a disposizione diverse funzionalità:

- **Login**: il client identifica gli utenti, il ruolo da essi ricoperto ed effettua la connessione col motore di workflow.
- **Creazione di nuove istanze di processo**: col termine "istanza di un processo" si intende una particolare realizzazione di una tipologia di processo. L'avvio di un'istanza di processo è anche detto "istanziazione".
- **Richiesta o assegnazione di una nuova unità di lavoro**: a seconda che il sistema di gestione del workflow sia di tipo "push" o "pull", il client implementa una delle due funzionalità.
- **Accesso ai dati di applicazione necessari allo svolgimento del compito**: a questo scopo il client formula una query per accedere ai dati, in genere memorizzati su un database relazionale. Questi possono includere dettagli sull'oggetto del processo, documenti in formato immagine e in generale dati riguardanti la particolare istanza.
- **Marcatura di un'unità di lavoro**: sono disponibili delle opzioni che consentono all'operatore di marcare un'unità di lavoro come "completata" o, alternativamente, di sospenderla od inviarla ad un altro operatore in caso vi siano problemi.
- **Visione dello stato delle attività**: questa utilità consente agli operatori di verificare quali attività non sono state ancora completate.
- **Aggiornamento della definizione di un processo**: in alcuni software di workflow è possibile applicare questa funzione direttamente dal client. Essa consente, ad esempio, di cambiare gli attributi di un ruolo o di definire un nuovo processo.

A tali componenti normalmente si aggiungono le componenti software o hardware che permettono l'integrazione con applicazioni o prodotti specializzati (per esempio di archiviazione ottica, di firma digitale, strumenti di BPR).

2.2 Standard

La necessità di uno standard comune nasce dall'esigenza delle organizzazioni che sfruttano i prodotti per workflow di assicurare i loro investimenti e diventa una priorità quando si richiede ai propri sistemi di workflow di comunicare ed operare con quelli di altre organizzazioni.

I due standard che negli ultimi anni hanno conosciuto il più ampio sviluppo e sono stati supportati da vaste comunità di sviluppatori ed utilizzatori sono BPEL e XPDL. Il primo è un linguaggio derivato da XML che viene utilizzato per modellare l'esecuzione di processi che utilizzano Web Services, mentre il secondo è lo standard proposto direttamente da WfMC per la modellazione di un generico processo di business. Prima di analizzare nel dettaglio un possibile sistema per la composizione di servizi, è necessario definire questi standard ed illustrarne le caratteristiche principali. Un primo paragrafo viene dedicato a BPMN [6], una notazione standard che aumenta le capacità di interscambio dei processi di business fra varie entità, fornendo una notazione grafica ed un linguaggio per lo scambio di procedure.

2.2.1 BPMN

La Business Process Modeling Notation [7] è una notazione grafica standardizzata per rappresentare processi di business tramite workflow. Lo scopo principale di BPMN è quello di fornire una notazione standard che sia facilmente comprensibile a tutti coloro interessati a definire processi di business a qualunque livello: analisti, sviluppatori, tecnici o business manager. BPMN non è stato pensato semplicemente per modellare le applicazioni, ma anche i processi nei quali tali applicazioni vengono utilizzate. Di conseguenza BPMN è stato sviluppato con

l'intento di diventare un linguaggio comune per colmare il divario che spesso compare fra il design e la progettazione di processi di business e la loro seguente implementazione.

2.2.2 XPDL

La continua crescita della popolarità del linguaggio XML ed il suo utilizzo per la definizione di documenti, in combinazione con l'accrescere dell'esperienza nell'uso di WPDL [32] per i workflow e di tool BPM, ha portato alla creazione del linguaggio XPDL, la cui versione 1.0 è stata ufficialmente rilasciata da WfMC nell'Ottobre del 2002 [36]. XPDL ha mantenuto la semantica utilizzata in WPDL, ma ha definito una nuova sintassi basandosi su XML schema. Né WPDL né XPDL proposero una rappresentazione grafica specifica, nonostante il sottostante metamodello del processo fosse basato su una struttura grafica orientata, composta da attività come nodi e transizioni come rami e percorsi fra di essi. Per questo motivo, nell'Ottobre del 2005, è stato ufficialmente rilasciato XPDL 2.0 [35], una nuova versione del linguaggio che include una notazione grafica basata su BPMN ed un certo numero di meccanismi specifici per la modellizzazione dei processi, quali per esempio, lo scambio di messaggi e la gestione di eventi. XPDL 2.0 fornisce un formato standard per la creazione di diagrammi BPMN e per definire lo scambio d'informazioni fra i processi. Il formato del file è basato sul metamodello definito da WfMC ed illustrato nel paragrafo precedente. Lo schema di definizione del formato è estendibile e fornisce strumenti per la gestione delle estensioni sia da parte di venditori che utenti. Non si esclude, infatti, un mapping verso specifici linguaggi di esecuzione (es. BPEL) o altre specifiche basate su XML (eb-XML).

Attualmente ci sono più di 50 grandi aziende e sviluppatori di applicazioni che supportano il linguaggio XPDL, fra questi IBM, Oracle, BEA, Jujitsu, Tibco e Global 360. La dimostrazione del fatto che XPDL ormai è diventato uno standard utilizzato su larga scala, è che 8 fra gli 11 "top vendors" nella lista di Gartner "BPMS Magic Quadrant 11" [17] supportano XPDL. Inoltre, grazie alla stabilità che ha dimostrato in questi ultimi anni, XPDL è stato supportato anche da una

larga comunità di sviluppatori open source. Esso, infatti, a differenza di altri standard, è rilasciato in forma completamente gratuita senza alcuna limitazione.

Ad oggi XPDL è lo standard per lo scambio di informazioni sulla definizione di processi di business e sulla gestione dell'intero ecosistema relativo al design di processi. Lo standard forse non ha ancora ricevuto i meriti e l'attenzione che potrebbe meritare, ma soddisfa pienamente le esigenze per le quali è stato progettato. Esso, infatti, è uno standard di basso livello, che implementa la gestione di generici processi di business dal basso, permettendo all'utente di definire le singole e più basilari interazioni fra gli attori del processo stesso. Questa caratteristica da un lato lo rende molto flessibile ed adattabile ad ogni ambito d'applicazione, dall'altro rende più complessa la progettazione del sistema completo allo sviluppatore. Molto probabilmente è proprio questo uno dei motivi principali per cui non si è ancora affermato in ambito industriale come potrebbero essere le sue potenzialità.

Il modello di un processo di business è composto da una collezione di processi, applicazioni e risorse richieste per l'esecuzione di tutti i passi indicati nel flusso del processo. I due attori principali in questo flusso sono le Activity e le Transition

La definizione di un processo è composta da una o più Activity, ognuna delle quali rappresenta un'unità logica ed autonoma di funzionalità. Un'Activity rappresenta un'azione che può essere eseguita da un insieme di risorse o applicazioni software. Altre informazioni opzionali possono essere associate ad un'Activity, quali i parametri per l'invocazione automatica attraverso un WfMS oppure la sua priorità rispetto alle altre Activity presenti nel workflow. Un'Activity ha un ambito locale al processo nella quale è dichiarata, ma può anche fungere da container per l'invocazione di processi esterni (in questo caso si parla di Subflow Activity). Infine, un'Activity può anche essere di tipo "`dummy`", ovvero non eseguire nessuna operazione, ma essere presente nel workflow solo per facilitare il corretto indirizzamento delle precedenti operazioni.

Le Activity sono correlate tra di loro grazie all'uso di Transition. Ogni Transition è composta da 3 elementi principali: un campo "`from`", ovvero l'indicazione dell'Activity precedente, un campo "`to`", ovvero l'indicazione

dell'Activity successiva ed un campo "`condition`" contenente un'eventuale condizione di transizione. Le Transition, inoltre, possono essere eseguite sia in parallelo che in serie utilizzando "`split`" e "`join`". Questo approccio permette di controllare il flusso del workflow e di gestirne la sincronizzazione indipendentemente dalle operazioni che ogni singola Activity esegue.

2.2.3 BPEL

Il linguaggio BPEL è un linguaggio di esecuzione ed il suo scopo principale è quello di fornire una definizione ed uno standard per la gestione dei Web Service, definendo non solo la sequenza di interazioni, ma anche il flusso dei dati da un servizio all'altro. Nelle sue ultime versioni BPEL gestisce tutto quello che è lo scambio e la manipolazione di bit da un punto della rete all'altro. Non cerca tuttavia di rappresentare il percorso grafico che è stato realizzato per specificare il processo di gestione dei servizi.

BPEL costituisce, quindi, il linguaggio standard per la "process orchestration" ed è un componente essenziale per l'implementazione di una SOA. Esso indirizza l'esigenza di ridurre i costi e le complessità tipiche dei progetti di integrazione grazie all'eliminazione di codice proprietario e complesso ed alla maggiore capacità di rispondere rapidamente alle necessità di eseguire, modificare e consolidare i processi di business. L'aderenza ad uno standard ampiamente riconosciuto, come il BPEL, aumenta il valore strategico di quanto viene implementato, essendone garantita la completa portabilità. Oracle BPEL Process Manager è un esempio di un'applicazione che fornisce una soluzione basata su questo standard per l'implementazione di business process cross-application, consentendo la realizzazione di una Service Oriented Architecture (SOA). Include un motore di workflow in grado di supportare sia step di tipo automatico che step che richiedano l'intervento umano.

Senza entrare troppo nel dettaglio si illustrano di seguito le caratteristiche tecniche principali del linguaggio BPEL. Esso permette di descrivere un processo business mediante un insieme di attività, che possono essere semplici o strutturate. Le attività semplici esprimono una generica azione (ad esempio l'invocazione di un

servizio, la ricezione della risposta, l'assegnamento di un valore ad una variabile, la terminazione di un processo, ecc.), mentre quelle strutturate sono normalmente utilizzate per raggruppare attività semplici allo scopo di esprimere loop, operazioni condizionali, esecuzione sequenziale, esecuzione concorrente, ecc. L'intero processo è descritto mediante un'unica attività strutturata denominata "`top-level activity`", generalmente di tipo sequenziale.

BPEL mette altresì a disposizione dei costrutti per esprimere le cosiddette transazioni di lungo periodo (Long Running Transactions, LRT), che rappresentano un'estensione delle transazioni ACID al caso di processi di lunga durata mediante la nozione di compensazione delle attività eseguite. Il meccanismo della correlazione è utilizzato per tenere traccia di una certa conversazione a livello business, identificando così una sorta di sessione tra più partecipanti ad una stessa istanza di processo.

Un diagramma di workflow contiene tipicamente operazioni, messaggi, attori (umani o applicativi), applicazioni che definiscono il Web Service, condizioni logiche (if), parallelismi, loop e task di sincronizzazione fra operazioni.

BPEL è particolarmente adatto a modellare workflow completamente automatizzati, per la composizione di Web Service e l'integrazione di servizi che sono rappresentati tramite le applicazioni che li eseguono, eterogenee per hardware, architetture di rete e linguaggio del relativo codice.

BPEL consente di descrivere un workflow esistente oppure un processo astratto non eseguibile, trasformando in codice di programmazione una modellazione grafica che contiene la semantica descrivibile con i costrutti di UML. Questo è particolarmente utile per far comunicare software proprietari per la modellazione dei processi, che utilizzano terminologie ed icone differenti per rappresentare quanto può essere descritto con una notazione UML. BPEL permette di esportare e importare questi diagrammi in un file "`.bpel`" da un database proprietario all'altro senza perdere il contenuto della rappresentazione.

2.3 Composizione di Servizi e Workflow

Nei metodi per la composizione di servizi basati su workflow bisogna innanzitutto distinguere fra la generazione statica e dinamica dei workflow stessi. Con il termine "generazione statica" si intende che il gestore della richiesta del servizio deve costruire un modello astratto del processo prima che la composizione abbia inizio. Tale modello deve contenere un insieme di task e la descrizione dei dati da essi utilizzati. Ogni task deve anche contenere una query specifica che permetta al sistema di composizione di trovare un fornitore di servizio che sia compatibile e che possa soddisfare le richieste. In questo caso, solo la selezione e la composizione dei singoli servizi atomici è realizzata in modo automatico dal sistema. La composizione statica è utilizzata soprattutto nel caso in cui si possano definire i servizi del modello del processo tramite un tool grafico.

Con il termine "generazione dinamica", invece, si intende l'automatizzazione sia della creazione del modello che della selezione dei servizi atomici. Queste operazioni vengono effettuate grazie alla definizione da parte dell'utente di alcuni vincoli che includono le preferenze di esecuzione e le specifiche che devono soddisfare i vari servizi atomici.

In generale la composizione di servizi, sia essa statica o dinamica, prevede la realizzazione di un sistema con delle caratteristiche basilari comuni a qualunque ambiente venga applicato. Di seguito si illustrano le caratteristiche di un possibile sistema astratto ad alto livello, senza considerare un particolare linguaggio, una piattaforma di applicazione o un algoritmo usato per la gestione della composizione. Lo scopo di questa descrizione è quello di fornire una base teorica sulla quale poter sviluppare le considerazioni che caratterizzeranno i capitoli successivi: in particolare si utilizzeranno i concetti espressi in questo paragrafo cercando di fornire una soluzione personalizzata alle problematiche qui di seguito elencate.

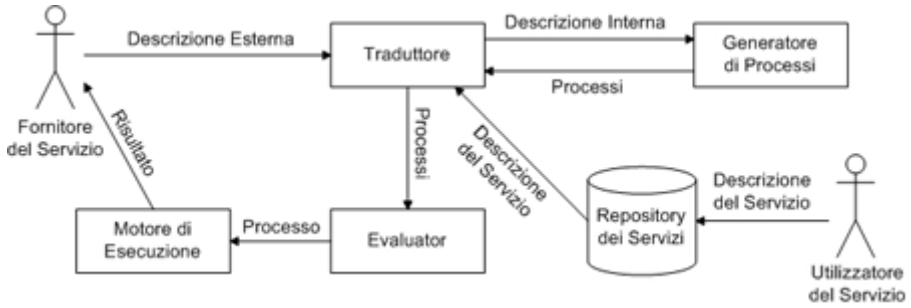

Figura 3 - Astrazione di un framewok per la composizione di servizi

In generale, un sistema per la composizione di servizi ha due attori principali, un fornitore ed un utilizzatore del servizio. Il primo mette a disposizione attraverso una tecnologia specifica (in questo lavoro di tesi siamo interessati a servizi esposti tramite Web Service) un determinato servizio, il secondo consuma le informazioni e le risorse messe a disposizione. Il sistema prevede anche i seguenti componenti: un traduttore, un generatore di processi, un misuratore di prestazioni, un motore di esecuzione ed uno o più repository di servizi. Il traduttore effettua una traduzione tra i linguaggi esterni utilizzati dai partecipanti ed i linguaggi interni utilizzati dal generatore del processo. Per ogni richiesta il generatore di processi cerca di creare un piano, componendo i servizi disponibili nel repository. Se vengono trovati più piani validi, il ragionatore li valuta ed estrae quello che per caratteristiche e potenzialità viene considerato il migliore per soddisfare la richiesta. Il motore di esecuzione pone il piano in esecuzione e ritorna il risultato dell'esecuzione stessa al fornitore del servizio. Per entrare maggiormente nel dettaglio, il processo di composizione automatica di servizi si compone delle seguenti fasi:

- PRESENTAZIONE DEL SINGOLO SERVIZIO: per prima cosa i vari fornitori devono rendere pubblico il loro servizio atomico. Per fare ciò esistono diversi linguaggi descrittivi, quali, per esempio, UDDI o DAML-S [8]. Questi linguaggi caratterizzano un Web Service attraverso una serie di attributi predefiniti, quali una firma, input, output ed eccezioni, formato dei dati implicati nel servizio, precondizioni e postcondizioni, ecc. Possono anche essere inclusi degli

attributi non funzionali, ma comunque fondamentali in fase di esecuzione, quali il costo, la qualità o la sicurezza.

- TRADUZIONE FRA I LINGUAGGI: molti sistemi di composizione di servizi effettuano una netta distinzione fra i linguaggi di specifica dei servizi interni ed esterni. Il linguaggio usato dai servizi esterni può essere eterogeneo e deve essere semplice ed uniformemente conosciuto per favorire l'accessibilità. Esso differisce dal linguaggio usato internamente dal generatore dei processi, il quale richiede un linguaggio più preciso e formale e, spesso, dipendente dall'ambiente di sviluppo ed esecuzione. Per questo motivo è fondamentale l'implementazione ad hoc di un traduttore che permetta ai servizi atomici, che utilizzano uno standard globale, di essere inseriti ed utilizzati all'interno del sistema di composizione specifico.
- GENERAZIONE DEL MODELLO DEL PROCESSO: chi richiede un determinato servizio esprime le proprie necessità in un linguaggio di descrizione del servizio stesso. Il generatore del processo, quindi, cerca di soddisfare le richieste, componendo i servizi atomici messi a disposizione dai fornitori. Il generatore del processo riceve in ingresso la descrizione delle specifiche e fornisce in uscita il modello che descrive il servizio composto. Il modello del processo contiene l'insieme dei servizi atomici utilizzati ed il flusso dei controlli e dei dati che vengono scambiati tra di essi.
- VALUTAZIONE DEL SERVIZIO GLOBALE: è abbastanza comune che in un ambiente distribuito ed eterogeneo alcuni servizi abbiamo delle caratteristiche simili. Per questo motivo non è da escludere che il pianificatore generi più di un piano che soddisfi i requisiti in ingresso. In questo caso i servizi atomici vengono valutati in funzione delle loro caratteristiche globali, usando le informazioni contenute negli attributi non funzionali. Chi richiede il servizio può specificare un peso per ogni attributo non funzionale, così che il servizio che ottiene un peso totale più alto venga scelto come migliore.

- ESECUZIONE DEL SERVIZIO: una volta che il processo composto è stato creato e selezionato, il servizio è pronto per essere eseguito. Il flusso di dati del processo composto è definito come la sequenza di azioni e dati che il servizio in esecuzione invia come input verso il successivo servizio atomico.

2.4 Web Services

Nel paragrafo precedente si è fatto riferimento a servizi distribuiti ed eterogenei. La tecnologia che meglio rappresenta questo tipo di servizi e che verrà utilizzata come base per la fornitura di applicazioni nel sistema descritto nei prossimi capitoli è quella dei Web Services. I Web Services possono essere definiti genericamente come dei servizi offerti attraverso il Web. Lo scenario tipico d'utilizzo di un Web Service è quello in cui un'applicazione di business invia una richiesta ad un servizio specificando un URL, impacchettandola attraverso il protocollo SOAP su HTTP. Il Web Service riceve la richiesta, la processa e ritorna una risposta. Sia chi pubblica sul Web i propri servizi che chi li utilizza sono realtà di business per questo, di fatto i Web Services trovano la loro naturale collocazione nella realizzazione di reti B2B.

La vera rivoluzione portata dai Web Services è stata quella di trasformare il Web da una rete di contenuti in una rete di servizi applicativi. Tramite lo standard definito per la caratterizzazione dei Web Services, è possibile utilizzare una serie di servizi ed applicazioni sviluppati e forniti da terzi, senza doversi preoccupare di quale sia la tecnologia utilizzata per realizzare il servizio, così come il fornitore del servizio non dovrà preoccuparsi della natura del client che si collegherà. Il tutto richiede soltanto che il servizio sia registrato su di un registro pubblico di servizi (UDDI Registry) nel formato standard WSDL.

Attualmente i Web Services possono essere applicati a tre diversi livelli di astrazione per soddisfare le esigenze di ogni tipo di applicazione:

- per l'integrazione all'interno dell'azienda (legacy integration and trasformation) tra sistemi e hardware eterogenei, su reti sicure ed utenti controllabili.
- Per integrare tra di loro aziende attraverso la rete, supply chain management, customer relationship management, ecc.
- Per realizzare sistemi ad integrazione lasca cooperanti su Internet. Questo avviene gestendo in modo centralizzato tutti gli aspetti di sicurezza, gestione delle transazioni, realizzazione di workflow.

L'aspetto fondamentale che rende i Web Services una tecnologia così allettante ed interessante è legato principalmente a due loro proprietà:

- self-describing, il servizio fornisce la descrizione delle caratteristiche offerte tramite una grammatica XML;
- discoverable, meccanismo di pubblicazione/identicazione di un Web Service tramite la consultazione di registry dedicati (UDDI).

Ci sono due modi di descrivere l'architettura di un Web Service: il primo è quello di esaminare le singole regole che li costituiscono; il secondo è quello di esaminare i protocolli che li compongono.

Le tre entità che costituiscono l'architettura di Web Service sono:

- **`Service Provider`**, per implementare e renderlo disponibile.
- **`Service Requestor`**, per invocarlo tramite delle richieste XML.
- **`Service Registry`**, per contenere le informazioni su come localizzarlo in base a delle chiavi di ricerca.

Dall'altro punto di vista possiamo esaminare l'insieme di protocolli:

- **`Service Transport`**, responsabile del trasporto dei messaggi tra le applicazioni; attualmente include HTTP, SMTP, FTP.
- **`XML Messaging`**, responsabile della codifica dei messaggi in formato XML; attualmente questo layer include anche XML-RPC e SOAP.
- **`Service Description`**, responsabile della descrizione delle interfacce da pubblicare per uno specifico Web Service; attualmente è gestito da Web Service Description Language (WSDL).

- **`Service Discovery`**, responsabile della centralizzazione dei servizi in un registro comune e fornitore delle funzionalità di publish/find. Attualmente, il servizio di discovery è gestito da UDDI.

In Figura 4 è rappresentata in modo schematico l'architettura di un Web Service come appena descritto.

