Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

GRid-Aware serviCEs administrator: (GRACE 1.0):

"Grid-Aware serviCEs administrator: un server di Web-services, operante sotto vincoli di Qualità di Servizio"

Versione 1.0 Implementazione in ASSIST 1.3

S. Lombardo, A. Machì

RT-ICAR-PA-05-13 **Dicembre 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: www.icar.cnr.it
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: www.na.icar.cnr.it
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

GRid-Aware serviCEs administrator:
(GRACE 1.0):

"Grid-Aware serviCEs administrator: un server di Web-services, operante sotto vincoli di Qualità di Servizio"

Versione 1.0 Implementazione in ASSIST 1.3

S. Lombardo, A. Machì

Deliverable III° anno
Progetto MIUR FIRB Grid.it

Work Package 9 Librerie Scientifiche

Data Aggiornamento **Dicembre 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: www.icar.cnr.it
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: www.na.icar.cnr.it
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: www.pa.icar.cnr.it

Indice

1 Introduzione

Lo scopo del sistema proposto è quello di realizzare un server di libreria che permetta l'esecuzione remota, sotto vincoli di QoS, di componenti software HPC su risorse appartenenti ad una griglia computazionale secondo il paradigma delle Organizzazioni Virtuali scalabili orientate ai Servizi (SOA). La funzione del server è quella di amministrare *pro-attivamente* sia risorse hardware, che componenti software configurabili, per conto di applicazioni esterne, ed in base alle strategie di utilizzo dettate da queste ultime [1].
Il sistema è in grado di configurare i componenti paralleli per attuare una gamma di modalità di esecuzione (*tipologie di contratto*). Ogni componente può essere eseguito secondo le modalità per il quale dipone ed espone espone le funzioni di controllo (*monitoring* ed *enforcement*) richieste. .
Le applicazioni clienti, quando intendono utilizzare remotamente un componente della libreria, possono effettuare l'ottimizzazione del proprio *workflow* selezionando la modalità di esecuzione più conveniente (*firma del contratto*) e delegando al server il controllo locale (*deployment, enactment, monitoring*) del componente, la cui amministrazione risulta vincolata dalle specifiche imposte dal contratto sottomesso.

Per usufruire delle funzionalità dei componenti amministrati dal server è necessaria la presenza di un controllore dell'applicazione cha ricopra il ruolo di un "**Application Manager**"[2] o di una **Workflow Engine** [3]. Il controllore (nel seguito *Manager*) negozia con l'amministratore la qualità del servizio di elaborazione, precedentemente al suo effettivo utilizzo da parte dell'applicazione client, quindi effettua operazioni accessorie, quali il recupero di informazioni sullo stato di una elaborazione lanciata, o il prolungamento del contratto.

La Figura 1 mostra uno scenario in cui l'esecuzione di alcune attività particolarmente onerose di una applicazione complessa organizzata a Workflow (Workflow-items), è realizzata attraverso chiamate remote a moduli di libreria (jobs), ed il cui controllo è affidato dal Manager al servizio di amministrazione di libreria.

Il servizio espone attraverso porte funzionali Web-Service interfacce sia per la negoziazione del contratto che per l'attivazione della chiamata, che per il monitoraggio dello stato di esecuzione dei job.

L'approccio classico utilizzato da altri sistemi per la sottomissione di job sulla griglia (GT4 WS_GRAM [4], Condor [5]) espone una interfaccia funzionale che permette di sottomettere l'esecuzione di uno o più job opportunamente descritti in uno specifico formalismo (JSDL[6],

RSL[4]). Tale approccio rende laboriosa l'integrazione dei job in applicazioni composte in base all'emergente paradigma SOA, in quanto richiede di volta in volta la traduzione della interfaccia di chiamata nello specifico formalismo e la sua integrazione con dati di controllo. Essa non permette una netta separazione fra il protocollo di negoziazione della Qualità del Servizio (WSLA-like) e il protocollo di richiesta di elaborazione, utile ad attribuire ad attori diversi le operazioni di controllo e di esecuzione.