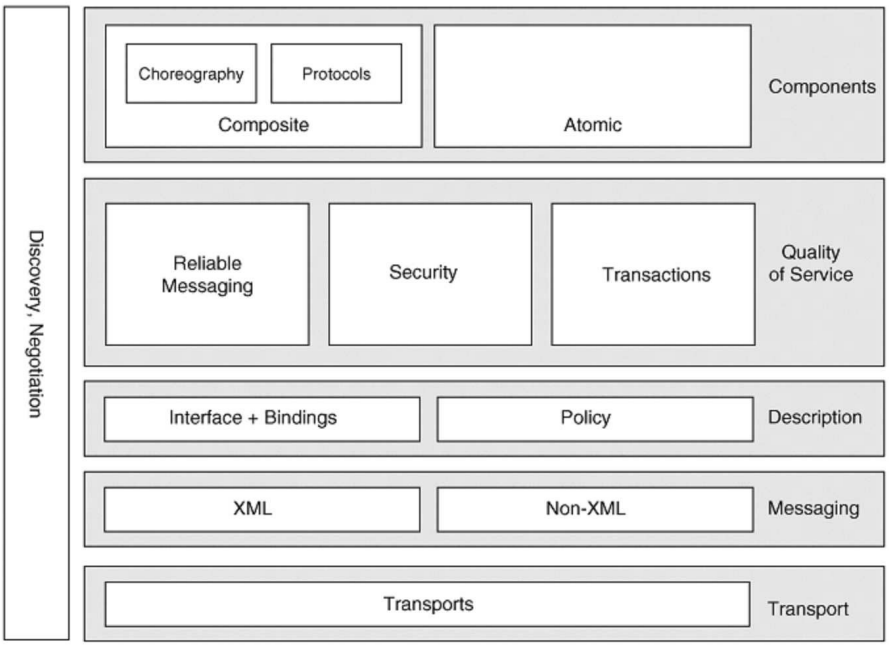

Figura 4 - Architettura di un Web Service.

2.5 Quale Standard?

Molti ricercatori al giorno d'oggi sostengono che BPEL e XPDL siano in diretta concorrenza. In realtà questo non è vero, in quanto essi sono due standard completamente diversi che si applicano a due realtà altrettanto diverse. Mentre BPEL è un "execution language", ovvero un linguaggio di programmazione completo di variabili ed operandi, XPDL è un linguaggio per il "design" di un

processo, con funzionalità di basso livello per la gestione dei nodi e delle transizioni. Lo scopo di BPEL è quello di fornire una definizione per la gestione dei Web Services, per gestirne le interazioni e lo scambio di dati, mentre XPDL ha l'obiettivo di memorizzare e pubblicare il diagramma rappresentativo di un processo.

Questi standard, quindi, non sono in diretta concorrenza fra loro e possono essere utilizzati tutti all'interno dello stesso sistema, seguendo la proposta identificata sotto il termine di "Value Chain" [25]. E' possibile, infatti, iniziare a progettare un processo di business disegnando un generico diagramma BPMN, quindi, in fase di sviluppo, salvare i vari diagrammi funzionali in XPDL ed infine, tradurne in BPEL quelle parti che richiedono uno scambio di dati, in particolare attraverso l'interazione di un individuo.

Nei prossimi capitoli si illustrerà un sistema che utilizza il linguaggio XPDL come unico standard per la realizzazione di processi di business: tale scelta è stata dettata dalla maggiore flessibilità che questo linguaggio ha dimostrato rispetto agli altri. In versioni future si potrebbe estendere il sistema in modo da utilizzare anche il linguaggio BPEL, in modo simile a quanto appena descritto.

3
La Piattaforma Eclipse

Questo capitolo presenta la base tecnologica che si è scelto di utilizzare per lo sviluppo delle applicazioni descritte nei successivi capitoli. In particolare si illustreranno le principali funzionalità della piattaforma Eclipse per quel che riguarda l'implementazione di tool in linguaggio Java.

3.1 Introduzione

Lo sviluppo di applicazioni basate su di una struttura complessa a più livelli ha bisogno di una varietà di tool da fornitori diversi per sostenere il pieno sviluppo del ciclo di vita del software. Gli sviluppatori possono essere più produttivi ed efficaci se questi strumenti funzionano bene insieme. Ambienti di sviluppo integrato (IDE) possono aiutare nell'intento di fornire un supporto per facilitare il processo di progettazione e generazione del software. In questo senso è stata progettata la piattaforma Eclipse [10], supportata da una comunità open source i cui progetti sono rivolti alla creazione di una piattaforma di sviluppo estensibile e di framework applicativi per la creazione di software. Eclipse offre framework e strumenti estensibili in grado di gestire l'intero ciclo di vita del software, incluso il supporto per la modellazione, ambienti di sviluppo per i linguaggi Java, C/C++ ed altri, ambienti di test e valutazione delle prestazioni, business intelligence, applicazioni client avanzate e per lo sviluppo embedded. La piattaforma Eclipse viene estesa, integrata e supportata da un ampio e vivace ecosistema formato da importanti

produttori di tecnologie, start-up innovative, università, enti di ricerca e singoli utenti.

3.1.1 Perché un IDE?

Agli albori della programmazione, gli unici tool di cui gli sviluppatori realmente avevano necessità erano un compilatore per il linguaggio utilizzato ed un "loader" per combinare insieme i file compilati a formare un programma eseguibile. Con il passare del tempo le necessità si sono moltiplicate e le esigenze del mercato richiedono una complessità sempre crescente. La crescita della rete Internet e la globalizzazione del mercato hanno portato, inoltre, ad avere a disposizione una base di conoscenza e sperimentazione inimmaginabile solo qualche decennio fa. Per questo motivo la creazione di un ambiente integrato di sviluppo all'interno del quale poter racchiudere il maggior numero di tool e risorse utili al programmatore ha attratto l'attenzione di molti sviluppatori ed ha portato alla nascita del termine IDE.

L'uso di un IDE, quindi, è fondamentale per un primo, unico e semplice motivo: gli sviluppatori di software diventano più produttivi. Proprio grazie a questo indubbio valore, gli ambienti integrati di sviluppo continueranno ad evolversi seguendo le nuove metodologie che verranno inventate per migliorare la produttività. Nel tempo, l'insieme di strumenti integrati in un IDE è andato sempre più crescendo, partendo da un semplice editor con compilatore e debugger fino ad arrivare ad includere compilatori incrementali, browser e gestori delle risorse per migliorare la presentazione all'utente del proprio progetto, editor visivi per la creazione di parti grafiche ed interfacce utente, ecc. Questa tendenza verso più sofisticati e potenti strumenti specifici per un determinato linguaggio continua ed oggi comprende strutture ben più complesse ed integrate.

Per aumentare ulteriormente la produttività un buon IDE deve coprire il maggior numero possibile degli aspetti del ciclo di vita del software. Per esempio, molti IDE commerciali includono opzionalmente la gestione della versione e la configurazione del codice sorgente, un aspetto cruciale per i programmatori, soprattutto nel caso in cui siano organizzati in gruppi di lavoro. Gli IDE, infine, si

stanno velocemente espandendo verso altre aree d'interesse, quali, per esempio, il software design tramite l'uso di UML o strumenti per la modellizzazione.

Nella realtà, i programmatori non si limitano solamente a scrivere e debuggare il codice in un singolo linguaggio di programmazione. È un luogo comune per un programmatore dover creare e manipolare artefatti non direttamente collegati al proprio codice, come per esempio pagine HTML o documenti testuali. Questo significa che il programmatore avrà comunque bisogno di tool non integrati nell'IDE e, quindi, verrà a far cadere tutti i vantaggi ottenuti dal suo uso. Per risolvere questo potenziale problema, la maggior parte degli IDE commerciali sono stati progettati con una struttura aperta ed estensibile in modo che terze parti possano integrare i propri tool. Questa caratteristica viene fornita attraverso un meccanismo attivato allo startup, per cui l'IDE è in grado di identificare i tool che sono presenti e mettere a disposizione delle API pubbliche per questi tool affinché possano integrare le loro funzioni al proprio interno. Per esempio Borland JBuilder ha un proprio Open Tools API, così come IntelliJ IDEA commercializza un prodotto denominato Plug-in API. Questo tipo di estensibilità è fondamentale nell'ambito della progettazione di IDE perché qualunque produttore o rivenditore di IDE, pur grande e rinomato che sia, non sarà mai in grado di fornire un insieme di tool sufficiente a soddisfare le esigenze di tutti i suoi utilizzatori. Quali tool di terze parti verranno inclusi all'interno di un determinato IDE verrà in questo modo deciso dal mercato.

Alcuni IDE partono da una base in un linguaggio specifico e si espandono da essa. Oracle JDeveloper Suite è un IDE Java-centric che è stato espanso con un editor di modelli UML, argomento ben al di là della pura programmazione in Java. NetBeans è stato creato come un IDE specifica per Java, ma ora si è evoluto verso una piattaforma open source ed indipendente dal linguaggio che è inclusa anche in IDE commerciali quali Sun ONE Studio. Microsoft Visual Studio .NET supporta svariati linguaggi all'interno dello stesso IDE e fornisce delle API sia generiche che specifiche per un linguaggio. Nello sforzo di ampliare i propri orizzonti gli IDE devono contrastare ogni pregiudizio verso uno specifico linguaggio ed espandersi verso un supporto universale e neutrale.

3.2 Eclipse

La piattaforma Eclipse è un IDE aperto ed indipendente dal linguaggio di programmazione. La prima versione open source della piattaforma fu rilasciata alla fine del 2001 ed iniziò ad apparire all'interno di prodotti commerciali subito dopo, partendo da WebSphere Studio Application Developer 4.0 di IBM.

Nonostante la piattaforma Eclipse abbia numerose funzionalità al proprio interno, esse rimangono sempre molto generiche. Necessita, quindi, di tool addizionali che la estendano per supportare nuovi contenuti, per gestire con maggiori funzionalità quelli già esistenti e per indirizzare le funzionalità generiche verso obiettivi più specifici.

La piattaforma Eclipse è basata su di un meccanismo per la scoperta, l'integrazione e la gestione di nuovi moduli chiamati plug-in. Lo sviluppatore di un tool scrive la propria applicazione come un plug-in separato che agisce su una serie di file all'interno del workspace e costruisce eventualmente la propria interfaccia utente su quella fornita dal workbench. Quando viene lanciata la piattaforma, all'utente viene presentato un IDE composto dall'insieme dei plug-in disponibili. Il livello di soddisfazione dell'utente dipende fortemente da come tali plug-in si integrano con la piattaforma e da come funzionano in modo combinato fra loro.

L'intera piattaforma Eclipse è distribuita sotto la licenza EPL, basata sulla licenza CPL, una delle licenze approvate a livello mondiale dalla OSI. Questa licenza priva di royalty permette a chiunque di utilizzare e ridistribuire Eclipse sia per progetti commerciali che non.

3.2.1 Caratteristiche

La piattaforma Eclipse è stata progettata e sviluppata per soddisfare le seguenti specifiche:

- supportare la creazione di tool per lo sviluppo di applicazioni.
- Supportare un insieme aperto di sviluppatori, inclusi venditori indipendenti di software.

- Supportare tool per manipolare liberamente vari tipi di contenuti (HTML, Java, C/C++, JSP, EJB, XML, Gif, ecc.).
- Facilitare l'integrazione di strumenti all'interno e tra i diversi tipi di tool e fornitori.
- Supportare ambienti di programmazione sia con GUI che senza.
- Funzionare nel più vasto range possibile di sistemi operativi.
- Basarsi sulla popolarità e funzionalità del linguaggio Java per l'integrazione di nuovi plug-in.

Lo scopo principale della piattaforma Eclipse è quello di fornire agli sviluppatori dei meccanismi e delle regole da seguire per l'integrazione ottimale dei propri tool. Questi meccanismi vengono resi pubblici attraverso delle ben definite interfacce API, classi e metodi. La piattaforma aggiunge a tutto ciò anche una serie di blocchi preconfezionati e framework per facilitare lo sviluppo di tool specifici. La figura 2 rappresenta i componenti principali e le API della piattaforma.

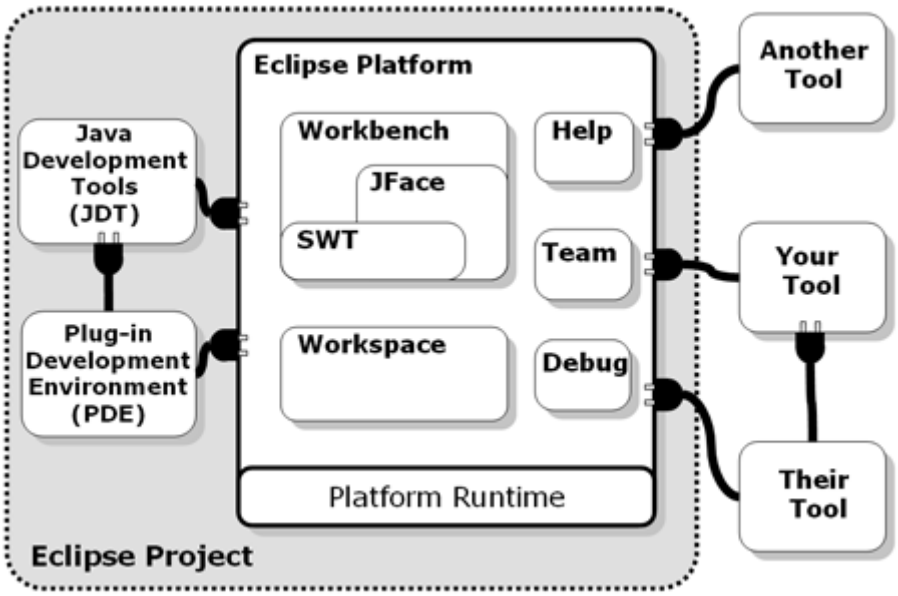

Figura 5 – Architettura della piattaforma Eclipse

3.2.2 Runtime ed Architettura dei Plug-in

Un plug-in è la più piccola unità funzionale della piattaforma Eclipse e può essere sviluppato ed integrato separatamente. Normalmente un singolo plug-in implementa un piccolo tool od una limitata funzionalità, mentre tool più complessi condividono le proprie caratteristiche fra più plug-in. Fatta eccezione per il kernel, ovvero il Runtime della piattaforma, tutte le funzionalità sono racchiuse in plug-in.

I plug-in sono interamente scritti in Java. Un classico plug-in è composto da un file "`.jar`" che comprende il codice Java compilato, alcuni file di configurazione (`plugin.xml`, `manifest.mf`) e tutte le risorse utilizzate quali immagini, Web template, librerie di terze parti, ecc. Possono esistere anche plug-in che non contengono alcun codice Java: un esempio può essere un plug-in che fornisce un help in linea sotto forma di pagine HTML. Ogni plug-in è localizzato in una cartella riservata all'interno del file system, normalmente all'interno della cartella "`plugins`" nella cartella d'installazione della piattaforma Eclipse. Esiste anche un meccanismo attraverso il quale un singolo plug-in può essere suddiviso in diversi segmenti ognuno localizzato all'interno della propria cartella. Questo meccanismo è utilizzato, per esempio, per la localizzazione di un plug-in in diverse lingue ed è conosciuto con il termine Plug-in Fragments.

Ogni plug-in ha un proprio manifest file che ne dichiara le interconnessioni con gli altri plug-in e ne caratterizza le funzionalità. Il modello che regola queste interconnessioni è estremamente semplice: un plug-in dichiara un certo numero di "`extension point`" ed "`extension`" di extension point dichiarati da altri plug-in. Un extension point può avere le proprie API specifiche. Un plug-in che ne voglia sfruttare od estendere le caratteristiche deve implementare l'interfaccia fornita da questo extension point. Qualunque plug-in può contribuire a queste preferenze definendo delle extension per questo extension point.

Ogni volta che viene avviata la piattaforma Eclipse il Runtime ricerca l'insieme dei plug-in disponibili all'interno della cartella "`plugins`", ne legge il file manifest e si costruisce un registro interno degli stessi. Il Runtime controlla tutte le corrispondenze fra le dichiarazioni di extension point e le rispettive extension.

Qualunque problema si dovesse verificare, quale per esempio la mancanza di alcune corrispondenze, viene rilevato e memorizzato. Il registro dei plug-in che viene creato al termine di questa fase è disponibile attraverso un'API pubblica. Dalla versione 3.3 della piattaforma Eclipse, denominata Europa, è disponibile un meccanismo per la rilevazione di plug-in anche dopo che la piattaforma è stata avviata.

Il plug-in manifest è un file XML. Un extension point all'interno del manifest file di un plug-in può dichiarare ulteriori elementi XML da utilizzare come extension. Questo permette all'extension di comunicare le informazioni necessarie al plug-in originario. Inoltre, l'informazione contenuta nel manifest è disponibile all'interno del registro dei plug-in senza la necessità di attivare il plug-in stesso o caricare qualche parte del suo codice. Questo aspetto è fondamentale per il corretto funzionamento di una piattaforma complessa basata su un grande numero di plug-in, in quanto verranno attivati e resi disponibile a runtime solamente quelli realmente utilizzati o richiesti dall'utente. Il caricamento di un plug-in occupa risorse in memoria ed ha un impatto non trascurabile sul tempo di avvio dell'intera piattaforma. L'utilizzo di un manifest file codificato in XML, inoltre, rende più semplice sviluppare tool che supportino la creazione di plug-in: un esempio è dato dal PDE, un framework per lo sviluppo di plug-in integrato nativamente all'interno della piattaforma Eclipse. Un plug-in è attivato solamente quando è necessario, ovvero quando una funzione da lui integrata viene richiesta dall'utente. Una volta attivato il plug-in utilizza il registro dei plug-in per conoscere ad accedere alle extension che eventualmente contribuiscono ai suoi extension point. Per esempio, il plug-in che dichiara un extension point per le preferenze dell'utente può controllare tutti i contributi che sono stati forniti da terze parti ed accedere alle loro dichiarazioni per creare una finestra di preferenze personalizzata. Questo viene effettuato utilizzando semplicemente le informazioni contenute nel registro, senza la necessità di attivare nessuno dei contributi. Il plug-in verrà attivato solamente quando l'utente selezionerà un campo all'interno della finestra. In questo modo l'attivazione dei plug-in non avviene in automatico; esistono però delle semplici API che ne permettono l'attivazione automaticamente. Una volta che il plug-in

viene attivato rimane in questo stato fino a che non viene chiusa la piattaforma Eclipse. Ogni plug-in ha la possibilità di memorizzare dati relativi al suo funzionamento all'interno di una sottocartella del proprio percorso: questo meccanismo permette al plug-in di mantenere le informazioni necessarie tra un'esecuzione e l'altra della piattaforma.

Il Runtime della piattaforma dichiara un extension point speciale per le applicazioni. Quando viene eseguita un'istanza della piattaforma, il nome dell'applicazione viene specificato all'interno del comando di esecuzione; gli unici plug-in che vengono attivati sono quelli che dichiarano un'estensione per quella specifica applicazione.

Determinando l'insieme dei plug-in disponibili in fase di inizializzazione e supportando lo scambio d'informazioni fra i plug-in senza necessità di attivarli, il Runtime permette alla piattaforma di arricchire ogni plug-in con una serie di informazioni pertinenti al contesto nel quale sarà eseguito. Questo contesto non cambierà mai durante l'esecuzione della piattaforma, quindi non è necessario avere un sistema di gestione del ciclo di vita dei plug-in. In questo modo si mantiene minima la lunghezza della sequenza di avvio, così come si elimina la possibilità di generare dei bug derivanti da imprevedibili catene di attivazione dei plug-in.

La piattaforma Eclipse è eseguita all'interno di una singola invocazione della Java Virtual Machine. Ad ogni plug-in è associato un Java class loader specifico, che è l'unico responsabile del caricamento delle sue classi e di eventuali risorse da lui richieste. Ogni plug-in dichiara esplicitamente le sue dipendenze da altri plug-in dei quali può invocare direttamente le classi. È il plug-in stesso che, attraverso la dichiarazione nel file manifest, controlla la visibilità delle sue classi ed interfacce pubbliche. Queste regole di visibilità vengono gestite a runtime dal class loader del plug-in.

Il meccanismo a plug-in finora descritto viene utilizzato per frazionare le funzionalità della piattaforma Eclipse. L'intera struttura viene gestita attraverso l'Update Manager, un applicativo che permette di scaricare ed installare nuove funzionalità o versioni più aggiornate di quelle già presenti. L'Update Manager costruisce una nuova configurazione dei plug-in che saranno disponibili nella

successiva esecuzione della piattaforma. Un servizio di ripristino di configurazioni precedenti è stato implementato nelle ultime versioni.

Nei prossimi paragrafi vengono analizzati i principali componenti della piattaforma che saranno parte integrante delle applicazioni introdotte nei capitoli successivi.

3.2.2.1 Workspace

Tutti i tool presenti all'interno della piattaforma Eclipse che operano su file, funzionano all'interno del Workspace. Il Workspace è composto da uno o più progetti ognuno dei quali è mappato su una specifica directory all'interno del file system. I vari progetti presenti all'interno del Workspace possono essere inseriti all'interno di diverse directory nel file system, ma sono mappati tutti come sottodirectory della directory principale del Workspace.

Ogni progetto contiene i file che sono stati creati e manipolati dall'utente. Tutti i file all'interno del Workspace sono anche accessibili dai comuni programmi installati sul sistema operativo, ma i tool integrati all'interno della piattaforma Eclipse ne permettono una gestione migliore, ottimizzata e centralizzata. Seguendo il classico approccio che domina tutta la piattaforma Eclipse, anche le risorse nel Workspace sono degli oggetti modificabili, in modo che possano essere manipolati ed estesi da terze parti.

La piattaforma Eclipse comprende un meccanismo generale che permette ad un tool di tenere traccia dei cambiamenti che avvengono all'interno del Workspace. Tramite l'uso di un Listener, un qualunque tool è in grado di ricevere notifiche riguardo alle operazioni che sono state effettuate su di una specifica risorsa e di conseguenza reagire a tale cambiamento.

3.2.2.2 Workbench

L'intera interfaccia grafica della piattaforma Eclipse è costruita attorno al Workbench che ne gestisce la struttura e presenta un'interfaccia utente estensibile (Figura 6). L'intero Workbench è basato sulla libreria grafica SWT ed il framework JFace.

Il paradigma che gestisce l'interfaccia grafica della piattaforma Eclipse è basato sul concetto di `Editor`, `View` e `Perspective`. L'Editor è il fulcro della piattaforma e permette all'utente di aprire, modificare e salvare oggetti. La piattaforma fornisce un Editor testuale nativo, mentre altri tipi di Editor sono implementati attraverso plug-in. Le View hanno una serie molto ampia di funzionalità e sono utilizzate per fornire informazioni all'utente riguardo alle azioni che compie all'interno della piattaforma. La Perspective definisce la disposizione degli Editor e delle View all'interno del Workbench, mostrando solamente quelle utili in quel particolare ambito in cui si sta operando. La piattaforma può avere a disposizione più di una Perspective, ma solamente una di esse può essere attiva in un determinato momento.