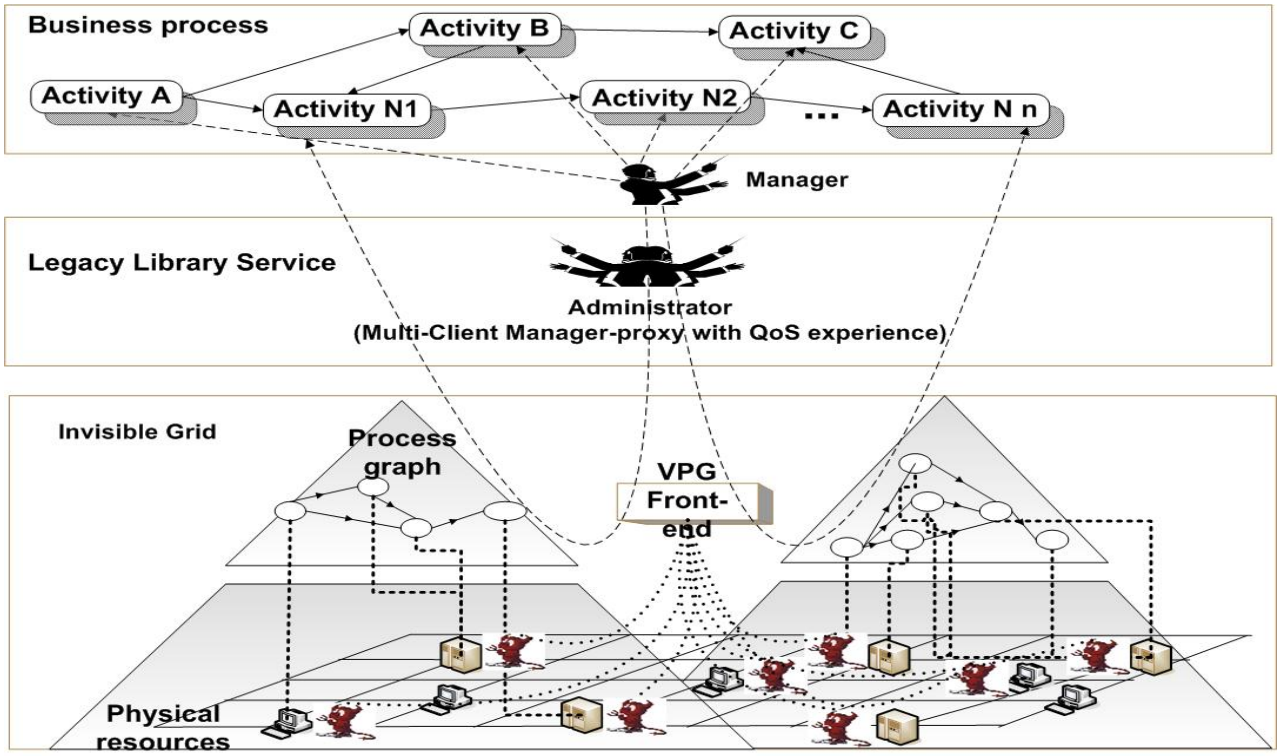

Figura 1. Modello di interazione di una applicazione a work-flow con il Server GRACE

Il sistema proposto permette, invece, di invocare l'esecuzione di un componente (job) attraverso una porta Web Service che espone le precise funzionalità offerte dal componente (mantenendo la segnature dell'interfaccia originaria del modulo elaborativo).

Poiché, però, l'esecuzione di un componente di libreria su griglia richiede una serie di operazioni di controllo del suo ciclo di vita (scheduling, mapping deployment, etc), non è possibile esporre direttamente una porta Web Service del Componente al Workflow esterno. La soluzione adottata per mantenere invariata la "signature", consiste nella dichiarazione una specifica Web-Service (fittizia) per ogni componente e nella introduzione, in ingresso al servizio di libreria, di un servizio front-end che espone le interfacce delle web-services amministrate ed opera de-multiplexer delle chiamate RPC [8]. L'amministratore di componenti è così trasformato in una WS-Resource multifunzionale.

Il demultiplexer distingue ed accoda opportunamente le chiamate sottomesse; la coda viene gestita in base alle politiche modellate in [8]. Una volta che il componente ha completata l'elaborazione, il risultato della stessa viene reso disponibile alle applicazioni client sempre grazie all'interfaccia della Web-service fittizia.

2 Contratti di perfomance

La maggior parte dei sistemi di gestione di risorse su griglia descritti in letteratura sono focalizzati al soddisfacimento delle richieste di performance della singola applicazione e non adottano politiche di ottimizzazione globale sul pool di risorse disponibili. Ad esempio, nella piattaforma GRADS [9], lo scheduler è progettato per minimizzare il tempo di completamento globale di una applicazione time-critical.

Questi sistemi assumono implicitamente che una quantità sufficiente di piattaforma risorse possa sempre essere scoperta ed ottenuta a richiesta dalla griglia. In pratica, non è sempre vero che una applicazione HPC sia necessariamente time-constrained ed altre variabili quali il costo e la affidabilità della elaborazione possono assumere, in certi casi, una importanza preponderante. Ancorpiù, in certe circostanze il pool di risorse cui un utente può efficacemente accedere può essere fortemente limitato. Una differenziazione delle garanzie sul livello di servizio in relazione al suo costo, può risultare utile nella gestione pratica delle risorse di griglia.

A tal fine la piattaforma GRACE espone ognuna i moduli di libreria come WebServices vincolate al rispetto di una gamma di possibili livelli di servizio (contratti SLA) a differente rapporto costo/prestazione. Le tipologie estreme consistono in un profilo real-time (time-constrained) ed in un profilo low-cost (cost constrained).

Nel caso del profilo economico, la riduzione di costo è ottenuta attraverso la esecuzione in backgrouns su risorse non riservate. I processi possono essere sospesi per richiamare risorse a vantaggio di altri processi real-time. Nel caso del profilo real-time, il gestore delle risorse (e lo Scheduler) garantisce, a costi rilevanti, il rispetto dei vincoli di performance espressi nel contratto in termini di numero di macro-operazioni richieste per unità di tempo. Esso alloca e riserva risorse dal pool , effettua un deployment preventivo del codice su un numero di risorse ridondante e, in caso di violazione del contratto recupera risorse da applicazioni che girano con un contratto economico, riducendo il loro parallelismo o anche sospendendo la loro esecuzione.

Lo scenario di negoziazione del contratto inizia con la richiesta dei Web-Services disponibili fino ad istanziare un Web-Service specificandone le modalità contrattuali. Un tipico scenario è il seguente:

- ***Richiesta dell'elenco delle WS–resources disponibili*** – il potenziale cliente, almeno per il momento è solo tale, avendo bisogno di una particolare elaborazione che in qualche modo ritiene di poter individuare fra quelle fornite dal sistema sotto forma di Grid services, si rivolge al sistema stesso per conoscere i dettagli delle Grid services disponibili. Individuata l'eventuale WS–resource che fa al caso suo, il cliente ne esamina i dettagli tecnici per completarne la valutazione. L'interfaccia dell'operazione in esame prevede come risultato fornito in uscita la lista delle WS–resources che, dietro la stipula di un apposito contratto, possono essere messe a disposizione del cliente;
- ***Richiesta del profilo di QoS associato a una WS–resource*** – il potenziale cliente, è ancora solo tale, scelto un Web-Service fra quelli offerti dal sistema, richiede allo stesso i profili contrattuali disponibili per quella WS–resource, per poter scegliere fra questi quello che lo soddisfa maggiormente.
- ***Richiesta di sottomissione del contratto*** – il cliente, che oramai è a tutti gli effetti tale, sottomette al sistema un contratto per la fruizione della WS–resource scelta. Il contratto è compilato in ogni sua parte ed è sottoscritto dal cliente che, quindi, dichiara formalmente di accettarne tutti i punti. Il sistema ricevendo il contratto provvede all'automatica accettazione, previo controllo di correttezza dello stesso. Al contratto viene associato un WS–address che identifica univocamente l'istanza della WS–resource che verrà riservata al cliente. L'interfaccia dell'operazione in esame prevede come parametro di ingresso il contratto; in uscita viene fornito il WS–address attraverso il quale si potrà invocare la WS–resource istanziata;
- ***Terminazione immediata del servizio*** – il cliente può, attraverso questa operazione, interrompere il rapporto che lo lega al sistema. Il che eventualmente comporta la brusca terminazione di ogni possibile elaborazione precedentemente sottomessa al sistema stesso ed ancora in corso di svolgimento. Tale operazione, quindi, fa decadere il contratto ed il sistema ne dealloca la Resource relativa.