Le applicazioni sviluppate per la piattaforma Eclipse sottostanno tutte a questo paradigma "Editor, Views, Perspective", nel senso che il Workbench può essere migliorato e customizzato inserendo nuovi Editor, nuove View o nuove Perspective che, a loro volta, possono essere combinate insieme per garantire ulteriori funzionalità od estenderne alcune già presenti.

E' la piattaforma che si prende cura di tutti gli aspetti del Workbench: gli Editor e le View vengono istanziati al momento della richiesta da parte dell'utente e deallocati nel momento in cui non sono più necessari. Tutte le risorse associate sono gestite con lo stesso paradigma.

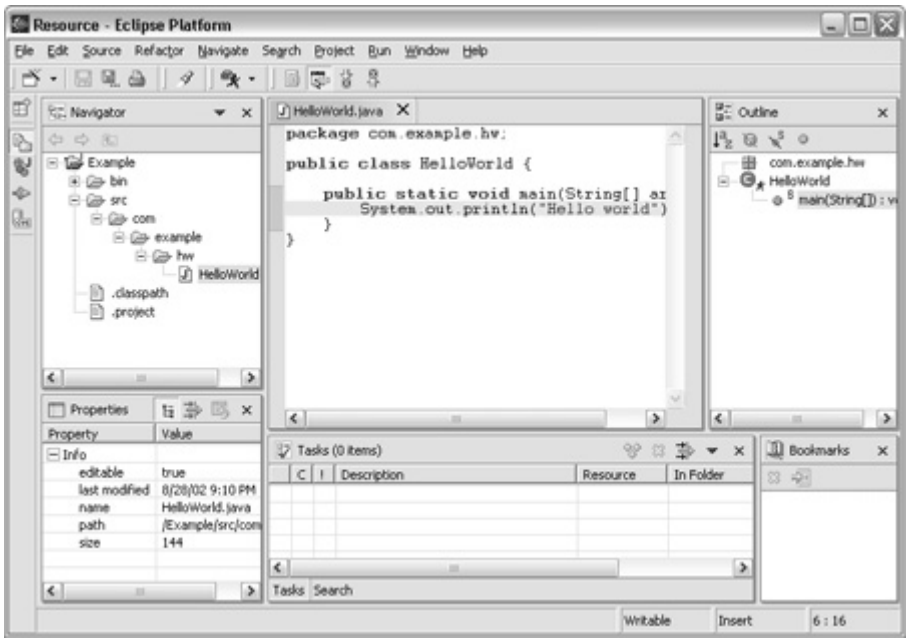

Figura 6 - Eclipse Workbench

3.3 SWT

Lo Standard Widget Toolkit (SWT) è una libreria che fornisce una serie di API indipendenti dal sistema operativo per la creazione di widget e parti grafiche [30]. Tali API sono state sviluppate per permettere una più stretta integrazione con le funzionalità del sistema operativo rispetto ad AWT, la libreria grafica standard di Java. L'intera piattaforma Eclipse e tutti i suoi plug-in sono stati sviluppati utilizzando SWT.

Un perenne aspetto di discussione nel disegno di widget grafici è la gestione del rapporto fra applicazioni portabili (caratteristica intrinseca in un programma Java) e la loro integrazione con il sistema a finestre nativo del sistema operativo. La libreria AWT di Java mette a disposizione del programmatore dei widget di basso livello quali campi di testo, bottoni e liste, ma non widget di più alto livello quali strutture ad albero, testo arricchito, ecc. I widget di AWT sono implementati

attraverso widget nativi su tutti i sistemi operativi. Costruire un'applicazione grafica utilizzando il solo AWT, significa obbligare il programmatore ad utilizzare solamente quelle proprietà che sono comuni a tutti i sistemi operativi. Il toolkit Java Swing tenta di risolvere questo problema fornendo dei widget quali alberi, tabelle, ecc. ed emulando il look-and-feel del sottostante sistema operativo. Nonostante questi sforzi l'esperienza grafica che si ottiene dall'utilizzo di Swing rimane differente a seconda del sistema sul quale viene eseguita l'applicazione e spesso non mantiene correttamente lo stesso comportamento.

SWT si propone di risolvere questo annoso problema definendo un'API generica ed indipendente dal sistema operativo sul quale verrà eseguita. Per ogni sistema operativo l'implementazione di SWT utilizzerà, laddove possibile, widget nativi. Nel caso in cui questi widget non siano disponibili SWT utilizza un sistema di emulazione simile a quello di Swing.

SWT fornisce anche una serie di API specifiche per ogni sistema operativo per dare supporto in quei casi nei quali il sistema fornisca una caratteristica particolare e significativa che non sia comune anche agli altri: un esempio è l'uso di ActiveX all'interno di Windows.

Per poter sfruttare appieno le librerie grafiche native del sistema operativo, SWT necessita di alcuni file specifici da installare sul sistema operativo all'interno del quale l'applicazione dovrà funzionare: tali file sono .dll in Windows oppure .so in Linux.

3.3.1 Perché SWT?

In questo paragrafo si vuole dare un brevissimo riscontro ad una comparativa che è stata effettuata fra la libreria SWT, messa a disposizione nella piattaforma Eclipse, e la libreria standard di Java, Swing. Quello che si vuole fornire non è una discussione definitiva su quale libreria sia meglio utilizzare, ma semplicemente illustrare alcuni vantaggi e svantaggi che si sono riscontrati dall'uso di SWT.

3.3.1.1 Vantaggi

Il vantaggio principale di SWT è la migliore integrazione dell'applicazione all'interno del sistema operativo. Questo è dovuto soprattutto all'uso dei file specifici che si devono installare per il suo funzionamento: la libreria, infatti, non emula l'interfaccia nativa del sistema come Swing, ma ne utilizza direttamente i widget nativi, facendo semplicemente da ponte con l'interfaccia utente. Le applicazioni sviluppate con SWT non si distinguono agli occhi dell'utilizzatore dalle altre normali applicazioni che è abituato ad utilizzare e questo comporta un maggiore e più rapido feeling con il nuovo programma.

Un ulteriore indubbio vantaggio è legato ad una migliore interazione e rapidità di risposta. SWT, infatti, utilizza il sistema di esecuzione degli eventi nativo del sistema operativo. Questo si rispecchia in una più veloce risposta ai comandi dell'utente ed una minore richiesta di risorse per il proprio funzionamento.

Infine, grazie al richiamo dei widget nativi, si può confidare sul fatto che SWT sia più solido e tollerante agli errori dovuti all'eterogeneità dell'hardware e dei processori grafici sul quale viene eseguito.

3.3.1.2 Svantaggi

Il grosso svantaggio nell'uso di SWT è legato al fatto che il suo funzionamento è garantito solamente per i sistemi operativi per i quali è stato realizzato un supporto. Il numero dei sistemi operativi supportati è in continuo aumento e l'interazione con i widget nativi è costantemente aggiornata ed ampliata, ma non si è ancora riusciti e probabilmente non si riuscirà mai a coprirne la totalità.

Generalmente le differenti implementazioni di SWT sono funzionalmente equivalenti, ma potrebbero nascere dei problemi legati ai dettagli di una particolare implementazione. E' necessario, quindi, nel caso in cui si voglia sviluppare un'applicazione multi-piattaforma, testare in profondità ogni aspetto.

Un ultimo svantaggio, legato in modo esplicito al programmatore ed ai pattern da utilizzare, è la modalità di gestione delle risorse. In SWT è necessario rilasciare esplicitamente le risorse tramite il metodo `dispose()` per evitare di occupare

inutilmente la memoria. Questo si rispecchia in un lavoro maggiore per lo sviluppatore che deve fare molta attenzione anche a questo aspetto.

3.4 JFace

JFace è un toolkit per la creazione di interfacce grafiche basato su SWT che integra la gestione di tutti i paradigmi per la programmazione grafica. JFace è stato creato per facilitare il lavoro del programmatore che deve realizzare parti grafiche, nascondendo ad esso le implementazioni di SWT. JFace integra al proprio interno vari componenti quali immagini, finestre di dialogo o di preferenze, wizard, ecc. Le due caratteristiche più interessanti sono la gestione delle action e dei viewer.

Il meccanismo delle action è stato sviluppato in modo da fornire al programmatore un sistema per la gestione dei comandi che fosse indipendente dalla modalità con cui è inserito nell'interfaccia grafica. Un'action rappresenta un generico comando che può essere invocato dall'utente attraverso la pressione di un bottone, la scelta di un menù, ecc. Ogni action ha a disposizione una serie di proprietà grafiche specifiche (etichetta, icona, tool tip, ecc.) che vengono utilizzate per rappresentarla graficamente. Questo sistema permette di richiamare la stessa azione in posizioni e modi differenti dall'interno dell'interfaccia grafica, rendendo, quindi, molto più semplice modificare l'aspetto di una finestra senza doverne modificare le action associate.

I viewer sono degli adapter basati su di un modello che implementano determinati widget di SWT. I viewer gestiscono il comportamento del widget e forniscono un accesso di più alto livello per la modifica e lo sviluppo di tali widget. Un esempio sono le liste, le strutture ad albero o le tabelle che, grazie a JFace, hanno integrato un sistema di sincronizzazione automatico con il dominio di utilizzo dell'utente. Questi viewer seguono esattamente il pattern di sviluppo MVC (Model-View-Controller) e sono, quindi, strettamente legati al modello di dati per il quale sono stati creati.

3.5 GEF

Il Graphical Editing Framework è una libreria che permette di sviluppare in modo semplice e controllato rappresentazioni grafiche per modelli preesistenti [18]. La sua caratteristica principale è quella di fornire uno strumento completo per la creazione di un editor grafico sotto la piattaforma Eclipse. La visualizzazione delle parti grafiche viene realizzata utilizzando il framework draw2D, una libreria basata su SWT per il disegno di figure bidimensionali.

Le funzionalità di editing di GEF permettono di creare editor per qualunque tipo di modello. Con questi editor l'utente può modificare in modo visuale il proprio modello, cambiando le proprietà degli elementi, la loro disposizione ed effettuando operazioni più complesse.

Tutte queste modifiche del modello possono essere effettuate utilizzando le più comuni funzioni già presenti all'interno della piattaforma Eclipse, quali il drag and drop, copia/incolla ed azioni invocate da menù o barre degli strumenti.

3.6 RCP

La struttura della piattaforma Eclipse è molto ricca, estensibile e completamente aperta a nuove implementazioni, ma al tempo stesso è complessa e piuttosto pesante.

Anche se la piattaforma Eclipse è stata progettata per essere utilizzata come una piattaforma aperta per il supporto allo sviluppatore, ha una struttura tale che i suoi componenti possono essere utilizzati autonomamente per costruire quasi qualunque applicazione lato client. L'insieme minimo di plug-in necessario per costruire un'applicazione di tipo "rich client" è stato inserito in un progetto chiamato Rich Client Platform. Utilizzando, quindi, un sottoinsieme della piattaforma Eclipse, è possibile realizzare delle applicazioni differenti da un IDE che siano in grado di funzionare esulando dall'intera istanza della piattaforma, in modalità stand-alone.

Tali applicazioni sono sempre basate su di un sistema dinamico a plug-in e l'interfaccia grafica è sempre realizzata utilizzando gli stessi strumenti ed extension point forniti dalla piattaforma.

La Rich Client Platform è il minimo insieme dei plug-in necessari per costruire un'applicazione stand-alone con un'interfaccia grafica ed utilizza solamente i due plug-in principali della piattaforma eclipse: `org.eclipse.ui` e `org.eclipse.core.runtime`, insieme chiaramente a tutti i loro prerequisiti. In realtà, comunque, le applicazioni basate su RCP sono in grado di utilizzare tutte le API ritenute necessarie per le proprie esigenze e di invocare qualunque plug-in che sia presente nella piattaforma.

4
W-GAIN - Workflow Grid Agent INfrastructure

Questo capitolo è dedicato alla descrizione del prototipo di un sistema per la composizione di servizi basato sullo standard XPDL e sul framework ad agenti JADE. Il suo sviluppo può essere visto come la base per la realizzazione di sistemi ad agenti distribuiti, flessibili ed autonomi.

4.1 Introduzione

Prima di passare alla descrizione tecnica del sistema è necessario fornire una breve descrizione delle architetture e delle specifiche che sono state analizzate ed utilizzate in fase di progettazione dello stesso. In particolare si pone l'attenzione sul problema della rappresentazione e fruizione dei servizi in ambienti distribuiti.

4.1.1 SOA

Dal punto di vista tecnologico, gli ultimi 15 anni sono stati impegnati nella realizzazione di middleware standard e di architetture avanzate, imparando dai successi e dai fallimenti dei sistemi distribuiti e dai middleware basati su Message Oriented Architecture, conosciute anche come Message Oriented Middleware. Secondo questo modo di procedere le applicazioni di business comunicano tra di

loro solamente attraverso lo scambio di messaggi: le informazioni che vogliono essere scambiate vengono impacchettate all'interno di un messaggio ed inviate ad un middleware o un broker che ne gestisce l'instradamento e la distribuzione verso altre applicazioni. Con la diffusione di standard per la definizione di servizi globali e lo sviluppo che hanno avuto il Web e l'uso globale delle reti, si è giunti alla formalizzazione di un nuovo approccio per la realizzazione di sistemi distribuiti. Il servizio diventa il punto centrale attorno al quale vive e si formalizza l'intero processo di business, per formare una Service Oriented Architecture. SOA, quindi, è un particolare tipo di architettura che permette il disaccoppiamento e il binding dinamico tra i servizi.

Il funzionamento è piuttosto semplice ed è basato su tre concetti fondamentali. In primo luogo è necessario fornire una definizione astratta dei servizi, includendo i dettagli che permettono, a chi vuole utilizzare tale servizio, di comporlo in modo appropriato (`bind`). Poichè tale servizio è globalmente accessibile e disponibile, i provider devono pubblicarne i dettagli in modo tale che chiunque ne voglia sfruttare le funzionalità possa capire precisamente cosa fare, ottenendo tutte le informazioni necessarie (`publish`). Infine, tutti gli utilizzatori hanno bisogno di una serie di metodi per determinare la disponibilità dei servizi a loro necessari (`find`). Per far in modo che l'approccio bind/publish/find possa essere applicato correttamente, è necessario definire gli standard per gestire cosa e come pubblicare, come trovare le informazioni ed i dettagli su come comporre i servizi. Il provider dei servizi descrive le sue semantiche, cioè le funzioni che supporta in aggiunta alle sue tipiche operazioni e richiede i dettagli su come accedere al servizio. Le informazioni descrittive sui servizi sono pubblicate in una directory o in un registro globale. Il richiedente utilizza una facility per il discovery associata ai registri per trovare ciò a cui è interessato, basandosi sulle informazioni pubblicate. Il richiedente, dopo aver ricevuto i dati relativi ai servizi disponibili li può comporre in modo autonomo ed indipendente. Le informazioni sulle descrizioni dei servizi sono basate sul linguaggio WSDL; il meccanismo di pubblicazione/identificazione di un servizio, tramite la consultazione di registri dedicati, è ottenuto attraverso UDDI che specifica sia come pubblicare che come ricercare i servizi.

In un tipico sistema SOA, quindi, i vari step che scoprono e compongono i servizi sono i seguenti: il richiedente descrive i servizi necessari attraverso delle funzioni basandosi su questi input, la facility per il discovery produce una lista di servizi candidati che soddisfano tali necessità, il richiedente sceglie un servizio e determina il suo indirizzo ed i dettagli per il binding, come il protocollo da utilizzare e il formato dei messaggi richiesto.

4.1.2 GRID

Con il termine Grid computazionale si intende genericamente un ambiente distribuito geograficamente, basato su di un'infrastruttura fortemente decentrata e di natura eterogenea per la realizzazione di calcolo computazionale complesso. Un Grid può racchiudere al proprio interno un'ampia gamma di risorse fisiche distribuite quali PC, workstation, cluster, sistemi di storage, database, ecc. Tali risorse vengono presentate come un'unica risorsa computazionale integrata. In generale, questo è possibile grazie ad un insieme di servizi globali che ogni Grid è in grado di fornire ed implementare: servizi d'informazione e di monitoraggio, gestione delle risorse, trasferimento di dati, sicurezza, tolleranza agli errori, ecc.

Negli ultimi anni sono stati affrontati numerosi studi e ricerche da parte della comunità legata al Grid per favorire l'integrazione delle tecnologie grid con i Web Services. La OGSI (Open Grid Service Infrastructure [26]) definisce i servizi del Grid come un'estensione dei Web Services. In particolare, definisce i meccanismi per creare, gestire e scambiare informazioni fra questi servizi. L'utilizzo dei Web Services all'interno delle strutture dei Grid permette agli sviluppatori di integrare i loro servizi fra strutture eterogenee e dinamiche utilizzando un unico standard. L'importanza che possono ricoprire i Web Services in questo ambito è stata rilevata anche dalla comunità degli sviluppatori di sistemi ad agenti ed in particolare dalla comunità di JADE che ha provveduto a sviluppare un add-on per la comunicazione tra le due tecnologie [9].

La creazione di un sistema Grid con tutti i suoi servizi è estremamente difficile e complessa ed esistono un numero limitato di strumenti a supporto del programmatore. Per costruire applicazioni Grid riutilizzabili ed affidabili, gli

sviluppatori devono affidarsi ad un insieme di strumenti che nascondono i dettagli della maggior parte dei servizi e permettono di creare modelli consistenti sui quali possono essere integrate unità testate ed affidabili già create in precedenza.

Da alcuni anni all'interno della comunità Grid, si sta facendo strada l'idea di poter utilizzare le tecnologie legate ai workflow ed in particolare ai sistemi di gestione a workflow. Tali sistemi si stanno evolvendo verso una struttura che offre da un lato, un supporto alla definizione di servizi ad alto livello senza doversi occupare dei dettagli dell'implementazione, e dall'altro, una serie di funzionalità per l'esecuzione ed il deploy automatico sulle risorse messe a disposizione dal Grid.

L'aspetto importante della ricerca oggetto del presente lavoro di tesi effettuata in questo ambito, è il tentativo di sfruttare gli agenti come motore per un sistema distribuito, basato sull'utilizzo dei workflow per lo sviluppo e la gestione dei servizi [3]. Il sistema ad agenti, realizzato sulla piattaforma JADE, quindi, sostituisce il comune concetto di Grid e descrive i propri servizi utilizzando workflow e Web Services [2].

4.2 Il Sistema

Con il passaggio da un sistema di gestione del workload ad un sistema per la gestione dei worflow non esiste più un mapping di tipo task/job, ma un mapping di tipo task/servizio.

W-GAIN (Workflow Grid Agent Infrastructure [1]) è un sistema multi-agente che fornisce supporto all'utente sia nella fase di sviluppo che in quella di esecuzione di applicazioni distribuite basate sui workflow.

In particolare il sistema permette di:

- definire workflow componendo i servizi forniti dai diversi nodi del sistema.
- Distribuire in modo trasparente il lavoro ai nodi della rete.
- Eseguire i task.
- Monitorare le risorse.

- Gestire situazioni di risorse non disponibili.

Il sistema è basato su cinque differenti tipologie di agenti: Personal Ggent, Component Agent, Legacy Agent, Workflow Manager, Directory e Task Facilitator. Il Personal Agent è un agente che, associato in modo diretto ad un utente, ne consente l'interazione con il sistema, permettendo di monitorare le risorse ed effettuare tutte le operazioni necessarie alla supervisione dei processi. Esso si presenta con una interfaccia grafica nativa sviluppata sulla piattaforma Eclipse utilizzando SWT/JFace e permette in modo intuitivo di sottoporre i workflow ai Workflow Manager ed eseguire le funzioni principali di gestione del sistema. I Component Agent rappresentano gli intermediari tra il sistema ed i Legacy Agent, ossia gli agenti che forniscono il servizio da utilizzare. Essi traducono l'ontologia di W-GAIN nell'ontologia specifica del Legacy Agent. I Workflow Manager sono gli attori principali del sistema e rappresentano il cervello centrale di W-GAIN. Essi si occupano di:

- interpretare ed eseguire le richieste mandate dagli utenti;
- smistare i task ai Component Agent;
- ricevere i risultati delle elaborazioni e memorizzarli nel contesto di esecuzione;
- informare l'utente dello stato del workflow posto in esecuzione.

I Directory e Task Facilitator sono i componenti preposti ad informare un agente dell'indirizzo degli altri agenti attivi del sistema, fornendo informazioni supplementari quali i servizi offerti ed una loro descrizione semantica: offrono un servizio simile ad un elenco telefonico (pagine gialle).

Un sistema W-GAIN è composto da una federazione di piattaforme JADE. Ciascuna piattaforma presenta il proprio Directory e Task Facilitator ed un Workflow Manager. La Figura 7 mostra una rappresentazione grafica di una piattaforma W-GAIN con in evidenza le interazioni tra le differenti tipologie di agenti. Gli agenti possono essere distribuiti in un qualsiasi nodo che abbia capacità computazionali o fornisca un servizio utilizzabile dal sistema.