Il ciclo di vita delle **Web-Services** è gestito in modo coerente con le specifiche **Web Service Resource Framework (WSRF)** [10] identificando ogni componente come una **ws-resource** (si veda l’appendice C per una definizione dei termini WSRF). La stipula di un contratto determina la creazione di una istanza di tale WS-Resource. Infatti, se analizziamo bene le specifiche ci accorgiamo di una corrispondenza tra le operazioni previste per la gestione del ciclo di vita ws-resource e il protocollo di negoziazione del contratto:

- ResourceFactory. Crea una istanza di WS-Resource e restituisce l'indirizzo della sua porta funzionale della Grid Service istanziata (WS-Resource EndPoint). Durante l'operazione di creazione devono essere specificati un tempo entro il quale il Web -Service dve essere distrutto se non termina la propria esecuzione (questa operazione coincide con la stipula del contratto);

- Destroy. Distrugge un componente precedentemente creato (questa operazione coincide con la revoca di un contratto).

- SetTerminationtime. Imposta il tempo di terminazione di un componente. Questo comando permette di attuare uno "Scheduled destruction", cioè la distruzione del componente entro un termine stabilito (ciò è stabilito al momento della stipula del contratto).

3 Architettura del servizio

Da un punto di vista tecnico, il servizio deve permettere la negoziazione e l'attivazione (on-demand) di funzionalità di componenti all'interno di un Workflow esterno attraverso l'impiego di tecnologie pienamente compatibili con gli standard WSRF. Partendo da questi presupposti appare chiaro che la tecnologia con cui deve essere realizzato il sistema di interfaccia è quella dei Web services. Inoltre, visto che le fasi che caratterizzano la fruizione di un ben determinato servizio da parte del cliente sono essenzialmente due (negoziazione del contratto ed invocazione della funzionalità del componente), risulta naturale associare a ciascuna di esse due distinte Web services. Nella fattispecie, una Web-Service unica per le operazioni che consentono di ottenere la stipula del contratto (WS-Notary) e una Web-Service specifica per ogni componente (fittizia) che ne espone le funzionalità. E' importante specificare che, a causa dell'overhead introdotto dall'intera catena delle chiamate interne, il Server espone delle Web-Services le cui funzionalità richiedono un certo tempo di elaborazione, altrimenti non si avrebbe la convienenza di utilizzare il sistema. La figura 2 mostra, con dei blocchi con bordo tratteggiato, le parti principali dell'intero sistema.

Il **Component Administrator** costituisce la parte centrale del sistema; esso si occupa di negoziare i contratti con i Manager delle applicazioni client e di attuare le politiche di ottimizzazione delle risorse di griglia. Esso possiede due interfacce funzionali:

- **WS-Notary**: Web-Service che espone i metodi per la negoziazioe del contratto;
- **WS-WorkItemList-Handler**: Web-Service per l'accodamento delle chiamate. Come si vede dalla figura questa interfaccia è nascosta dal demultiplexer che, abilita le applicazioni client ad invocare le funzionalità del componente anziché le funzionalità del gestore della coda.

Sono inolte compiti del Component Administrator la gestione l'intero ciclo di vita dei componenti (in funzione del contratto associato all'esecuzione) e l'attuazione delle dirttive di enforcement emanate dal JobInspector. Come si vedrà più avanti questo componente è stato realizzato grazie all'impiego della teconogia ASSIST 1.3. I moduli principali sono: Notary (gestore dei contratti di performance), WorkItemListHandler (gestore delle chiamate), Scheduler (gestore delle code di priorità e mapper delle istanze di componenti di libreria sulle risorse), WorkItemHandler (gestore dei istanze di esecuzione dei componenti),

Un altro componente che ricopre un ruolo importante all'interno del sistema è il **Demultiplexer** (il blocco denominato DeMlx). Esso ha il compito di intercettare le chiamate alle Web-Service fittizie e di sottometterle nella coda del Component Administrator. Al fine di poter essere allineati e perfettamente integrabili con il framework Globus Toolkit 4, per l'implementazione di questo componente sono state scelte delle tecnologie java servlet di Java. Quindi è stato scelto Apache Tomcat come servlet container, della versione servlet di Axis.

Il **JobInspector** [8] è il componente che si occupa di monitorare e controllare gli adempimenti contrattuali firmati col Manager delle applicazioni client. Il suo compito è quello monitorare sia lo stato delle risorse che l'esecuzione del componente (attraverso dei sensori event-based) e di emanare delle direttive di enforcement (sempre attraverso eventi) nel caso di violazioni contrattuali. Tali direttive vengono tradotte, dal Component Adminitrator (modulo WorkItemHandler), in riconfigurazioni del componente (variazioni del grafo fisico).