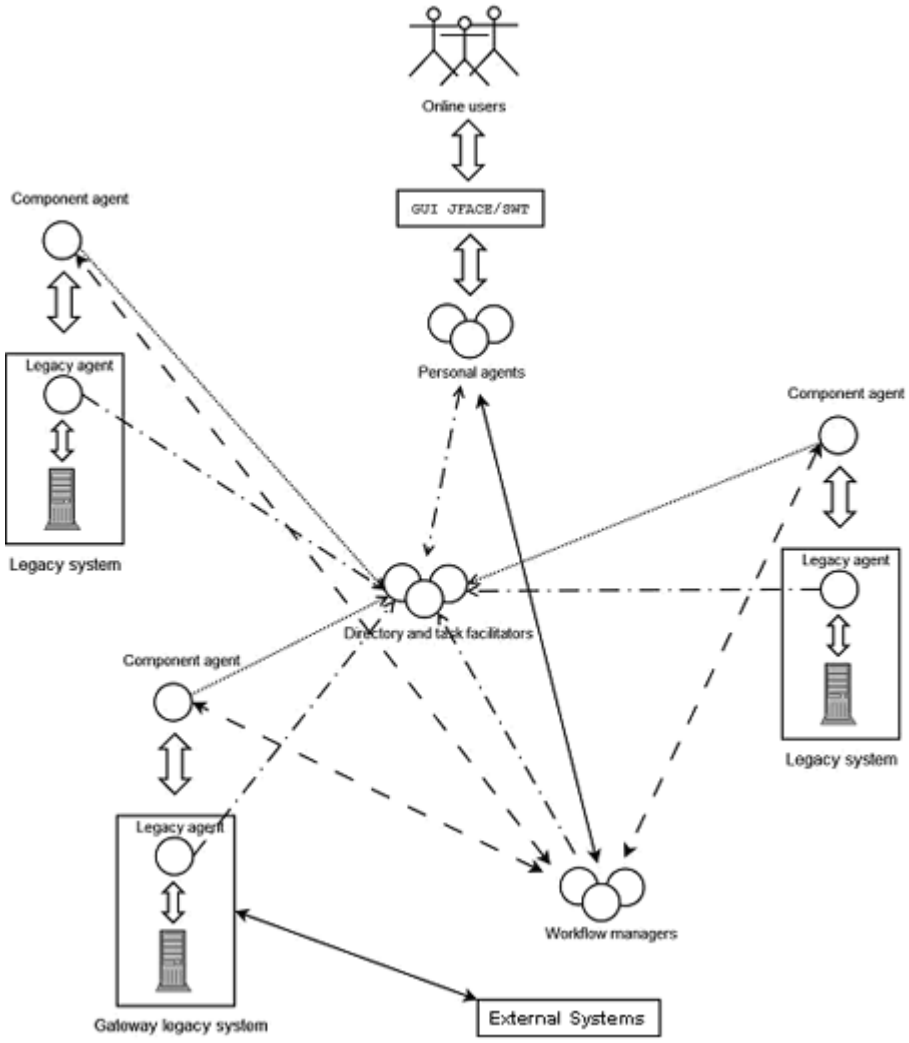

Figura 7 - W-GAIN: architettura del sistema

4.3 Personal Agent

Il Personal Agent è l'agente del sistema che integra l'interfaccia grafica con la quale l'utente interagisce ed accede alle varie funzionalità. Esso rappresenta il

punto di partenza ed arrivo di ogni operazione che si voglia effettuare ed è responsabile della visualizzazione delle informazioni verso l'utente. Il Personal Agent, mediante un'interfaccia grafica sviluppata interamente in SWT/JFace, consente all'utente di entrare nel sistema, valutare le risorse disponibili, sottoporre i task, fornire politiche di scheduling e monitorare l'avanzamento dei lavori da lui sottoposti. Nell'implementazione attuale, il Personal Agent presenta un'interfaccia grafica molto semplice e basilare (Figura 8) e consente un'interazione solamente tramite una serie di pulsanti o menù. In implementazioni future non si esclude la possibilità di poter fornire un supporto a nuove tipologie di interfacce anche non visuali (mail, instant messenger, ecc.).

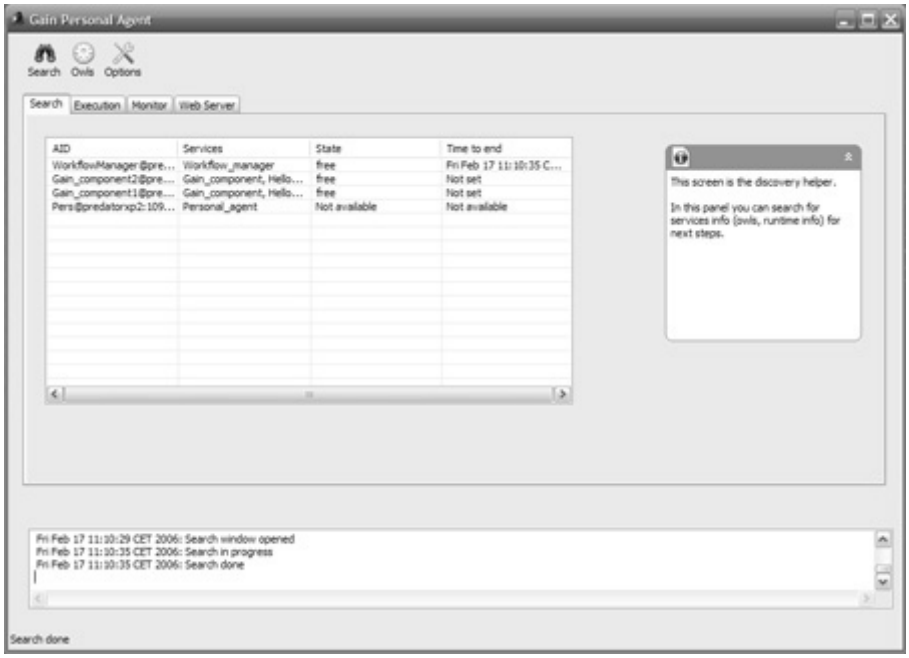

Figura 8 – W-GAIN: interfaccia utente del Personal Agent

Il Personal Agent, come del resto l'intera applicazione, è stato progettato seguendo un pattern di programmazione di tipo MVC, classico per l'implementazione di tool all'interno della piattaforma Eclipse, separando la parte relativa alla visualizzazione rispetto al controllo/modello.

La finestra principale, che compare alla richiesta da parte dell'utente di entrare nel sistema, è composta da quattro pannelli divisi in base al loro compito specifico. Il primo pannello fondamentale è quello `Execution` che permette di caricare un file XPDL nel sistema e, di conseguenza, pubblicare i task ai Workflow Manager presenti. Per la composizione grafica dei file XPDL si utilizza un editor per workflow sviluppato appositamente durante il lavoro di tesi che verrà descritto nel prossimo capitolo. Allo stato attuale tale editor non è integrato all'interno del sistema ad agenti, ma si crede che tale operazione possa essere effettuata senza particolari problemi.

È possibile lavorare in due modalità distinte:

- MANUALE: viene sottoposto un file XPDL al Workflow Manager che ne prende in carico l'esecuzione in modo distribuito.
- AUTOMATICA: viene sottoposto un file XPDL contenente soltanto alcune informazioni parziali, quali il goal che si desidera ottenere e lo stato iniziale (opzionale), necessarie affinché possa essere eseguito un compositore automatico. Esso cercherà di comporre un workflow in modo automatico per soddisfare il goal indicato: una volta che il workflow è stato completato, verrà posto in esecuzione come per il caso precedente.

A queste modalità di esecuzione si aggiunge la possibilità di selezionare l'opzione RELAXED. Tale opzione è piuttosto importante in quanto permette al Personal Agent di proseguire nell'esecuzione del workflow, anche nel caso in cui alcuni task non possano essere eseguiti per mancanza di risorse. E' piuttosto comune, infatti, che in un sistema distribuito, basato su servizi remoti, non vi sia il ritorno di un task posto in esecuzione entro i tempi previsti. Tramite l'opzione RELAXED, quindi, l'utente lascia la facoltà al Workflow Manager di bloccare l'esecuzione di un task o spostarla su di un altro nodo in modo che l'intero processo possa essere portato a termine almeno in parte.

Per permettere lo scambio di risorse quali file tra i vari agenti all'interno del sistema è stato integrato un web server, disponibile attraverso l'interfaccia del Personal Agent ed a lui direttamente associato.

4.3.1 PA - Funzione di Ricerca

Una finestra fondamentale dell'interfaccia messa a disposizione dal Personal Agent è quella di ricerca, attraverso la quale è possibile effettuare ricerche per ottenere lo stato degli agenti presenti sul sistema e la lista dei servizi a disposizione sui vari nodi. Numerose informazioni a runtime caratterizzano lo stato di esecuzione dell'agente, quali il consumo di cpu, la data in cui si prevede di terminare un task bloccante, lo stato attuale, la memoria libera o la memoria di sistema. Tali parametri sono fondamentali per avere sempre sotto controllo lo stato globale del sistema e delle risorse in uso. Nella Figura 9 viene mostrata la sequenza dei messaggi che caratterizza la ricerca.

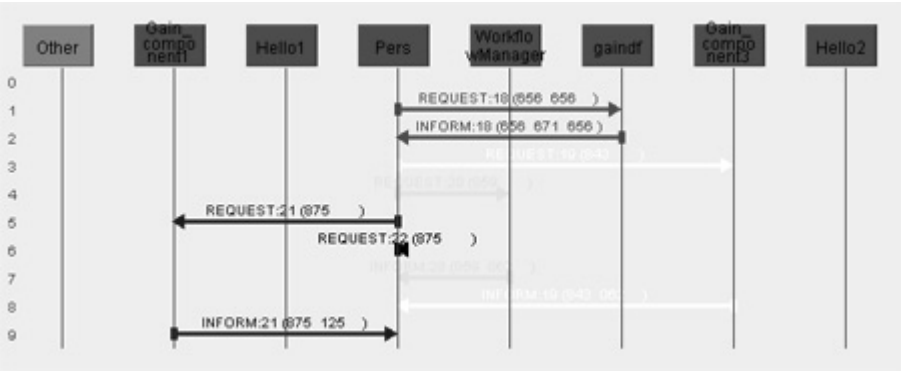

Figura 9 - W-GAIN: sequenza dei messaggi per la ricerca dei servizi

Per effettuare ricerche con informazioni runtime, il Personal Agent esegue le seguenti operazioni:

- richiede al Directory e Task Facilitator l'indirizzo di tutti gli agenti che soddisfano il criterio di ricerca, ovvero l'appartenenza al sistema W-GAIN. Un vantaggio importante dell'uso di un sistema ad agenti, in questo caso, è dovuto al fatto che sulla stessa piattaforma possono coesistere più agenti con funzionalità diverse: sulla stessa istanza JADE, infatti, possono coesistere anche agenti che non fanno parte del

sistema W-GAIN. Per questo motivo è importante capire se un agente fa parte o meno del sistema W-GAIN.

- Ricevute le informazioni sull'indirizzo, contatta ogni Component Agent inviando una RequestInfo.
- Riceve un messaggio di tipo RuntimeInfo contenente le informazioni runtime.
- Presenta le informazioni ricevute all'utente.

L'interfaccia fornisce anche un comodo pannello di log che tiene traccia di tutte le informazioni relative allo scambio di informazioni ed eventi della GUI.

4.3.2 Generazione di un Workflow

Nella fase di esecuzione, l'utente definisce il workflow componendo i task che possono essere eseguiti dai differenti nodi del sistema. In questa fase, un aiuto concreto viene dato dal pannello Search precedentemente trattato, mediante il quale è possibile ottenere informazioni sui servizi disponibili. Il formato scelto per descrivere il workflow è XPDL, grazie alla sua adattabilità alle varie situazioni e possibilità di espansione. La potenza del linguaggio permette la rappresentazione di ogni struttura necessaria anche di notevole complessità. W-GAIN, al momento, implementa buona parte delle strutture definite dallo standard XPDL v1.0.

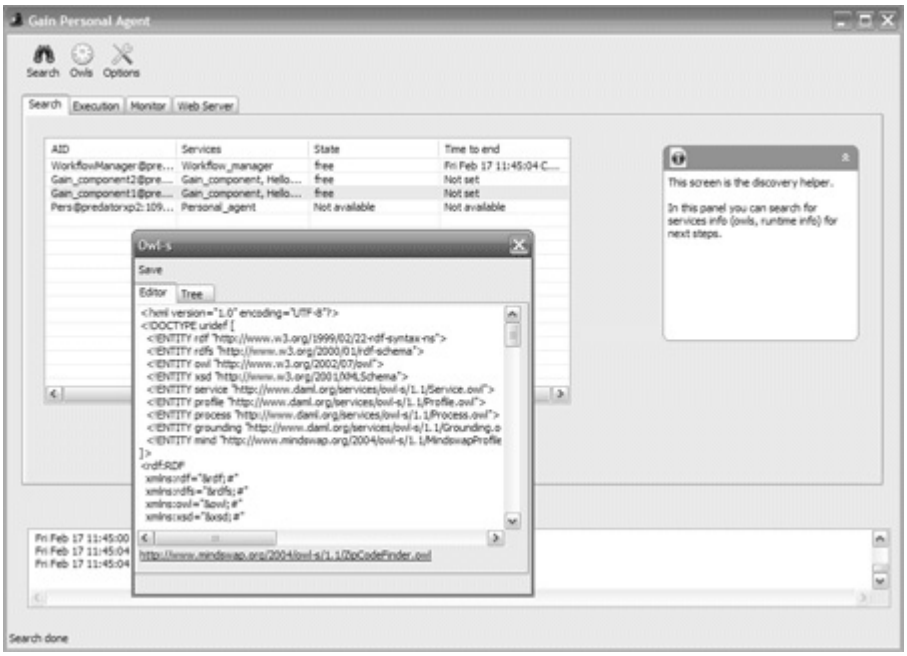

Figura 10 – Visualizzazione file di descrizione di un servizio

Per la gestione e l'interpretazione delle espressioni interne alle strutture XPDL, è stato utilizzato il linguaggio di scripting TCL [23] mediante l'implementazione JACL. JACL (Java Command Language [20]) è una versione del linguaggio TCL sviluppata per ambienti Java. Esso è stato progettato per essere un linguaggio universale di scriptin in Java: l'interprete JACL è scritto completamente in Java e può essere eseguito su qualunque Java Virtual Machine. Esso è stato scelto in quanto presenta un potente sistema per la gestione delle espressioni regolari e si integra perfettamente all'interno di un software Java come W-GAIN. Le variabili definite nel file XPDL possono essere utilizzate direttamente all'interno delle espressioni. Il Workflow Manager gestisce in modo trasparente il flusso delle informazioni durante l'esecuzione dei vari task. Ad ogni passo, le variabili vengono aggiornate con i nuovi valori: in questo modo è possibile modellare qualsiasi casistica. Anche per quanto concerne il linguaggio di scripting, è stato definito un extension point mediante la classe LanguageInterpreter. Linguaggi quali Beanshell

[5], Rhino [27] o Ruby [28] possono essere pertanto agevolmente integrati incrementando notevolmente la versatilità di W-GAIN.

4.3.3 Strutture XPDL e Convenzioni

Nella composizione di file XPDL compatibili con W-GAIN è doveroso tenere in considerazione alcune convenzioni utilizzate per permettere all'utente di definire le strutture più complesse. Per ridurre, infatti, la complessità nella definizione delle strutture, si è preferito adottare un approccio "convention based" preferendolo alla verbosità imposta da file di configurazione xml. Come accennato in precedenza, tutte le espressioni del linguaggio XPDL sono state implementate. Tra le strutture supportate notevole importanza assumono la gestione di:

- `Split` (sia di tipo `XOR` che `AND`): sono operatori di flusso che permettono di scegliere la successiva attività da eseguire. Se di tipo `XOR`, presentano delle condizioni espresse in linguaggio TCL che permettono di selezionare una sola attività da eseguire. Se di tipo `AND` provocano l'esecuzione di task in parallelo che termineranno in relazione alla condizione di `Join` stabilita.
- `Join` (sia di tipo `XOR` che `AND`): sono operatori aggregatori di flusso. Stabiliscono in che modo unire le elaborazioni provenienti da attività eseguite parallelamente. Nel caso di `Join XOR` l'attività corrispondente viene eseguita se almeno un input in ingresso ha terminato la sua esecuzione. Nel caso di `Join AND`, invece, l'attività viene eseguita solo se tutti gli input hanno terminato la loro esecuzione.

4.3.4 Monitoraggio dei Workflow

Tramite la scheda rappresentata in Figura 11, viene installato nel Personal Agent un behaviour che ciclicamente interroga il Workflow Manager ottenendo informazioni runtime. Nella tabella appaiono tutti i workflow in esecuzione nei vari Workflow Manager.

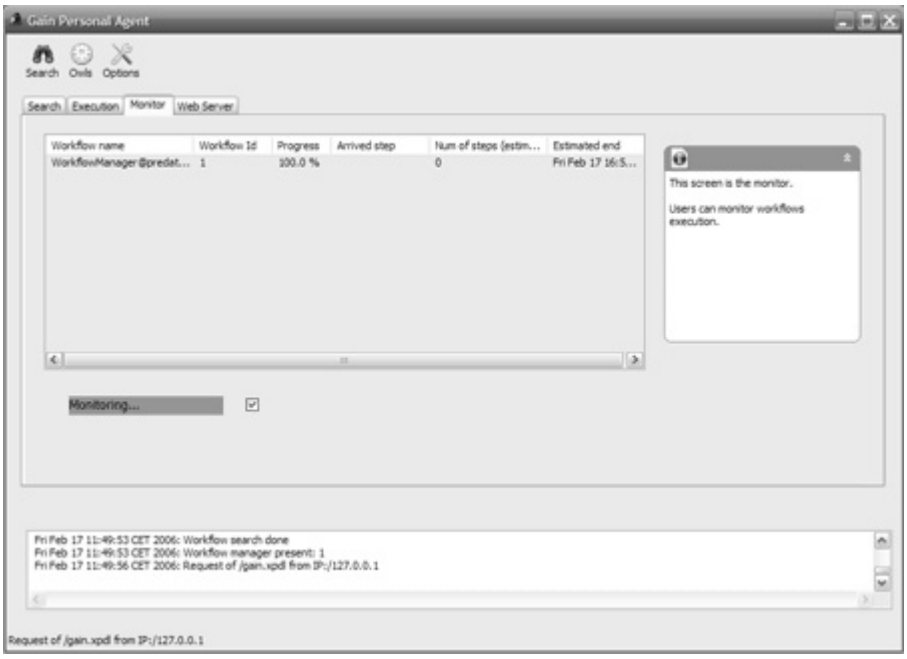

Figura 11 - Monitoring dell'esecuzione

Dal punto di vista dei protocolli, il Personal Agent invia delle Request del tipo RequestWorkflowStatus ricevendo come risposta dei WorkflowStatus (Figura 12 - Schema Workflow Manager).

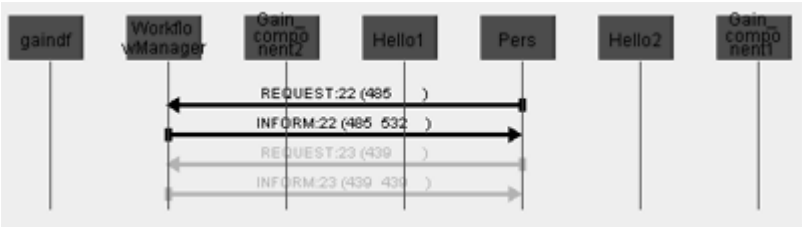

Figura 12 - Schema Workflow Manager

4.3.5 Web Server Integrato

W-GAIN integra nel Personal Agent e nel Component Agent un Web server rispettivamente sulla porta 80 e 81 per consentire un agile trasferimento dei file. Quando viene pubblicato un file XPDL, tutti i file ed i package esterni vengono automaticamente messi in condivisione e nell'ontologia viene passato il link alla risorsa. E' anche stata implementata la possibilità di impostare l'indirizzo pubblico della rete per consentire di operare dietro dispositivi NAT. Tramite un menù contestuale è infine possibile aggiungere e rimuovere manualmente le risorse ed i file.

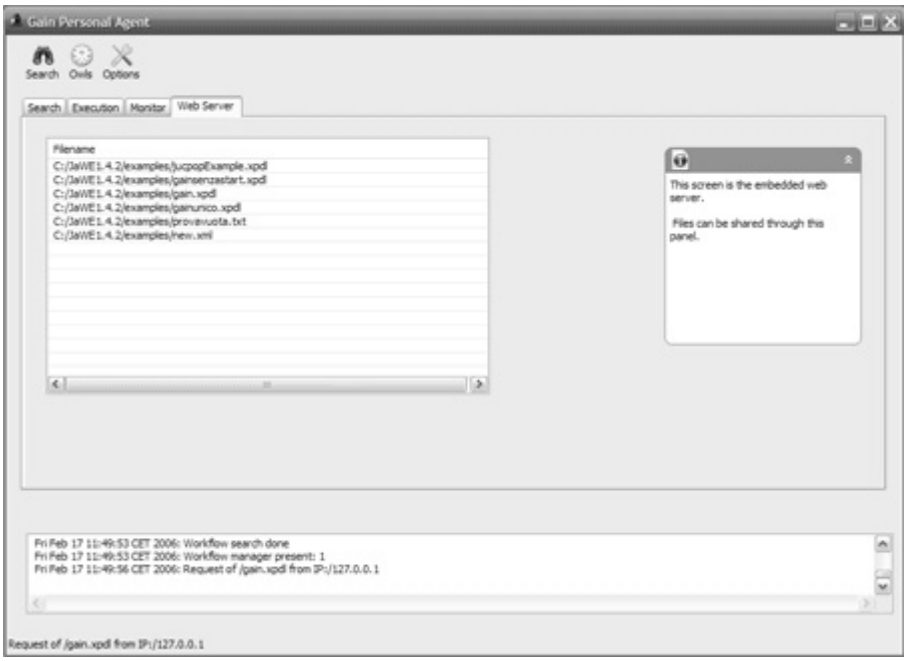

Figura 13 - Web Server integrato

4.4 Legacy Agent

Un Legacy Agent è un normale agente JADE con determinate caratteristiche che lo rendono adatto all'inserimento in una rete W-GAIN. Ogni agente JADE, infatti, presenta una propria ontologia che utilizza per comunicare con altri agenti. L'idea di base nella progettazione del sistema W-GAIN è stata quella di realizzare un'architettura aperta, che permettesse di sfruttare tutto quanto è già stato realizzato finora in ambito di programmazione ad agenti. Per questo motivo sono stati progettati e realizzati i Component Agent, ovvero agenti che svolgono l'essenziale ruolo di intermediari tra il sistema ed i Legacy Agent. Con il termine Legacy Agent, quindi, si intendono tutti quegli agenti preesistenti in una piattaforma JADE, ai quali è stato aggiunto semplicemente un richiamo alla compatibilità con W-GAIN:

```
super.accessToGain(this, dfd, owlsURI);
```

Per entrare a far parte del sistema W-GAIN, un qualunque sviluppatore di agenti JADE deve inserire questa chiamata all'interno del metodo `setup()` dell'agente stesso ed avere a disposizione un riferimento al proprio Directory Facilitator.

Un Legacy Agent può mandare informazioni sul suo stato, utili al Workflow Manager per realizzare politiche di scheduling e poter scegliere le risorse più efficienti per i compiti richiesti. Il sistema W-GAIN fornisce anche un ulteriore metodo statico:

```
super.accessToGain(this, dfd, owlsURI, String[] filesToShare);
```

che accetta un array di percorsi a file da mettere in condivisione sul file server integrato.

4.4.1 Web Service Integration Gateway

Il Legacy Agent ha la necessità di interfacciarsi con i Web Services che rappresentano i servizi da rendere disponibili nel sistema. Per fare questo, è stato utilizzato un add-on sviluppato per il framework JADE da Whitestein Technologies AG, chiamato Web Service Integration Gateway [9]. Questo add-on permette ad un sistema ad agenti JADE di invocare un Web Service ed al tempo stesso di esporre le funzionalità degli agenti come Web Services. WSIG supporta tutti gli standard che definiscono i Web Services, ovvero WSDL per la descrizione, SOAP per il trasporto dei messaggi e UDDI come registro per la pubblicazione delle descrizioni. In Figura 14 è rappresentata l'architettura dell'intero add-on WSIG il quale si presenta come una Web application composta da due elementi principali:

- Servlet WSIG
- Agente WSIG

La servlet è l'interfaccia con la rete Internet ed è responsabile di:

- servire le richieste HTTP/SOAP entranti.
- Estarre il messaggio SOAP.
- Preparare la corrispondente azione e passarla all'agente WSIG.

Una volta che l'azione è stata servita:

- convertire i risultati dell'azione in messaggi SOAP.
- Preparare la risposta HTTP/SOAP affinchè possa essere spedita al richiedente.

L'agente WSIG funge da gateway tra il Web ed il mondo degli agenti ed è responsabile di:

- trasmettere le azioni ricevute dalla servlet agli agenti presenti nel sistema che sono in grado di eseguirle.
- Ricevere le azioni eseguite dagli agenti del sistema.
- Registrarsi presso il DF per ricevere le notifiche sullo stato del sistema, in particolare per monitorare le registrazioni/cancellazioni.
- Creare la descrizione WSDL corrispondente ad ogni agente registrato nel DF e pubblicare il servizio all'interno di un registro UDDI.

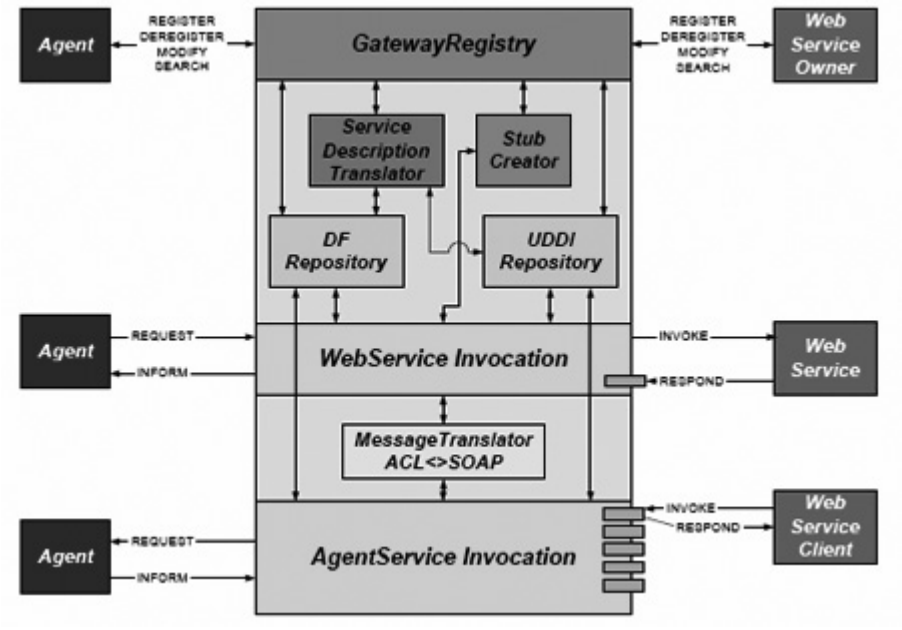

Figura 14 - WSIG – Architettura

Due processi base sono sempre attivi all'interno di WSIG:

- il processo responsabile di intercettare le registrazioni/cancellazioni all'interno del DF e di convertirle in descrizioni WSDL (processo completamente gestito dall'Agente WSIG).
- Il processo responsabile di intercettare le richieste entranti da Web Services esterni e trasformarle in azioni per gli agenti. Questo processo è gestito in modo congiunto dalla Servlet, che traduce le richieste entranti, e dall'Agente che le invia al sistema.

Grazie a queste funzionalità il Legacy Agent è in grado di interfacciarsi in maniera completa con i Web Services e di far interagire gli altri agenti del sistema W-GAIN tramite i servizi messi a disposizione sul Web.

4.5 Component Agent

Il Component Agent svolge la funzione di intermediario tra il sistema ed il Legacy Agent, realizzando numerose operazioni di gestione e consentendo agli utenti di sistemi JADE preesistenti di inserire agevolmente il proprio agente in un contesto W-GAIN. Il Component Agent si trova sulla stessa piattaforma del Legacy Agent e può accedere alle classi dell'ontologia specifica dell'agente di cui svolge il ruolo di intermediario. Esso conosce, inoltre, l'ontologia generica del sistema W-GAIN (un'ontologia orientata ai servizi) che utilizza per comunicare con il sistema. Per semplificare la comprensione dell'utilità del Component Agent, se ne riassumono le funzionalità:

- gestione del Legacy Agent associato, tramite l'interrogazione sullo stato e la comunicazione di tali informazioni ai Workflow Manager.
- Traduzione delle invocazioni dei servizi espresse nell'ontologia generica del sistema in invocazioni specifiche nell'ontologia del Legacy Agent.
- Traduzione delle risposte specifiche espresse nell'ontologia del Legacy Agent in invocazioni nell'ontologia generica del sistema.

Queste importanti funzioni permettono di raggiungere un obiettivo ritenuto fondamentale, ovvero rendere il sistema agnostico per quanto concerne le ontologie dei sistemi collegati. Tale particolare architettura consente inoltre di poter utilizzare non solo agenti, ma qualsiasi entità con un protocollo sottostante compatibile.

4.6 Workflow Manager

Il Workflow Manager rappresenta il componente principale e centrale di W-GAIN. Esso si occupa principalmente di ricevere le richieste dai Personal Agent e di eseguirle svolgendo una complessa serie di operazioni, nello specifico:

- effettuare il parsing del file XPDL che viene sottoposto al sistema in input.
- Nel caso di composizione automatica dei servizi, ricercare nella rete W-GAIN tutti gli agenti che soddisfano particolari requisiti, prelevare i file di descrizione associati e comporre automaticamente, grazie al pianificatore non lineare integrato, un file XPDL fittizio utilizzato nelle interazioni successive.
- Ricercare nella rete W-GAIN i servizi che possono eseguire il task richiesto, scegliendo il migliore di essi in funzione delle informazioni runtime fornite dal Component Agent.
- Eseguire tale task e immagazzinarne il risultato all'interno di un contesto di workflow.
- Gestire situazioni di parallelismo e split `XOR` invocando, quando necessario, il motore di scripting TCL integrato.
- Fornire informazioni sullo stato di esecuzione del workflow.
- Gestire situazioni di risorse impegnate, tramite la riallocazione delle stesse e la migrazione dei task su di un nodo libero o più scarico.

L'architettura progettata presenta numerosi “extension point” che permettono l'introduzione di nuovi linguaggi di scripting, linguaggi per esprimere le precondizioni e postcondizioni, nuove politiche di scheduling o di selezione delle risorse migliori.

4.6.1 Protocolli

Il sistema W-GAIN utilizza un’ontologia specifica per realizzare le operazioni. Protocolli specifici di JADE, inoltre, forniscono il supporto ad essa per ottenere i risultati voluti. Nell'ambito dell'invio dei workflow, è stato definito un protocollo che permette di ottimizzare le caratteristiche del Workflow Manager prima di inviare ad esso i dati del workflow. In Figura 15 è rappresentato lo scambio di messaggi che avviene durante la richiesta di invio ed esecuzione di un workflow ad un Workflow Manager. Tale procedura prevede le seguenti operazioni:

- invio di una CFP (Call For Proposal) a tutti i Workflow Manager del sistema dopo aver consultato il DF. La proposta contiene il link al file XPDL con la descrizione del workflow, le caratteristiche richieste (se automatizzato o rilassato) e la data entro cui è necessario che l'esecuzione del workflow sia completata.
- Risposta da parte dei Workflow Manager al Personal Agent con un messaggio contenente informazioni sul loro stato attuale come, per esempio, il carico di lavoro o l'occupazione delle risorse. È possibile agire sui criteri di selezione del Workflow Manager per individuare, per esempio, l'agente che, anche se occupato, ha una coda di lavori minore oppure la cui previsione di tempo necessario a concludere i piani in coda sia inferiore rispetto agli altri.
- Selezione, da parte del Personal Agent, del Workflow Manager ritenuto più idoneo allo svolgimento del task tramite l'invio di un messaggio di "`Accept Proposal`" al Workflow Manager selezionato e di "`Reject Proposal`" a quelli scartati.
- Caricamento del file XPDL dal server Web integrato nel Personal Agent.
- Parsing del file XPDL con particolare attenzione alla gestione delle situazioni di `Split` e `Join`.
- Interrogazione del Directory and Task Facilitator per la ricerca di un Component Agent in grado di soddisfare il singolo task specifico e, una volta ottenuto l'indirizzo, richiesta al Component Agent dell'istanza del Legacy Agent che fornisce realmente tale servizio. La richiesta viene effettuata con una Action di tipo "`Invoke`", che fa parte dell'ontologia generica W-GAIN orientata ai servizi.
- Traduzione da parte del Component Agent dell'ontologia generica W-GAIN nell'ontologia specifica del Legacy Agent. Il Component Agent agisce sempre come intermediario tra il sistema ed il Legacy Agent e, poiché appartiene al suo stesso container, può accedere alla sua libreria

ontologica di classi Java. Vengono così tradotte le strutture dati XPDL in strutture JADE e trasformate le invocazioni in azioni e predicati.

- Ricezione da parte del Component Agent della risposta dal Legacy Agent sull'esecuzione del task e traduzione della stessa nell'ontologia generica W-GAIN.
- Inoltro della risposta al Workflow Manager per le elaborazioni successive.

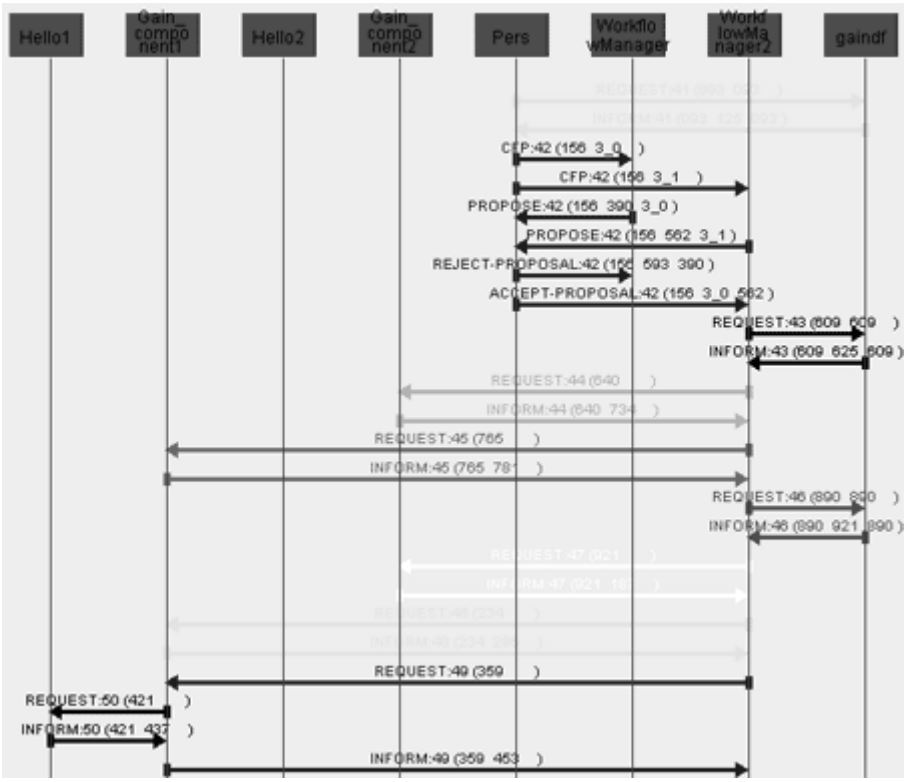

Figura 15 - Protocolli di invio ed esecuzione di un workflow

Ad ogni Workflow Manager è associato un Responder grazie al quale è in grado, in ogni momento dell'esecuzione, di inviare agli altri agenti del sistema il proprio stato interno. In questo modo, attraverso la finestra Monitor dell'interfaccia

del Personal Agent (rappresentata in Figura 11), è possibile per l'utente mantenere il controllo sull'esecuzione. In particolare è possibile ottenere informazioni sullo stato dei workflow posti in esecuzione e visualizzarne le risposte. Tale funzione è gestita attraverso un behaviour ciclico che invia costantemente delle richieste al Workflow Manager.

5
Wolf - Workflow Editor

In questo capitolo si presentano la struttura ed il funzionamento di un editor grafico per workflow descritti tramite il linguaggio XPDL. Tale editor viene utilizzato per la creazione delle rappresentazioni dei processi gestiti dal sistema W-GAIN descritto nel capitolo precedente.

5.1 Descrizione del Sistema

Nello studio del sistema presentato nel capitolo precedente, è sorta la necessità di avere a disposizione una serie di applicazioni e tool che permettessero all'utente di interagire in modo assistito e più completo con lo strato di agenti sottostante. Tale necessità ha portato, per prima cosa, allo studio ed alla realizzazione di un editor di workflow che racchiudesse al proprio interno funzionalità di creazione e visualizzazione di workflow. Lo standard sul quale ci si è concentrati è stato XPDL in quanto si è rivelato quello di più basso livello e, di conseguenza, maggiormente adattabile alle differenti necessità del sistema.

La realizzazione di tale editor è stata supportata da un progetto di ricerca sviluppato in collaborazione con Telecom Italia. Tale progetto è interno all'azienda stessa ed ancora in fase di sviluppo: questo non permette la diffusione delle informazioni legate alla sua struttura ed implementazione. L'idea di base comune è stata quella di utilizzare un sistema ad agenti basato sulla piattaforma JADE per gestire apparati di rete. Per facilitare la configurazione e gestione degli agenti si è

realizzato un editor che, grazie all'uso dei workflow, permette all'utente di generare il codice Java degli agenti in modo visuale, senza ricorrere ad un'elevata esperienza nella programmazione e, soprattutto, senza avere alcuna conoscenza di sviluppo sulla piattaforma JADE.

5.1.1 Obiettivo

L'obiettivo del progetto "Wolf – Workflow Editor" è stato quello di realizzare editor visuale per la generazione e gestione di workflow. E' stato scelto lo standard XPDL promosso da WfMC. Tramite questo editor si vuole avere a disposizione uno strumento che permetta di gestire l'intero ciclo di vita della generazione di un workflow descritto con tale standard. Le fasi fondamentali che si vogliono gestire sono:

- creazione ex-novo di un file "`.xpdl`".
- Apertura di un file precedentemente creato.
- Modifica e salvataggio del file.
- Verifica della corrispondenza del file allo standard XPDL v1.0.
- Visualizzazione testuale con possibilità di modifica del file.

Sul mercato esistevano già alcuni editor per workflow che potevano incontrare le esigenze sopra elencate, ma nessuno si adattava in modo soddisfacente alle esigenze del sistema W-GAIN e del progetto di Telecom Italia. In particolare, è stato analizzato in modo approfondito un editor open source sviluppato da Together Teamlösungen, chiamato JaWE [11]. Tale editor è interamente scritto in Java e supporta in modo completo lo standard XPDL v1.0 seguendo le specifiche imposte da WfMC [36]. Nonostante tale editor fosse funzionante e piuttosto completo, è stato scartato in quanto non rispondeva alle esigenze di estensibilità ed eterogeneità richieste. Esso, infatti, si presenta come un programma stand alone, difficilmente modificabile e poco adattabile alle mutevoli esigenze di un utente che opera in un ambiente distribuito. Infine, da una serie di prove pratiche e dirette effettuate si sono notati grossi problemi in termini di prestazioni ed occupazione delle risorse.

Per questo motivo principale si è scelto di sviluppare ex novo un editor utilizzando la piattaforma Eclipse. Quest'ultima scelta si è rivelata fondamentale per il corretto funzionamento dell'editor e dell'intero sistema: la piattaforma Eclipse, grazie alla sua struttura aperta a plug-in, infatti, ha permesso la creazione non solo di un semplice editor di file XPDL, ma di un sistema integrato per la generazione di codice per agenti JADE. Essa, inoltre, mette a disposizione dello sviluppatore una serie di tool e librerie che hanno permesso di velocizzare e rendere più coerente lo sviluppo dell'editor. Infine, il prodotto finale ha superato una serie di test prestazionali dimostrando un notevole miglioramento rispetto a JaWE.

5.1.2 Specifiche

Vista la scelta di realizzare ex novo un editor per la gestione grafica di workflow, per prima cosa è stato necessario definire in modo preciso le specifiche da implementare. Tali specifiche sono state dettate dall'esigenza di potere non solo mantenere in modo visuale un workflow, ma di avere uno strumento adattabile e facilmente estensibile per generare flussi di azioni che potessero essere realizzate attraverso un sistema ad agenti distribuito.

In particolare il sistema deve:

- editare workflow scritti in XPDL v1.0: in particolare deve permettere di aprire, modificare e salvare in XPDL.
- Visualizzare tutte le proprietà interessanti relative ad un elemento in focus.
- Visualizzare l'albero delle dipendenze di un workflow: permettere all'utente di avere una visione globale dell'intero processo mostrando tutte le informazioni contenute in file esterni e workflow importati.
- Visualizzare graficamente i tag principali definiti dallo standard: WorkflowProcess, Activity, Transition.
- Editare l'aspetto grafico di ogni singolo tag, modificandone la posizione, il colore e le dimensioni.

- Validare il file XPDL, secondo lo schema "`.xsd`" [34] fornito da WfMC, all'apertura, al salvataggio del file o su richiesta esplicita dell'utente.
- Segnalare gli errori nei costrutti con collegamento rapido all'oggetto con l'errore.
- Visualizzazione di tutte le Application implicate nel processo descritto dal workflow.
- Generazione automatica degli Id dei nuovi oggetti.
- Creazione di Application tramite invocazioni di codice Java direttamente all'interno dell'editor.

5.1.3 Caratteristiche Tecniche

L'editor di workflow Wolf è stato sviluppato interamente all'interno della piattaforma Eclipse, seguendo tutti gli standard ed i paradigmi imposti dalla piattaforma stessa. Esso si presenta come un insieme di plug-in che devono semplicemente essere inseriti all'interno della directory "`plugins`" dell'installazione di Eclipse affinché possa funzionare. La parte grafica relativa alla visualizzazione dei componenti del workflow ed alla loro modifica visuale è stata realizzata utilizzando GEF, un progetto ufficiale della piattaforma Eclipse, creato appositamente per generare editor grafici basandosi su di un modello dati preesistente (vd. Paragrafo 3.5). Tale libreria basa interamente il proprio funzionamento su di un paradigma di tipo MVC, nel quale:

- il Model è un qualunque sistema di classi Java che rappresentano i tag del linguaggio XPDL;
- la View è rappresentata tramite la libreria `draw2d` (inclusa all'interno di GEF) per il disegno di oggetti bidimensionali;
- il Controller è rappresentato dalle sole parti che hanno una rappresentazione grafica, ovvero i Process, le Activity e le Transition.

5.2 Il Modello Dati

Il modello dati è la parte fondamentale sulla quale è basato l'intero funzionamento dell'editor. Per questo motivo è stata posta estrema cura ed attenzione nello sviluppo di un modello che fosse il più coerente possibile con lo standard XPDL: per fare ciò sono stati mappati all'interno di una specifica classe, tramite una corrispondenza uno a uno, tutti i tag definiti dallo standard e sono stati astratti i costrutti principali e raggruppati in quattro classi fondamentali:

- `XPDLSimpleTag`: è l'elemento atomico del modello dal quale derivano tutti gli altri elementi come sua estensione. In questa classe sono state racchiuse tutte le caratteristiche che accomunano i tag XPDL, quali un `Id` univoco (ogni tag deve avere un proprio nome identificativo e tale nome deve essere unico all'interno del medesimo file XPDL) ed un valore associato a tale `Id` (tale valore è una stringa testuale che descrive l'azione od un attributo associato al tag). La classe `XPDLSimpleTag` è anche responsabile dell'interazione con il Controller e, quindi, con tutte le funzionalità dell'editor.
- `XPDLComplexTag`: estende la classe `XPDLSimpletag` e rappresenta un elemento XPDL più complesso. Se il `XPDLSimpleTag` è composto solamente da un `Id` ed un valore, `XPDLComplexTag` è un insieme di tag atomici e la sua complessità può aumentare a piacere. Esso rappresenta la maggior parte dei tag XPDL con i loro attributi e sottoelementi.
- `XPDLCollectionTag`: rappresenta il ramo padre di una struttura ad albero o meglio l'elemento base di una `Collection`. In XPDL è possibile, infatti, definire una serie di oggetti dalle caratteristiche comuni come appartenenti alla stessa collezione di elementi. Per esempio, all'interno della `Collection` di `WorkflowProcesses` sono definiti tutti gli oggetti di tipo `WorkflowProcess`: il numero di tali oggetti è compreso fra 1 ed infinito (l'unico limite è dato dalla

scalabilità del sistema e dalle prestazioni dei singoli nodi sui quali tali processi verranno posti in esecuzione). Nella classe `XPDLCollectionTag` sono implementati tutti i metodi per la gestione della `Collection`, quali la creazione di un nuovo figlio, la sua eliminazione, ecc.

- `XPDLCollectionElementTag`: rappresenta il singolo elemento appartenente ad una `Collection` e, quindi, figlio di `XDPLCollectionTag`. Nella pratica è semplicemente un `XPDLComplexTag` con l'aggiunta del riferimento al proprio oggetto padre.
- `XPDLAttribute`: rappresenta un attributo che può essere associato ad un tag XPDL ed è composto da un Id ed un valore. Il valore può essere scelto da una lista di valori predefiniti: in questo caso si parla di `XPDLChoice`.