Il **supporto runtime** (la scatola VPG Front-end in figura 2) possiede un bus ad eventi che ha permesso di realizzare sia i sensori necessari per il monitoraggio (stato delle risore di griglia, avanzamento dell'esecuzione del componente) che gli attuatori per il controllo dell'esecuzione (configurazione dei componenti).

Il **Grid-Resource Manager** è il componente che si occupa di stabilire sia le risorse hardware da utilizzare (nodi della griglia) che i componenti da rendere disponibili in un dato momento. Questo componente è ancora in fase di sviluppo; attualmete esso permette di configurare la piattaforma a partire da un un pool statico di risorse.

Nei paragrafi successivi sono descritti il Demultiplexer e il Component Administrator. Il componente JobInspector, il supporto runtime e la spiegazione dettagliata dell'impiego degli eventi sono descritti in [8].

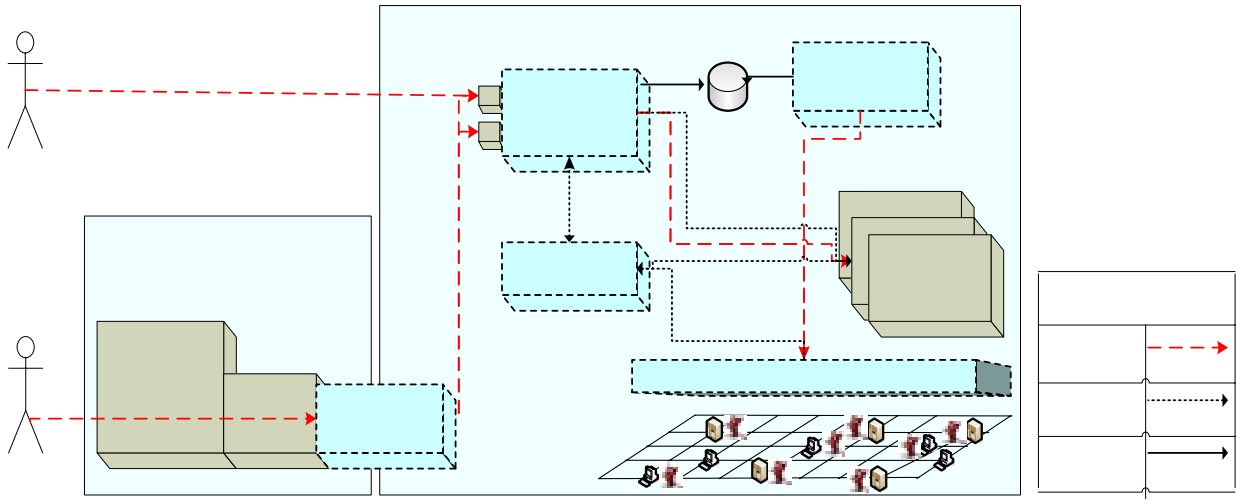

Figura 2. Struttura complessiva del Server

4 Demultiplexer

Si è accennato al fatto che le Web-Services esposte dal Server sono fittizie in quanto le chiamate ad esse, prima di essere eseguite, devono essere gestire dal server al fine di attuare le proprie politiche di scheduling e ottimizzazione globale delle risorse nel rispetto delle specifiche contrattuali. Per far ciò è necessaria la presenza di un demultiplexer in grado di intercettare le chiamate alle Web-Services e sottomettere tali chiamate nella coda dei WorkItems gestita dal Component Administrator. Data la presumibile durata delle elaborazioni attivate dalle chiamate alle Web-Services, si è pensato di sdoppiare ogni metodo esposto nell'interfaccia del componente in due metodi distinti nell'interfaccia della Web-Service fittizia esposta dal server. Il primo metodo viene utilizzato per invocare la funzionalità del componente, il secondo per avere il risultato della funzionalità invocata. Per fare un esempio, se un componente espone un metodo "*float Sum(float a, float b*) allor ala Web-Services fittizia esposta dal server avrà i seguenti due metodi:

int SubmitSum(float a , float b): sottomette la chiamata e ritorna un codice che assicura che la chiamata era valida (si deve verificare la validità del contratto associato alla chiamata);

float GetSum(int code): ritorna il risultato dell'elaborazione, è necessario trasmettere il codice ottenuto durante la chiamata precedente;