Questi sono solamente gli oggetti principali del modello, il quale ha la funzione di trasformare una struttura annidata di tag XPDL in un albero di dipendenze espresso in linguaggio Java. Come sottolineato in precedenza, è stato realizzato un mapping uno a uno fra i tag XPDL e le classi Java, mapping che per brevità si evita di riportare in modo completo.

5.3 Struttura delle Classi

In questo paragrafo si fornisce una breve descrizione della struttura delle classi dell'intero editor di workflow Wolf e si dettagliano le funzionalità delle classi principali. In Figura 16 è rappresentata la struttura dei package che compongono il sistema.

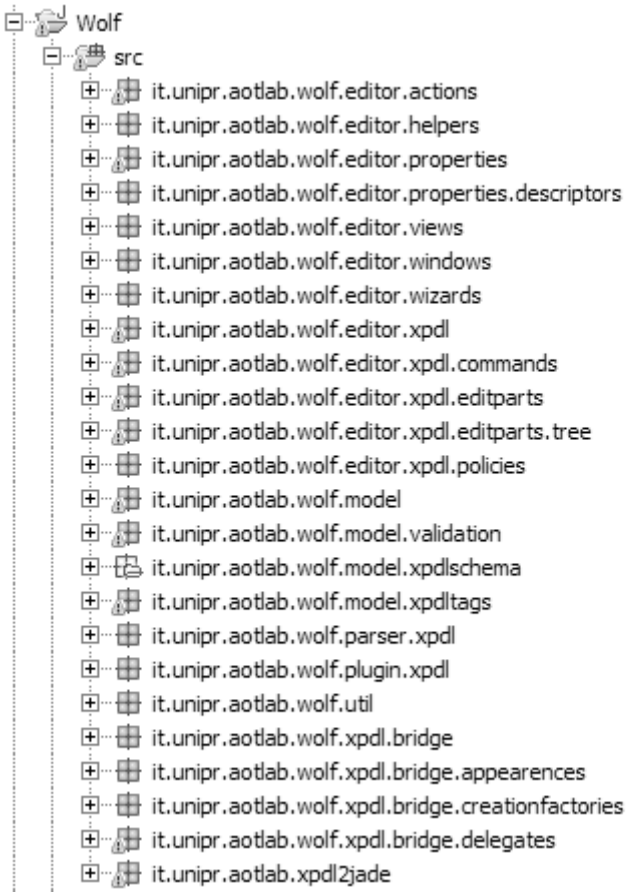

Figura 16 - Struttura delle classi di Wolf.

In particolare si distinguono due gruppi fondamentali di classi: **`it.unipr.aotlab.wolf.editor`** contenente tutto il codice relativo allo sviluppo dell'editor visuale all'interno di Eclipse utilizzando GEF e **`it.unipr.aotlab.wolf.model`** contenente il codice relativo al modello che rappresenta il linguaggio XPDL.

it.unipr.aotlab.editor.actions

In questo package sono implementate le Action utilizzate all'interno dell'editor. Le Action possono essere specifiche della piattaforma o di GEF (`org.eclipse.ui.actions` o `org.eclipse.gef.ui.actions`).

it.unipr.aotlab.wolf.editor.helpers

In questo package sono presenti le classi che implementano `org.eclipse.jface.viewers.ICellEditorValidator` e che sono utilizzate all'interno della `Properties View` per validare i valori inseriti dall'utente.

it.unipr.aotlab.wolf.editor.properties

In questo package sono presenti le classi che implementano `org.eclipse.ui.views.properties.IPropertySource`. Ad ogni classe corrisponde un elemento XPDL al quale si vuole fornire una relativa finestra di proprietà. Nella versione attuale di Wolf sono implementate le properties di tutti i tag XPDL v1.0.

it.unipr.aotlab.wolf.editor.properties.descriptors

In questo package sono presenti i `Descriptor` ed i `CellEditor`, due classi che servono per fornire un supporto personalizzato ad un elemento all'interno della `Properties View`. Sono usate, per esempio, per aprire le finestre di popup su di un elemento, oppure per aprire un FileDialog, ecc.

it.unipr.aotlab.wolf.editor.views

In questo package sono presenti le classi che implementano nuove viste, quali, per esempio, la `Validation View`.

it.unipr.aotlab.wolf.editor.windows

In questo package sono presenti tutte le classi che implementano una finestra di interazione con l'utente.

it.unipr.aotlab.wolf.editor.wizards

In questo package sono presenti tutte le classi relative ad una finestra di `Wizard`. Nella versione attuale di Wolf è presente solamente il wizard per la creazione di un nuovo file XPDL.

it.unipr.aotlab.wolf.editor.xpdl

In questo package sono presenti le classi principali dell'editor, in particolare:

- `XPDLEditor`: è la classe base all'interno della quale è implementato il core dell'editor e la finestra di `Outline`.
- `XPDLEditorActionBarContributor`: utilizzata per abilitare i comandi delle Action Bar.
- `WolfContextMenuProvider`: utilizzata per abilitare il menu contestuale.
- `WolfConstants`: contiene tutte le costanti utilizzate nel progetto.

it.unipr.aotlab.wolf.editor.xpdl.commands

In questo package sono presenti tutte le classi che implementano un `org.eclipse.gef.commands.Command`. Ogni azione all'interno dell'editor è eseguita attraverso uno specifico comando che ne gestisce tutti gli aspetti, compreso l'`Undo/Redo`.

it.unipr.aotlab.wolf.editor.xpdl.ediparts

In questo package sono presenti tutte le classi che implementano un `org.eclipse.gef.editparts.AbstractGraphicalEditPart`.
Queste rappresentano le parti grafiche dell'editor secondo lo standard imposto da GEF:

- `XPDLContainerEditPart`: è la parte principale che contiene il `Canvas` (ovvero lo sfondo) all'interno del quale è disegnato l'intero file XPDL. Può corrispondere ad un Package oppure ad un Workflow Process.

- `XPDLProcessEditPart`: è la parte relativa al Workflow Process, quando si è all'interno della vista del Package.
- `XPDLActivityEditPart`: è la parte relativa ad una Activity.
- `XPDLTransitionEditPart`: è la parte relativa ad una Transition.

`it.unipr.aotlab.wolf.editor.xpdl.ediparts.tree`

In questo package sono presenti tutte le classi che implementano un `org.eclipse.gef.editparts.AbstractTreeEditPart`.
Queste rappresentano i rami dell'albero visualizzato nell'`Outline View`: nella versione attuale di Wolf sono implementati tutti i tag XPDL v1.0.

`it.unipr.aotlab.wolf.editor.xpdl.policies`

In questo package sono presenti tutte le classi che implementano una `Policy`, ovvero che forniscono un comportamento in relazione ad un'azione dell'utente su una parte grafica.

`it.unipr.aotlab.wolf.model`

In questo package sono presenti tutte le classi di base del modello dati che rappresenta lo standard XPDL v1.0. In particolare le tre classi principali:

- `XPDLSimpleTag`: classe base che viene implementata da tutti gli oggetti del modello e che rappresenta l'elemento base, senza attributi.
- `XPDLComplexTag`: classe che rappresenta tutti quei tag che hanno un Id, un nome e degli attributi.
- `XPDLCollectionTag`: classe che rappresenta tutti quegli oggetti che fanno parte di una `Collection`, come per esempio, i `DataFields`, i `FormalParameters`, le `Activities`, ecc.

`it.unipr.aotlab.wolf.model.validation`

In questo package è presente la classe che implementa la `Validation` del file XPDL, secondo gli standard imposti dalle specifiche della versione 1.0.

it.unipr.aotlab.wolf.model.xpdltags
In questo package sono presenti tutte le classi del modello: ogni classe rappresenta un tag dello standard XPDL v1.0.

it.unipr.aotlab.wolf.xpdl.bridge
it.unipr.aotlab.wolf.xpdl.bridge.appearences
it.unipr.aotlab.wolf.xpdl.bridge.creationfactories
it.unipr.aotlab.wolf.xpdl.bridge.delegates
Le classi appartenenti a questi package sono state implementate all'inizio del progetto per fornire la possibilità di migrare l'editor su di uno standard differente: esse implementano una struttura di tipo Bridge/Proxy basandosi sul paradigma Parent/Child.

5.4 Funzionalità

In questo paragrafo si descrivono le funzionalità principali dell'editor per workflow Wolf dal punto di vista dell'utilizzatore. Come descritto in precedenza, l'editor è stato sviluppato come un plug-in della piattaforma Eclipse e, di conseguenza, viene rilasciato come un pacchetto "`.jar`" da inserire all'interno della directory "`plugins`" di installazione di Eclipse. All'interno del manifest file del plug-in è stato associato un editor visuale a tutti i file con estensione "`.xpdl`", per cui, cliccando su di un file con tale estensione presente nel workbench, si accede direttamente all'editor. E' stato realizzato anche un Wizard per la creazione di un nuovo file XPDL (Figura 17): tale Wizard, accessibile direttamente dal menu "`File`", permette la creazione guidata di un file XPDL ed istanzia in modo automatico la parte visuale dell'editor per la realizzazione del workflow.

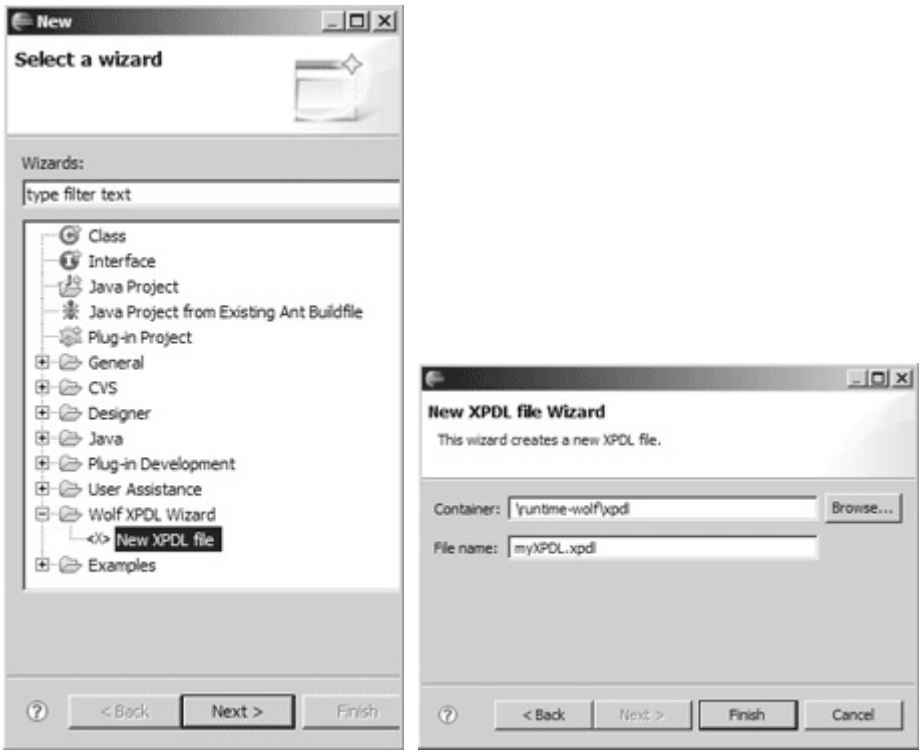

Figura 17 - Wizard per la creazione di un nuovo file XPDL

5.4.1 L'Editor

Una volta aperto un file XPDL, nella parte centrale della piattaforma Eclipse, viene istanziato l'editor per workflow Wolf. Esso è composto da due parti principali: il `Canvas`, ovvero la lavagna sulla quale si disegna il workflow e la `Palette`, ovvero la barra degli strumenti che contiene gli oggetti che si possono disegnare all'interno della lavagna. La `Palette` è stata configurata in modo che sia sensibile al contesto: questo significa che, in funzione del tipo di oggetto che si sta creando, permette di inserire all'interno dell'editor solo ciò che è compatibile con tale oggetto. In Figura 18 è rappresentato un editor con relativa `Palette`: in particolare è stato creato un nuovo file XPDL con al proprio interno quattro WorkflowProcess.

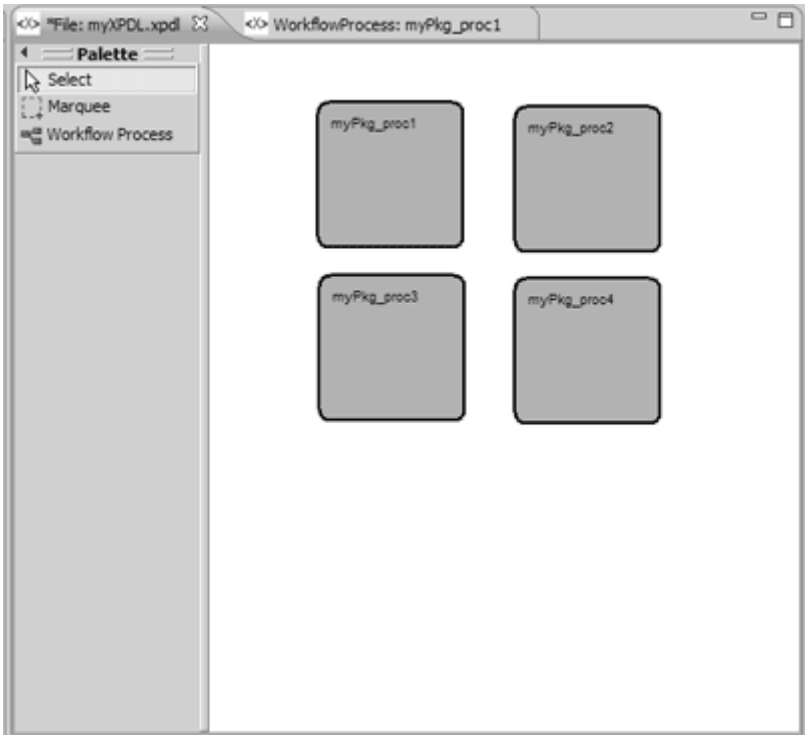

Figura 18 - Istanza dell'Editor con Palette

Una funzionalità molto importante, che non è presente in nessun editor in commercio e che è fornita dalla piattaforma Eclipse, è la possibilità di suddividere la visualizzazione di ogni elemento base del workflow (sia esso un Package od un WorkflowProcess) all'interno di un'istanza dedicata. In questo modo è possibile visualizzare contemporaneamente più oggetti dello stesso workflow e, al tempo stesso, concentrarsi sulle funzionalità fornite da ogni singolo elemento. Selezionando, per esempio, un WorkflowProcess, si apre una nuova istanza dell'editor che ne rappresenta il contenuto. In Figura 19 si può vedere un WorkflowProcess piuttosto complesso che rappresenta le azioni da compiere per configurare un servizio di ADSL dal punto di vista del fornitore. In particolare si nota come la `Palette` sia popolata con gli oggetti Activity compatibili con il

WorkflowProcess selezionato. La versione attuale dell'editor supporta cinque tipi di Activity, ovvero tutte quelle possibili definite dallo standard XPDL v1.0: `Route`, `No Implementation`, `Tool`, `Subflow` e `Block`. Le ultime due sono fondamentali in quanto permettono di invocare blocchi di altre Activity (possono essere considerate come delle librerie esterne) oppure altri workflow. In questo modo è possibile sfruttare il linguaggio XPDL, assimilandolo ad un linguaggio di programmazione ad oggetti, per semplificare la composizione e la creazione di nuovi servizi.

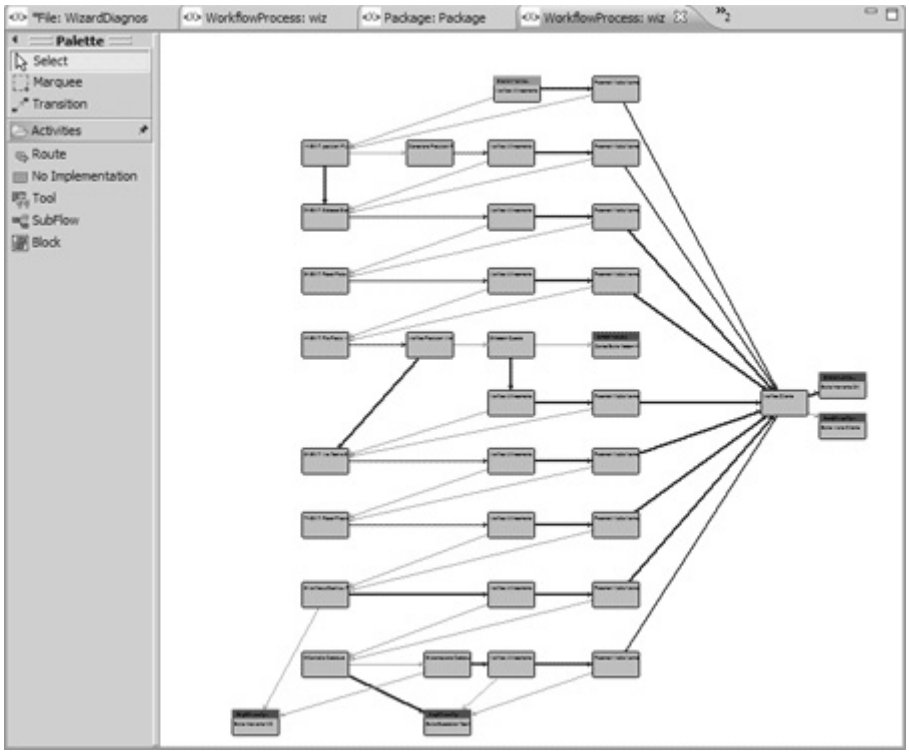

Figura 19 - Visualizzazione di un WorkflowProcess

Lo standard XPDL v1.0 impone che ogni workflow abbia un'unica Activity di tipo Start ed una o più Activity di tipo End. L'Activity di tipo Start è identificata all'interno dell'editor da una barra superiore di colore verde e viene impostata

automaticamente nel caso in cui non sia indicata dall'utente, mentre l'Activity di tipo End deve essere impostata dall'utente ed è identificata tramite una barra rossa.

5.4.2 L'Outline

Una View fondamentale per qualsiasi editor grafico realizzato all'interno della piattaforma Eclipse, è l'`Outline`. Tale View è stata progettata per rappresentare in modo rapido e con una visualizzazione il più intuitiva possibile la struttura degli oggetti visualizzati e selezionati nell'Editor principale. Per rappresentare un oggetto di XPDL, si è scelto di implementare all'interno dell'`Outline` un albero che ne raffigurasse tutte le dipendenze. In Figura 20 è rappresentata l'`Outline View` nel caso del WorkflowProcess relativo alla configurazione del servizio ADSL. All'interno dell'`Outline` è anche possibile visualizzare una miniatura dell'intero workflow, nel caso in cui questo sia molto complesso e si necessiti comunque di una visione grafica globale della relativa struttura.

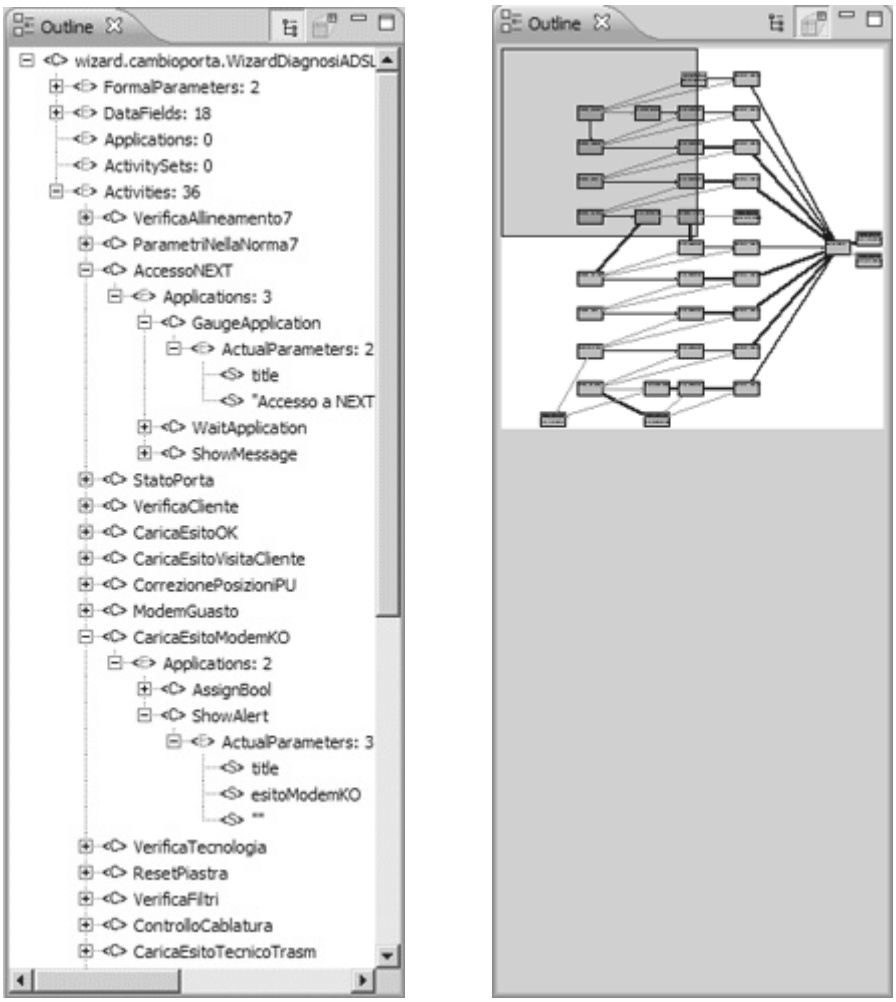

Figura 20 - Outline View: vista ad albero e overview.

5.4.3 La Properties View

Per ogni oggetto rappresentato all'interno dell'editor o selezionato dall'`Outline` è stata implementata una finestra di proprietà all'interno della `Properties View`. Tale finestra visualizza le informazioni principali sui valori dei vari tag XPDL che rappresentano l'oggetto in focus e ne permette la modifica.

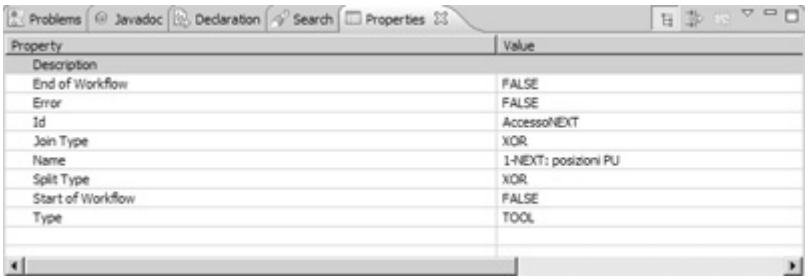

Figura 21 - Properties View

Per incontrare le esigenze di utilizzo dei workflow per la composizione di servizi tramite un sistema ad agenti, è stato necessario estendere alcune delle funzionalità fornite dallo standard XPDL v1.0. Per fare questo, è stato sfruttato il tag `Extended Attribute` messo a disposizione proprio per questo scopo dallo standard. Grazie a tale tag è stato possibile definire delle proprietà specifiche per l'esecuzione di un agente o per la creazione di codice Java. Questa flessibilità del linguaggio XPDL è stato uno dei motivi principali che ha portato alla sua scelta in fase di definizione delle specifiche dell'intero sistema.

Tutte le proprietà rappresentate nella `Properties View` sono editabili dall'utente in modo diretto, attraverso un menù a tendina o, più spesso, attraverso una procedura guidata che lo aiuta nella scelta di parametri o nella configurazione di un determinato servizio. In questo modo l'editor non è solamente un'applicazione grafica che disegna un workflow, ma diventa uno strumento per la creazione di tale workflow che assiste l'utente nell'intero processo e lo guida nella generazione di un oggetto aderente allo standard e coerente.

5.4.4 Funzionalità Avanzate

L'editor per workflow Wolf si presenta nella sua versione completa come rappresentato in Figura 22.

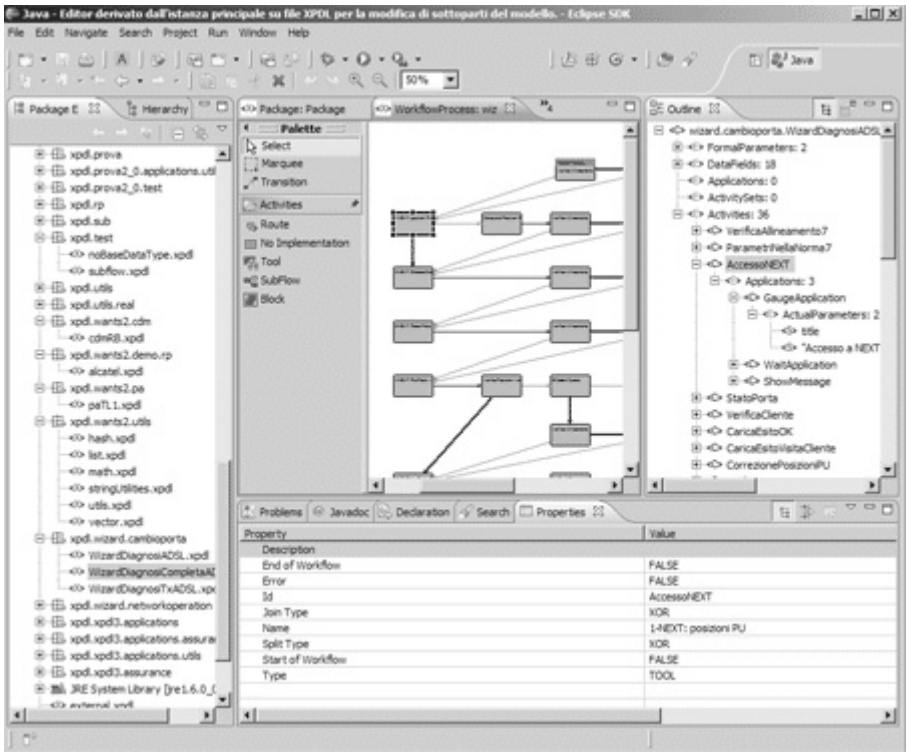

Figura 22 - L'interfaccia completa dell'editor Wolf

Esso è completamente integrato nella piattaforma Eclipse e ne sfrutta appieno tutte le funzionalità ed i tool. Per gli scopi di questo lavoro di tesi si è reso necessario estendere tali funzionalità in modo da soddisfare le richieste specifiche del particolare ambito di applicazione. Per questo motivo sono state aggiunte delle estensioni al plug-in principale, create appositamente per la gestione migliorata di un file XPDL e la traduzione di tale codice in linguaggio Java.

5.4.4.1 Validation

Come descritto in precedenza, l'editor non è solamente uno strumento grafico, ma fornisce costantemente un supporto all'utente controllandone le azioni e suggerendo delle soluzioni. Uno strumento fondamentale che è stato inserito al suo interno è la Validation, ovvero la possibilità di validare il workflow che si sta

creando. La validation viene eseguita ad ogni salvataggio del file XPDL, alla sua chiusura oppure su esplicita richiesta dell'utente. Con il termine validation si intende una verifica automatica della adesione del file XPDL che si sta creando allo standard XPDL v1.0. Tale controllo è principalmente di tipo sintattico e viene effettuato confrontando il file con lo schema XSD fornito dalla WfMC: `TC-1025_schema_10_xpdl.xsd` [34]. Il controllo, quindi, si limita a verificare che i costrutti utilizzati siano stati inseriti correttamente e che non vi siano errori sintattici.

In realtà, un controllo di questo tipo si è rivelato immediatamente troppo superficiale, per cui, soprattutto in relazione alle estensioni che si è reso necessario inserire al linguaggio XPDL, sono stati aggiunti dei controlli semantici. In totale, nella versione attuale vengono effettuati quattro tipi di controlli:

- SINTATTICO: secondo lo schema XPDL v1.0.
- CONNETTIVO: vengono verificate tutte le connessioni fra gli oggetti, controllando che siano tutti connessi e che le condizioni sulle transizioni possano verificarsi e conducano a risultati validi.
- TOPOLOGICO: si verifica che non vi siano cicli nel grafo che definisce il workflow.
- LOGICO: si verifica che il file XPDL sia coerente, per esempio controllando che tutte le variabili invocate siano state definite, eventuali workflow o blocchi esterni siano stati inclusi, che non vi siano variabili od applicazioni inutilizzate, ecc.

I risultati della validation vengono rappresentati all'interno di una nuova View appositamente creata e chiamata `Validation View`. Come si può notare dalla Figura 23, all'interno di tale view sono stati separati i quattro tipi di validazione e ad ognuno è stata riservata un'area. Nel caso in cui si verifichi un errore, l'editor lo segnala all'utente all'interno della `Validation View`: l'utente ha la possibilità di cliccare direttamente sugli oggetti implicati nell'errore e visualizzarli immediatamente all'interno dell'editor per una modifica diretta.

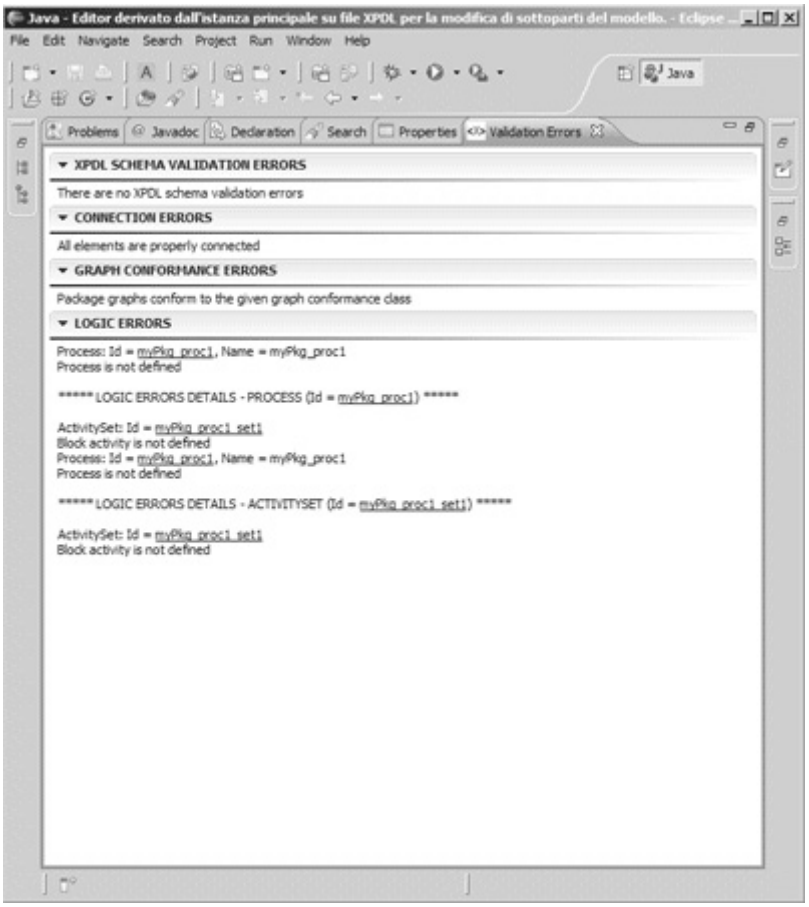

Figura 23 - Validation View

5.4.4.2 Editor delle Applicazioni

Come descritto già in precedenza, l'editor per workflow è stato sviluppato per permettere ad un utente del sistema di definire un proprio processo complesso, componendo servizi atomici, e di porlo in esecuzione su di un sistema ad agenti. Per questo motivo si è reso necessario estendere il linguaggio XPDL, affinché supportasse l'inserimento di codice Java direttamente all'interno del file, come se si trattasse dell'invocazione di un metodo. Per fare ciò, è stato definito un nuovo `ExtendedAttribute` sul tag Application per contenere il codice Java.

L'invocazione di un metodo o, più in generale, di codice Java, si risolve con l'invocazione di un'Application che contiene tale codice.

Per favorire la scrittura di tale codice Java direttamente all'interno dell'editor di workflow, è stato implementato un sistema integrato di creazione dell'Application, sfruttando il fatto che l'editor è stato sviluppato nella piattaforma Eclipse che ha già al suo interno un editor Java. Quando l'utente richiede la creazione di una nuova Application, viene eseguito automaticamente l'editor Java dal quale può creare il codice desiderato, sfruttando così tutte le funzionalità di editing proprie del Java Development Toolkit della piattaforma Eclipse. Il codice Java da inserire all'interno del file XPDL segue un determinato template, imposto in fase di definizione del sistema, in modo da avere un'invocazione univoca del codice stesso. E' stato, quindi, implementato un meccanismo per cui è possibile sia generare un'Application direttamente da codice Java ("Import from Code"), sia generare il template del codice Java partendo dalla definizione dell'Application ("Generate Code"). Di seguito si riporta un esempio di come il codice Java viene mappato all'interno del file XPDL.

```
package unipr.ce.testWf2;

import . . .

public class FillStr extends Application {
  public static final String DEST_ID0 = "DEST";
  public static final String SOURCE_ID1 = "SOURCE";
  public static final String MAXLEN_ID2 = "MAXLEN";

  public FillStr() {
  // Initialize Formal Parameters
  formalParams = new jade.util.leap.ArrayList();
  Parameter p = new Parameter();
  p.setName(DEST_ID0);
  p.setMode(Constants.OUT_MODE);
  p.setType(STRING);
  formalParams.add(p);
  p.setName(SOURCE_ID1);
  p.setMode(Constants.IN_MODE);
  p.setType(STRING);
  formalParams.add(p);
```

```
  p.setName(MAXLEN_ID2);
  p.setMode(Constants.IN_MODE);
  p.setType(INTEGER);
  formalParams.add(p);
}

  public void execute() throws Throwable {
    int len = SOURCE.length();
    if (len>MAXLEN.intValue()) {
      DEST = SOURCE.substring(0, MAXLEN.intValue());
    } else {
      DEST=SOURCE + "0000000000".substring(0,
         MAXLEN.intValue()-len);
    }
  }
}
```

Figura 24 - Esempio di codice Java per l'invocazione di un'Application

```
<Application Id="FillStr" Name="FillStr">
  <FormalParameters>
    <FormalParameter Id="DEST" Index="DEST" Mode="OUT">
      <DataType><BasicType Type="STRING"/></DataType>
    </FormalParameter>
    <FormalParameter Id="SOURCE" Index="SOURCE" Mode="IN">
      <DataType><BasicType Type="STRING"/></DataType>
    </FormalParameter>
    <FormalParameter Id="MAXLEN" Index="MAXLEN" Mode="IN">
      <DataType><BasicType Type="INTEGER"/></DataType>
      </FormalParameter>
    </FormalParameters>
    <ExtendedAttributes>
      <ExtendedAttribute Name="Java.Code" Value="int len =
SOURCE.length();&#10;if (len&gt;MAXLEN.intValue()) {&#10;
DEST = SOURCE.substring(0, MAXLEN.intValue());&#10;}
else {&#10; DEST=SOURCE +
&quot;0000000000&quot;.substring(0, MAXLEN.intValue()-
len);&#10;}"/>
      <ExtendedAttribute Name="Java.Import"
        Value="java.lang.String"/>
    </ExtendedAttributes>
</Application>
```

Figura 25 - Esempio di codice XPDL di un'Application

5.4.4.3 Editor XPDL Testuale

Oltre alla visualizzazione grafica standard è stato implementato un editor testuale di file XPDL per permetterne la lettura o la modifica direttamente dal codice. L'editor testuale è stato implementato estendendo un plug-in già presente all'interno della piattaforma Eclipse: Xml Author [29], un generico editor testuale per il linguaggio XML. Esso è stato esteso per supportare appieno il linguaggio XPDL v1.0. Grazie all'uso di tale plug-in è possibile visualizzare la struttura ad albero del file XPDL, che si è creato sia all'interno della finestra dell'editor che nella finestra dell'Outline. Le modifiche che vengono effettuate sul file a livello testuale si riflettono immediatamente anche sull'editor grafico e viceversa, rendendo l'uso dei due strumenti uniforme come se si trattasse di un'unica applicazione.

Figura 26 - Editor XPDL Testuale

5.5 Sviluppi Futuri

La versione attuale dell'editor di workflow Wolf è perfettamente funzionante e supporta interamente lo standard XPDL v1.0. Come elencato nei paragrafi

precedenti, sono state anche aggiunte delle funzionalità non propriamente legate allo standard XPDL, per facilitare la creazione di un workflow complesso. Esistono, però, ancora alcune migliorie che possono essere apportate e che si è riusciti ad includere nella versione descritta in questo lavoro di tesi. In particolare si elencano due funzionalità che sono considerate molto importanti per il corretto completamento dell'editor.

La prima riguarda l'integrazione dell'editor all'interno del sistema W-GAIN o, ancora meglio, l'integrazione dell'intero sistema W-GAIN – Editor all'interno della piattaforma Eclipse. Come descritto nel capitolo 4, il sistema W-GAIN è stato sviluppato utilizzando la libreria SWT, ma non è stato integrato all'interno della piattaforma Eclipse. Esso si presenta come un'applicazione stand alone che ha la capacità di integrare al proprio interno una piattaforma ad agenti ed è in grado di invocarne l'esecuzione e controllarne lo stato. Sono stati effettuati alcuni studi sulla possibilità di integrare l'intera piattaforma JADE all'interno di Eclipse ed hanno portato ad un esito sicuramente positivo. Sarà di indubbio interesse in futuro pensare di poter inserire il sistema W-GAIN con l'intera piattaforma multi-agente all'interno della piattaforma Eclipse. In questo modo si migliorerebbe l'integrazione fra la fase di creazione dei workflow e la loro esecuzione da parte degli agenti. Al momento, infatti, una volta che viene realizzato un workflow utilizzando l'editor Wolf, viene generato un insieme di file XPDL. Tali file vengono inviati tramite il protocollo HTTP ad un agente, che ha il solo scopo di interfacciarsi con il sistema W-GAIN e trasferire i file all'interno del Web Server integrato, in modo che possano essere utilizzati dai Worflow Manager.

Una seconda importante espansione dell'editor, a seguito dell'integrazione descritta in precedenza, è quella di inserire un sistema di discovery automatico dei servizi per una più semplice creazione del workflow. Al momento della richiesta di creazione di un file XPDL l'utente avrebbe la possibilità di interrogare il sistema per ottenere la lista dei servizi disponibili in quel momento, con relativa descrizione specifica, effettuando direttamente una richiesta ai Legacy Agent. In questo modo sarebbe notevolmente agevolato nella creazione del workflow e

potrebbe anche essere assistito da un agente a lui dedicato che lo guiderebbe nella fase di definizione dei processi da creare.

6
XPDL2JADE - Tradurre XPDL in Java

In questo capitolo si descrivono l'architettura e le caratteristiche principali di un software che permette la conversione di un workflow, descritto tramite il linguaggio XPDL, in codice Java eseguibile direttamente all'interno della piattaforma ad agenti JADE.

6.1 Introduzione

Una volta definito un file XPDL tramite l'editor di workflow descritto al capitolo 5, nasce la necessità di porlo in esecuzione su di un sistema distribuito multi-agente. Per fare ciò, è stato necessario implementare un traduttore che prendesse in ingresso un file XPDL e generasse in uscita del codice Java direttamente eseguibile da un agente JADE. Tale software, chiamato XPDL2JADE, è descritto graficamente nella Figura 27: esso si pone come ponte fra il mondo dei workflow e quello degli agenti, in particolare fra l'editor di workflow Wolf ed il sistema W-GAIN.

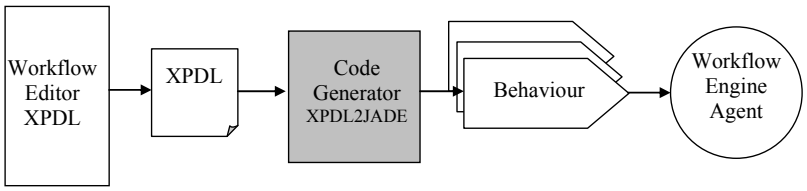

Figura 27 - Architettura della catena XPDL - JADE

6.2 Specifiche

Lo scopo del traduttore XPDL2JADE è quello di riconoscere tutti i costrutti dello standard XPDL v1.0 e di trasformarli in codice Java eseguibile direttamente tramite un sistema ad agenti JADE. Per fare ciò, si sfrutta il sistema dei `Behaviour` messo a disposizione dalla piattaforma JADE: un agente esegue le proprie operazioni sotto forma di `Behaviour` che vengono posti in una coda dedicata e richiamati in sequenza. Esistono vari tipi di `Behaviour`, ognuno specifico per un determinato modo di esecuzione. Si è scelto di utilizzare il `FSMBehaviour`, ovvero un `Behaviour` che esegue le operazioni nella propria coda seguendo le regole di una macchina a stati finiti. In questo modo è possibile simulare il funzionamento di tutti i costrutti XPDL, compresi anche gli `Split` ed i `Join`, siano essi di tipo `AND` che di tipo `XOR`.

6.3 Struttura delle Classi

In questo paragrafo si fornisce una breve descrizione della struttura delle classi del traduttore XPDL2JADE e si dettagliano le funzionalità delle classi principali.

`it.unipr.aotlab.xdl2jade.common.MappingFactory`
Questa classe è utilizzata per mappare i dati fra il parser XPDL ed il traduttore. Nella versione attuale, il traduttore XPDL2JADE, tramite la classe `MappingFactory`, riceve dal parser un oggetto `Package` che rappresenta

l'intero contenuto del file XPDL. Da questo oggetto, esso estrae i valori di interesse e si crea una struttura parallela contenente tali valori sotto forma di stringhe. Se, in futuro, si volesse modificare la struttura restituita dal parser, bisognerebbe inserire nella classe `MappingFactory` il codice che effettua la mappatura fra la vecchia struttura e la nuova. Si è scelto di seguire questa strada affinchè la struttura di oggetti generata dal parser fosse molto semplice e standard. Esso, infatti, è un software sviluppato appositamente per gestire i file XPDL e realizza una gerarchia di oggetti che si mappano perfettamente sui vari tag del file XPDL. E' presumibile, quindi, che qualunque software effettui il parsing del file, generi una struttura molto simile a quella attuale, per cui le modifiche da apportare all'oggetto `Package` risulterebbero minime.

`it.unipr.aotlab.xdl2jade.output.JavaGenerator`

E' la classe responsabile della generazione di un nuovo file Java: tale file viene creato utilizzando un singleton della classe `FileManager`. Dopo aver creato il file di output, vengono generati i vari oggetti per l'estrazione dei dati dalla struttura in memoria fornita dalla `MappingFactory`.

`it.unipr.aotlab.xdl2jade.util.FileManager`

E' la classe responsabile della gestione dei file di output, in particolare è in grado di creare un nuovo file aprendo un nuovo flusso di stream, chiudere un file precedentemente aperto e scrivere una nuova riga di testo nel file.

`it.unipr.aotlab.xdl2jade.common.XpdlJade`

E' la classe principale del programma contenente il metodo `main()`. Per il controllo dei dati inseriti da riga di comando è stato utilizzata la libreria `JSAP_1.03a` [24], distribuita con licenza LGPL [19].

`it.unipr.aotlab.xdl2jade.common.xml.XpdlParser`

E' la classe responsabile dell'operazione di parsing di un file XPDL. Tale classe utilizza un'istanza del parser:

```
Package pkg=it.unipr.aotlab.common.XPDL.parseDocument(inputStream);
```

Il metodo `parseDocument(java.io.Reader inputFileOrString)` accetta come parametro un qualunque `Reader` (nella versione attuale sono supportati lo `StringReader` ed il `FileReader`) e restituisce un oggetto di tipo `Package` che contiene tutti i dati contenuti nel file XPDL. Questo oggetto `Package` viene passato alla `MappingFactory` che provvede a creare una struttura di dati più adatta per essere tradotta in file Java.

`it.unipr.aotlab.xdl2jade.common.X2JBoot`

E' la classe responsabile dell'esecuzione embedded del traduttore: per maggiori informazioni riguardo alle modalità di esecuzione si faccia riferimento al paragrafo 6.5.2. Per richiamare un'istanza del traduttore, è stato implementato il metodo `run()`. In caso di problemi durante la traduzione, tale metodo lancia un'eccezione che deve essere poi gestita all'interno del codice chiamante. Nella versione attuale possono essere lanciati quattro tipi di eccezione: una `RequiredTagException` nel caso in cui venga a mancare un `<tag>` obbligatorio in un file XPDL, una `IOException` nel caso ci siano problemi nella gestione dei file di input/output, una `ParserException` nel caso in cui si verifichi un errore durante il parsing del file XPDL oppure una generica `Exception` nel caso in cui si verifichi un crash inaspettato del programma.