La gestire delle chiamate alle Web-Services fittizie ha portato alla messa a punto di una architettura software relativamente sofisticata,. Il componente software realizzato (***Multiplexer)*** è in grado di dare al Workflow l'apparenza di aver a che fare con una Web-Service specializzato e dedicata, ma che in realtà utilizza lo stesso codice per gestire qualsivoglia chiamata relativa a operation di qualsivoglia Web service. In sintesi il demultiplexer convoglia tutte le SOAP envelopes correlate a queste ultime ad un unico modulo software, il quale dopo una serie di microelaborazioni,

che per lo più astraggono dalla particolare Web service target, inoltra lo stesso pacchetto SOAP al successivo componente del sistema determinato dalla sequenza logica dell'ideale processo. In un ulteriore stadio del processo, un determinato componente del sistema si occupa di fare una speculare operazione di multiplexing, sottoponendo ad un opportuno componente finale il job descritto nella SOAP envelope. Una volta che il componente finale restituisce il risultato dell'elaborazione effettuata, questo viene memorizzato insieme alle informazioni sul destinatario di tale informazione, cioè l'applicazione client, in un registro condiviso. A tale registro ha accesso il ***Multiplexer***, il quale, su richiesta del client, è in grado di restituire indietro il risultato dell'elaborazione precedentemente sottomessa dal client stesso.

L'implementazione del ***Multiplexer*** con le API di Axis è nel prosieguo descritta. Essa ha dato luogo alla scrittura di cinque classi, ognuna delle quali implementa l'interfaccia org.apache.axis.Handler della libreria di Axis. Uno di questi Handler è più importante rispetto agli altri in quanto costituisce il modulo software a cui per ultimo l'engine di Axis consegna il pacchetto. Si tratta cioè dell'ultimo elemento della pipeline a cui viene sottoposta una SOAP envelope pervenuta al server su cui è installato Axis. In figura 3 è illustrato della distribuzione dei vari handler del demultiplexer nella pipeline di processo di Axis.