`it.unipr.aotlab.xdl2jade.common.InitLogger`

Nella versione attuale è stata utilizzata la libreria `log4j` [4] per effettuare il logging delle informazioni relative alla traduzione. La classe `InitLogger` è stata creata per inizializzare il logger e gestirne l'esecuzione.

6.4 Funzionalità

Il traduttore XPDL2JADE supporta tutti i costrutti del linguaggio XPDL v1.0 ed anche tutte le estensioni che sono state aggiunte al linguaggio per incontrare le esigenze del sistema W-GAIN e dell'editor per workflow Wolf. Di seguito si fornisce un elenco schematico delle principali funzionalità del traduttore:

- verifica della correttezza sintattico/semantica del file XPDL.
- Creazione di un file Java che estende la classe `FSMBehaviour` per ogni WorkflowProcess presente all'interno del file XPDL in input. Il nome di tale classe coincide con l'`Id` del WorkflowProcess.
- Inserimento diretto dei Workflow Relevant Data nel database del sistema.
- Inizializzazione dei Workflow Relevant Data con i rispettivi `Initial Value`: nel caso non sia indicato tale valore viene generato un WARNING e viene utilizzato un valore di default.
- Inizializzazione della lista dei package inserendovi eventuali package esterni. In quest'ultimo caso viene anche realizzato il parsing del file XPDL relativo a tali package per estrarre informazioni eventualmente utili per il workflow corrente.
- Registrazione delle Activity e delle Transition del WorkflowProcess.
- Riconoscimento di tutti i tipi di Activity definite dallo standard XPDL v1.0: `Route`, `NoImplementation`, `Tool`, `Subflow`, `BlockActivity`.
- Creazione di una inner class per eventuali ActivitySet estendendo la classe `BlockActivityBehaviour`.
- Riconoscimento del Package in cui è inserita la descrizione di un'Application, anche se questo è un Package esterno a quello corrente.

- Riconoscimento di tutti i tipi di Transition definite dallo standard XPDL v1.0: `Condition`, `NoCondition`, `Otherwise`, `Exception`, `DefaultException`.
- Manipolazione delle espressioni algebriche sulle Transition, sostituendo gli operandi `AND`, `OR`, `NOT` con i valori attuali dei workflow relevant data opportunamente castati.
- Realizzazione del mapping fra i tipi base di XPDL ed i rispettivi oggetti Java secondo lo schema illustrato nella tabella seguente.

XPDL basic type	Java class
STRING	`java.lang.String`
INTEGER	`java.lang.Integer`
FLOAT	`java.lang.Double`
DATETIME	`java.util.Date`
REFERENCE	`java.lang.Object`
BOOLEAN	`java.lang.Boolean`

6.5 Esecuzione

Il traduttore XPDL2JADE nella sua versione attuale è un'applicazione stand alone che può essere eseguita all'interno della piattaforma Eclipse come un classico applicativo Java. Esistono due modalità differenti di esecuzione, realizzate per incontrare le esigenze di integrazione all'interno del sistema W-GAIN.

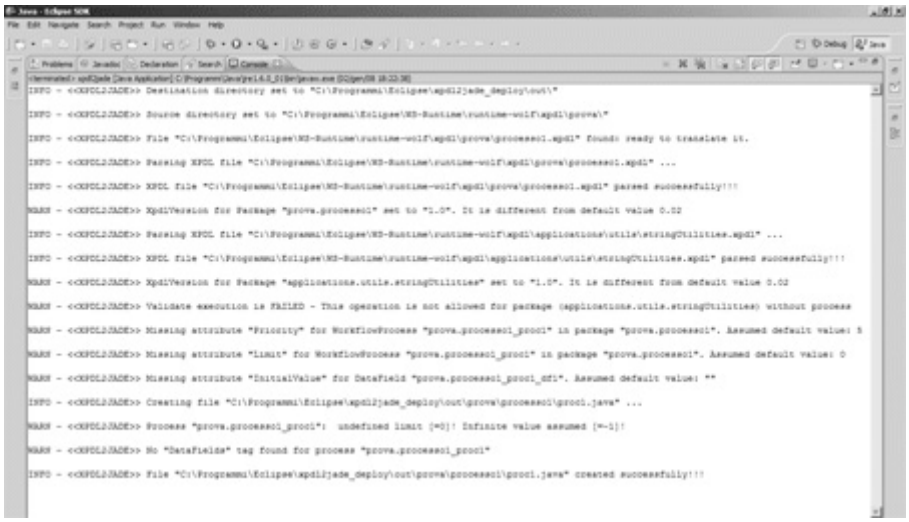

Figura 28 - XPDL2JADE eseguito nella piattaforma Eclipse.

6.5.1 Linea di Comando

XPDL2JADE è un programma Java stand alone che può essere lanciato tramite linea di comando su qualunque macchina che abbia una Virtual Machine Java installata:

```
java -cp lib\xpdl2jade.jar;lib\exprParser_1.0.jar;lib\jawe.jar;
lib\JSAP_1.03a.jar;lib\log4j-1.2.9.jar
it.unipr.aotlab.xdl2jade.common.Xpdl2jade <nomeFile>.xpdl
```

Da linea di comando è possibile passare al traduttore alcune informazioni utili per la generazione dei file Java. In particolare sono stati previsti i seguenti attributi:

`[-s <source>]:` tramite questa opzione è possibile indicare una directory diversa rispetto alla corrente, in cui sono contenuti i file XPDL da tradurre. Se viene utilizzata questa opzione, il traduttore cercherà i file XPDL solamente all'interno della directory indicata.

[-d <destination>]: tramite questa opzione è possibile indicare una directory diversa rispetto alla corrente, in cui saranno generati i file Java. Nel caso in cui tale directory non esista, verrà creata dal traduttore.

[-v]: tramite questa opzione viene abilitata la modalità "`verbose`", grazie alla quale vengono visualizzate informazioni più dettagliate sulle operazioni compiute dal traduttore. Nel traduttore è presente un logger a livelli basato sulla libreria Java `log4j`. Se tale opzione non viene indicata in fase di esecuzione da linea di comando, il logger viene automaticamente settato a livello di WARNING.

[xpdlFile1 … **xpdlFileN]:** come ultimo parametro il traduttore accetta un elenco di file XPDL separati da uno spazio. Se non viene indicato nessun file, il traduttore effettuerà una ricerca all'interno della directory `<source>` e procederà alla traduzione di tutti i file XPDL presenti. I file XPDL devono essere riportati con il nome completo dell'estensione: es. `testWf.xpdl`.

Nel caso in cui non venga indicato nessun parametro, le directory `<source>` e `<destination>` vengono impostate sulla directory corrente, ovvero quella da cui è stato eseguito il programma.

Nel caso in cui non vengano indicati nomi di file XPDL, saranno tradotti tutti i file presenti nella directory `<source>`.

6.5.2 Embedded

E' stata inserita la possibilità di eseguire il traduttore XPDL2JADE in modalità embedded, ovvero all'interno di una qualunque altra applicazione Java. Per fare ciò, è stata inserita la classe `X2JBoot`: per prima cosa è necessario creare una nuova istanza di questa classe. All'interno della classe `X2JBoot` è stato implementato il metodo pubblico `run()`, tramite il quale si invoca un'istanza del traduttore. Prima di richiamare il metodo `run()`, è possibile impostare tutti i parametri supportati dal traduttore, elencati al paragrafo precedente. Per ognuno dei

parametri è stato implementato un metodo `setXXX()` al quale si passa il valore del parametro. Di seguito è riportata una tabella con l'elenco completo di tali metodi.

Metodo	Descrizione parametro
`setSource(String path)`	La directory sorgente in cui sono presenti i file XPDL che devono essere tradotti.
`setDestination(String path)`	La directory destinazione all'interno della quale il traduttore inserirà i file .java di traduzione.
`setLogger(int type)`	Il tipo di logger che si vuole utilizzare: · 0 = log su file di testo · 1 = log su console Nel caso non venga utilizzato il metodo `setLogger(type)` di default, è utilizzato il logger su console.
`setXpdlList(List xpdlNames)`	Una lista di stringhe contenenti il nome completo dei file XPDL da tradurre.
`setVerbose(boolean value)`	Un booleano che indica se si vogliono informazioni dettagliate sull'esecuzione. L'utilizzo di questo metodo corrisponde al settaggio del livello di INFO per il logger.
`setLoggerLevel(int level)`	Il livello al quale si intende settare il logger. Ogni messaggio di un livello inferiore a quello indicato verrà ignorato.

Nel caso in cui si verifichino degli errori bloccanti durante la traduzione, viene lanciata un'eccezione dal metodo `run()`. E' lasciata al programmatore dell'applicazione chiamante la gestione di tale eccezione. Il lancio di un'eccezione corrisponde sempre ad un messaggio di log di livello `FATAL`.

Per maggiore chiarezza si riporta un esempio di codice per l'esecuzione embedded.

```
try {
unipr.ce.common.X2JBoot boot =
  new unipr.ce.common.X2JBoot();
  boot.setLogger(0);    // logger su file di testo
  boot.setLoggerLevel(3); // livello WARNING
  boot.setSource("xpdl");    // directory sorgente
  boot.setDestination("out");    // directory di output
  ArrayList files = new ArrayList();
  files.add("testwf.xpdl");
  files.add("testdelegation.xpdl");
  boot.setXpdlList(files);    // aggiunti due file per la
traduzione
  boot.run();     // invocazione del traduttore
}
catch(Exception ex) {
    // gestione dell'eccezione da parte dell'applicazione
chiamante
}
```

6.6 Funzionalità Avanzate

Oltre alla semplice traduzione del linguaggio XPDL, il traduttore XPDL2JADE effettua anche una serie di operazioni più complesse che lo rendono indispensabile per il corretto funzionamento del sistema W-GAIN. In particolare, un aspetto fondamentale sul quale è stata posta grande enfasi ed interesse durante

lo sviluppo, è quello relativo alla gestione delle espressioni. Il problema principale, infatti, nell'uso di workflow per la gestione di un sistema ad agenti, è l'ampio uso all'interno di un file XPDL di espressioni. Esse vengono usate per definire il valore delle variabili, per effettuare dei calcoli o, soprattutto, per definire il verificarsi o meno di condizioni sulle transizioni. Una corretta interpretazione delle espressioni, quindi, è fondamentale per la giusta valutazione delle funzionalità del workflow e, di conseguenza, per il corretto funzionamento del sistema ad agenti. Il traduttore XPDL2JADE è in grado di riconoscere qualunque espressione che utilizzi i tipi base definiti da XPDL v1.0 e di tradurla in codice Java eseguibile. In particolare, è stata implementata anche la possibilità di effettuare confronti fra stringhe utilizzando i metodi messi a disposizione dal linguaggio Java. Il traduttore, infine, è in grado di ricavare il tipo ritornato da un'espressione, effettuando tutte le conversioni del caso.

Un'ultima funzionalità importante, inserita nel traduttore, è la possibilità di tradurre il codice XPDL in formato stringa. Questo è molto utile nel caso in cui non si abbia a disposizione il file XPDL, ma sia stato memorizzato l'intero contenuto all'interno di una variabile di tipo String, per esempio all'interno di un database o di un estratto di un altro programma Java.

Conclusioni

Nel lavoro di tesi descritto è stata studiata la possibilità di sfruttare le caratteristiche di un sistema ad agenti, basato sulla piattaforma JADE, per la gestione e la composizione di servizi distribuiti. Tali servizi vengono esposti al sistema come Web Services e la configurazione e rappresentazione degli stessi viene effettuata tramite l'uso di workflow.

In particolare è stato realizzato il prototipo di un sistema ad agenti chiamato W-GAIN. Tale sistema è basato sulla piattaforma JADE e prevede di fornire supporto all'utente sia nella fase di esecuzione che in quella di sviluppo di applicazioni distribuite basate sui workflow. Per migliorare tale sistema e, soprattutto, per facilitare l'interazione con l'utente è stato necessario sviluppare due ulteriori tool. Il primo, un editor per workflow, permette all'utente di comporre in modo grafico i servizi messi a disposizione dal sistema, tramite l'uso del linguaggio XPDL. Grazie a tale editor, l'utente è in grado di generare in modo assistito e semplificato un processo di business e di porlo in esecuzione sul sistema W-GAIN. Il secondo, un traduttore fra linguaggi, effettua una traduzione della descrizione di un processo (realizzata in XPDL) in codice Java direttamente eseguibile su di un sistema ad agenti. L'uso del traduttore si è rivelato fondamentale per il funzionamento dell'intero progetto, in quanto permette ad un utente, con poca esperienza di programmazione su sistemi JADE, di controllarne ugualmente l'esecuzione.

L'intero studio presentato in questo lavoro di tesi ha avuto come applicazione pratica un progetto effettuato in collaborazione con Telecom Italia Lab, grazie al quale è stato possibile verificarne la fattibilità e testarne le prestazioni in un

ambiente reale. Questo aspetto ha avuto una notevole importanza in quanto ha permesso di comprendere e verificare direttamente sul campo come un sistema multi-agente sia perfettamente in grado di funzionare e mantenere una struttura distribuita basata sui servizi. Nello stesso tempo, si è potuto constatare come un sistema di composizione a workflow possa aiutare l'utente nel processo di creazione di un'attività complessa di business, evitando di acquisire conoscenze approfondite sui processi di gestione dei servizi e del sistema stesso. Il sistema ad agenti, inoltre, si è dimostrato perfettamente adatto per la gestione delle problematiche descritte, confermandosi un approccio migliore rispetto alla definizione tradizionale di SOA, grazie alla sua maggiore flessibilità, adattabilità ed autonomia.

Il sistema W-GAIN rappresenta un progetto molto ambizioso. Nel lavoro di tesi descritto è stato trattato un suo primo prototipo con la definizione dell'architettura globale e dei servizi di base. Le scelte effettuate hanno avuto come obiettivo la necessità di evitare forzature che provocassero limitazioni concettuali ed hanno portato allo sviluppo di un'architettura a moduli che agevola le operazioni di integrazione con tecnologie differenti. Rimane aperta, quindi, la possibilità in futuro di migliorarne ed estenderne le funzionalità fornendo, per esempio, il supporto a standard e nuove tecnologie che dovessero via via affermarsi sul mercato.

Elenco delle Sigle

ACID	**A**tomicity, **C**onsistency, **I**solation, **D**urability
ACL	**A**gent **C**ommunication **L**anguage
AMS	**A**gent **M**anagement **S**ystem
API	**A**pplication **P**rogramming **I**nterface
AWT	**A**bstract **W**indow **T**oolkit
B2B	**B**usiness **to B**usiness
BPEL	**B**usiness **P**rocess **E**xecution **L**anguage
BPM	**B**usiness **P**rocess **M**anagement
BPMN	**B**usiness **P**rocess **M**odeling **N**otation
CFP	**C**all **F**or **P**roposal
DAML	**D**ARPA **A**gent **M**arkup **L**anguage
DF	**D**irectory **F**acilitator
EJB	**E**nterprise **J**ava**B**eans
FIPA	**F**oundation for **I**ntelligent **P**hysical **A**gents
FTP	**F**ile **T**ransport **P**rotocol
GEF	**G**raphical **E**diting **F**ramework
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper **T**ext **M**arkup **L**anguage
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
IDE	**I**ntegrated **D**evelopment **E**nvironment
JACL	**Ja**va **C**ommand **L**anguage
JADE	**J**ava **A**gent **DE**velopment framework
JaWE	**Ja**va **W**orkflow **E**ditor
JDT	**J**ava **D**evelopment **T**ools
JSP	**J**ava **S**erver **P**ages
LGPL	GNU **L**esser **G**eneral **P**ublic **L**icense
LRT	**L**ong **R**unning **T**ransactions
MOA	**M**essage **O**riented **A**rchitecture

MOM	**M**essage **O**riented **M**iddleware
MTS	**M**essage **T**ransport **S**ystem
MVC	**M**odel **V**iew **C**ontroller
NAT	**N**etwork **A**ddress **T**ranslator
OGSI	**O**pen **G**rid **S**ervice **I**nfrastructure
PDE	**P**lug-in **D**evelopment **E**nvironment
RCP	**R**ich **C**lient **P**latform
RMI	**R**emote **M**ethod **I**nvocation
RPC	**R**emote **P**rocedure **C**all
SCE	**S**ervice **C**reation **E**nviroment
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
SOA	**S**ervice **O**riented **A**rchitecture
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
SWT	**S**tandard **W**idget **T**oolkit
TCL	**T**ool **C**ommand **L**anguage
UDDI	**U**niversal **D**escription **D**iscovery and **I**ntegration
UML	**U**nified **M**odeling **L**anguage
URL	**U**niform **R**esource **L**ocator
WAPI	**W**orkflow **A**pplication **P**rogramming **I**nterface
WES	**W**orkflow **E**nactment **S**ervices
WfMC	**W**orkflow **M**anagement **C**oalition
WfMS	**W**orkflow **M**anagement **S**ystem
WPDL	**W**orkwork **P**rocess **D**escription **L**anguage
WS-BPEL	**W**eb **S**ervices **B**usiness **P**rocess **E**xecution **L**anguage
WSDL	**W**eb **S**ervices **D**escription **L**anguage
WSIG	**W**eb **S**ervice **I**ntegration **G**ateway
XML	e**X**tensible **M**arkup **L**anguage
XPDL	**X**ML **P**rocess **D**efinition **L**anguage

Bibliografia

[1] A. Negri, A. Poggi, M. Tomaiuolo, “Intelligent Task Composition and Allocation through Agents”. In Proc. Emerging Technologies for Next generation GRID (ETNGRID-2005) Workshop - WETICE 2005, Linkoping, Svezia, Giugno 2005

[2] A. Negri, A. Poggi, M. Tomaiuolo, P. Turci, "Agents for e-Business Applications". Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS’06), pag. 907-914, May 8-12, 2006, Hakodate, Japan

[3] A. Negri, A. Poggi, M. Tomaiuolo, P. Turci, “Dynamic Grid Tasks Composition and Distribution through Agents”. Concurrency and Computation: Practice and Experience, 18(8):875-885, 2006

[4] Apache log4j. Sito ufficiale: *http://logging.apache.org/log4j/index.html*.

[5] Beanshell, Lightweight Scripting for Java. Sito ufficiale: *http://www.beanshell.org/*.

[6] BPMN 1.0: OMG Final Adopted Specification, February 6, 2006. Disponibile su: *http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf*.

[7] Business Process Modeling Notation. Sito ufficiale: *http://www.bpmn.org/*.

[8] DAML, The DARPA Agent Markup Language. Sito ufficiale: *http://www.daml.org/*.

[9] D. Greenwood, P. Buhler, A.Reitbauer, “Web Service Discovery and Composition using the Web Service Integration Gateway”, Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service, pp.789-790, 2005

[10] Eclipse Platform. Sito ufficiale: *http://www.eclipse.org/.*

[11] Enhydra JaWE - Graphical Java Workflow Process Editor. Sito ufficiale: http://www.enhydra.org/workflow/jawe/index.html.

[12] F. Bellifemine, A. Poggi, G. Rimassa. *Developing Multi-agent Systems with a FIPA-compliant Agent Framework*. Software Practice and Experience, 31:103-128, 2001.

[13] F. Bellifemine, A. Poggi, G. Rimassa, P. Turci, “Dalle specifiche FIPA alla piattaforma JADE un framework software per la realizzazione di sistemi multi-agente”, 2002.

[14] FIPA Architecture Board, “FIPA ACL Message Structure Specification”, FIPA TC Communication, 03/12/2002.

[15] FIPA Architecture Board, “FIPA Agent Management Specification”, FIPA TC Agent Management, 03/12/2002.

[16] FIPA (Foundation for Intelligent Physical Agents). Sito ufficiale: *http://www.fipa.org/*.

[17] Gartner Inc., “BPMS Magic Quadrant 11”. Sito ufficiale: *http://www.gartner.com/*.

[18] GEF. Sito ufficiale: *http://www.eclipse.org/gef/.*

[19] GNU Lesser General Public License. Disponibile su: *http://www.gnu.org/licenses/lgpl.html*.

[20] I. K. Lam, B. Smith, “Jacl: A Tcl Implementation in Java”, Proceedings of the Fifth Annual Tcl/Tk Workshop, Boston, Massachusetts, Luglio 1997.

[21] JADE. Sito ufficiale: *http://jade.tilab.com/.*

[22] J. Ferber, “Multi-Agent Systems”, Addison-Wesley, 1999.

[23] J. Ousterhout, “Tcl and the Tk Toolkit”, Addison-Wesley, Massachusetts, 1994.

[24] JSAP, Java Simple Argument Parser. Sito ufficiale: *http://sourceforge.net/projects/jsap/*.

[25] N. Palmer, “Undestanding The BPMN-XPDL-BPEL Value Chain”, Business Integration Journal, Novembre/December 2006, 54-55.

[26] Open Grid Services Infrastructure, OGSI, Versione 1.0, Luglio 2003. Disponibile su: *http://globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf.*

[27] Rhino: Javascript for Java. Sito ufficiale: *http://www.mozilla.org/rhino/*.

[28] Ruby Programming Language. Sito ufficiale: *http://www.ruby-lang.org/en/*.

[29] SVC Delivery – XML Author. Sito ufficiale: *http://www.svcdelivery.com/xmlauthor/*.

[30] SWT. Sito ufficiale: *http://www.eclipse.org/swt/*.

[31] W3C, Web Services Consotium, Sito ufficiale: *http://www.w3c.org/*.

[32] Workflow Management Coalition – Interface 1: Process Definition Interchange Process Model, Luglio 1998. Disponibile su: *http://www.wfmc.org/standards/docs/if19807m.pdf*.

[33] Workflow Management Coalition. Sito ufficiale: *http://www.wfmc.org/*

[34] Workflow Management Coalition, XPDL v1.0 Schema (TC-1025_schema_10_xpdl.xsd). Disponibile su: *http://www.wfmc.org/standards/docs/TC-1025_schema_10_xpdl.xsd*

[35] Workflow Process Definition Interface - XML Process Definition Language (XPDL) v2.0, Ottobre 2005. Disponibile su: *http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf*.

[36] Workflow Process Definition Interface - XML Process Definition Language (XPDL) v1.0, Ottobre 2002. Disponibile su: *http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf*.