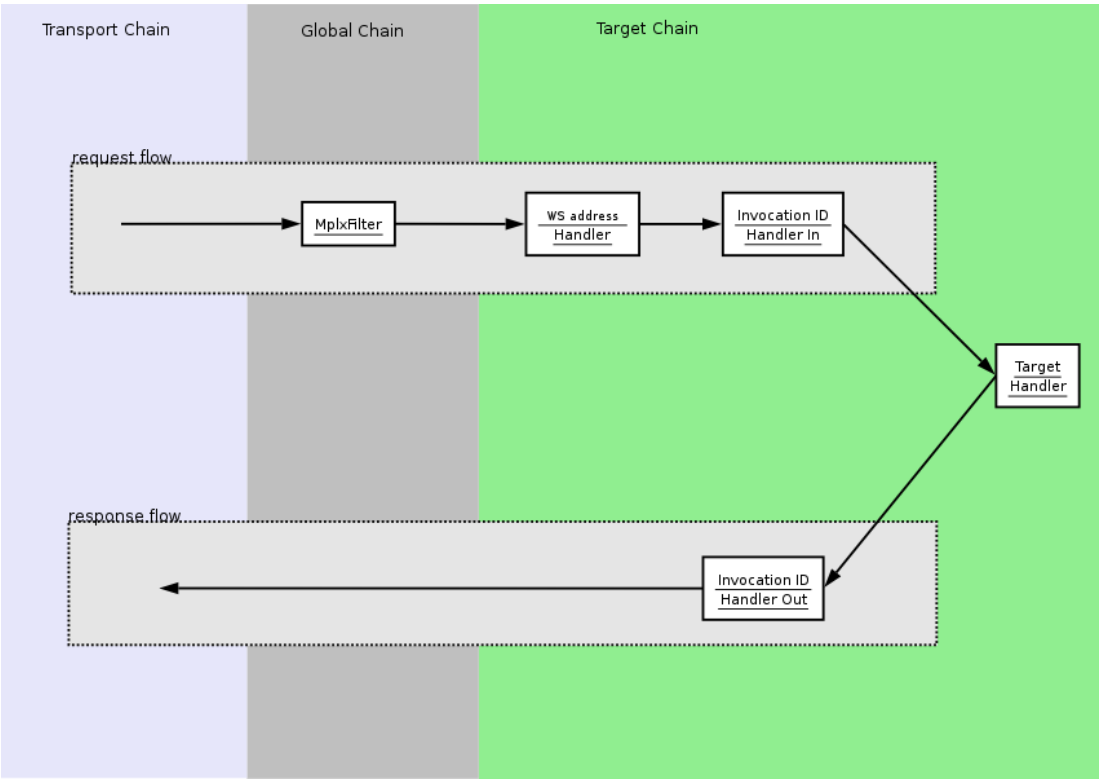

Figura 3. schema della distribuzione dei vari handler del demultiplexer nella pipeline di processo di Axis

L'uso di un Handler in questo modo è possibile grazie ad un particolare configurazione del file di deployment. Gli altri quattro Handler sono, per così dire, gli handler dell'Handler prinicipale, svolgono quindi compiti ausiliari. E' da sottolineare che uno per uno di essi il deploy deve essere fatto nella Global Chain di Axis. Si forniscono di seguito ulteriori spiegazioni:

- ***Handler principale***: modulo target;
- ***MplxFilter***: questo Handler è installato nella Global Chain nel request flow, in quanto ha il compito di filtrare le buste SOAP destinate alle ***Specialized Web services*** e reindirizzarle al TargetHandler;
- ***WS address Handler***: questo Handler è installato nella Target Chain nel request flow, si occupa di gestire il WS-addressing secondo le relative specifiche[WS-ADDRESS];
- ***Invocation ID Handler IN***: questo Handler è installato nella Target Chain nel request flow;
- ***Invocation ID Handler OUT***: questo Handler è installato nella Target Chain nel response flow.

5 Il Component Administrator

La parte centrale del sistema è costituita da una applicazione distribuita il cui scopo è quello di gestire l'intero ciclo di vita dei componenti software generati dalle chiamate alle Web -Service fittizie sopra descritte. Il server deve inoltre garantire che tali componenti vengano amministrati nel rispetto delle direttive contrattuali stabilite. Tale scopo viene raggiunto attraverso diverse politiche:

- scheduling delle chiamate alle Web-Services in base allo stato delle risorse disponibili ed al profilo di contratto asociato alla chiamata;
- selezione e riserva delle risorse in base al profilo contrattuale;
- configurazione del componente in base al profilo contrattuale;
- direttive di enforcement per variare il carico di lavoro o il maping fisico del grafo dei processi del componente;
- qualora previsto nel contratto e grazie alla presenza dell'Inspector [8], controllo dello stato di avanzamento dell'esecuzione (opzione speed constraint) e controllo dello stato delle risorse riservate al componente (opzione Resource Reservation e Power Constraint) con relative politiche di enforcement realizate all'interno dell'Inspector;

La tecnologia utilizza nell'implementazione dell'applicazione è ASSIST versione 1.3, essa ha permesso di distribuire i moduli su procesi differenti e di realizzare un gestore della coda concorrente in modo da poter gestire esecuzioni di più componenti contemporaneamente. In figura 4 è schematizzata la struttura dell'applicazione. Come si vede dalla figura l'applicazione presenta due porte provide SOAP che permettono rispettivamente all'Application Manager di negoziare i contratti e al demultiplexer di accodare le chiamate alle Web-services fittizie. L'applicazione comunica con la Virtual Private Grid [10] attarverso il protocollo SOAP per gestire il ciclo di vita dei componenti e per monitorare lo stato delle risorse. Inoltre, un protocollo di interazione con il JobInspector, basato su eventi, permette di realizzare le politiche di enforcement al fine di poter garantire le opzioni contrattuali negoziate. ASSIST ha inoltre permesso la realizzazione di una memoria condivisa (Reference) tra i vari moduli dell'applicazione. In seguito sono descritti i singoli moduli e la struttura della memoria distribuita.

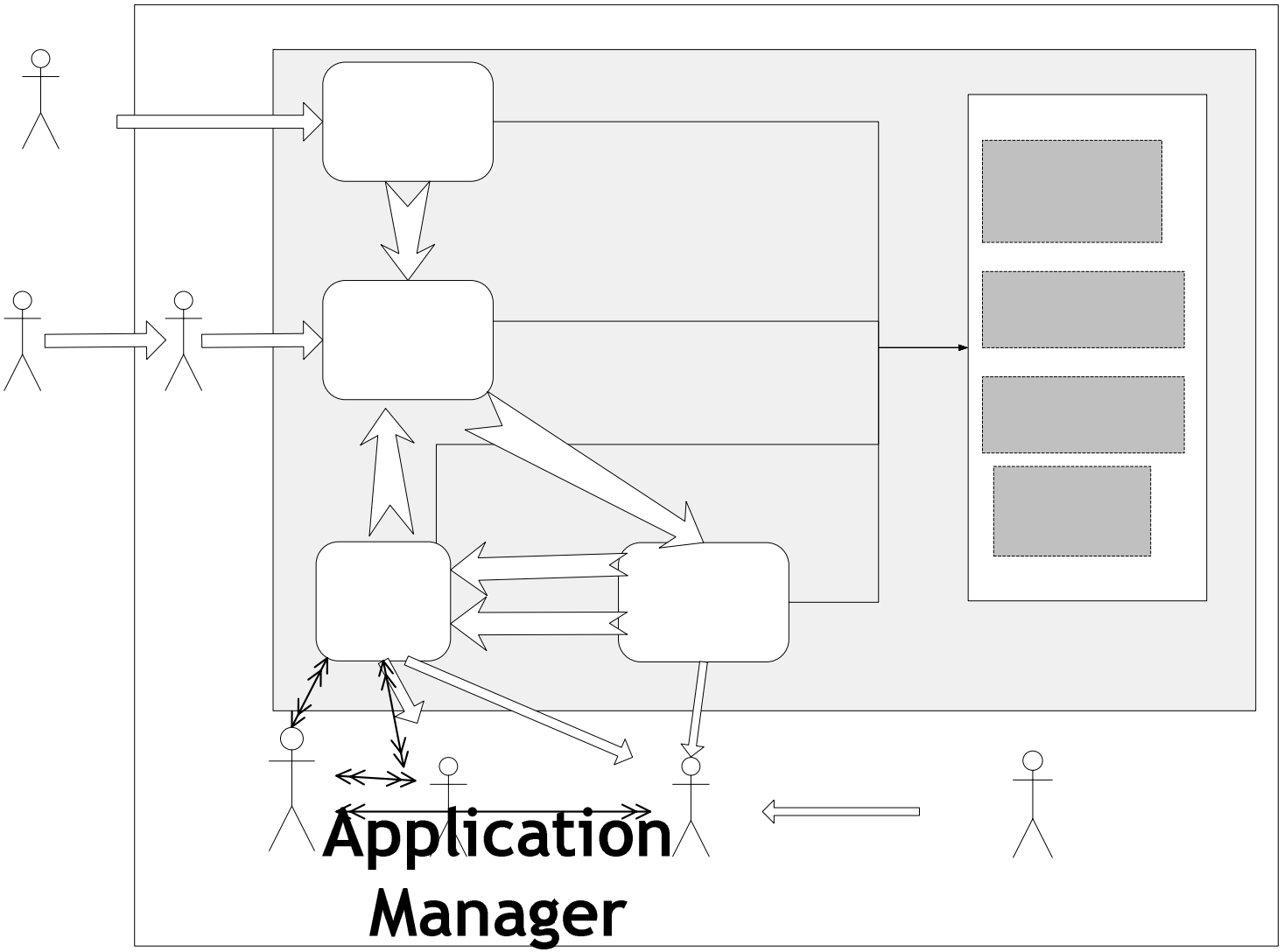

Figura 4. Moduli dell'applicazione Component-Administrator

5.1 Notary

Questo modulo ha il compito di negoziare con il controllore delle applicazioni client (Manager) l'utilizzo dei componenti ospitati sulla piattaforma. Come già detto ogni componente viene visto all'esterno come un *Web-Service* per cui l'operazione di stipula del contratto coincide con l'operazione di factory della Web-Service. Il protocollo di negoziazione del contratto è sostanzialmente è formato dalle seguenti fasi:

1. Il Manager chiede i profili contrattuali con cui un servizio è usufruibile;
2. Il Manager seleziona il profilo contrattuale e compila il contratto specificando i valori delle opzioni contrattuali;
3. Il Manager sottomette il contratto al Notaio che ne verifica la validità;
4. Nel caso in cui il contratto sia valido il Notaio restituisce un identificatore della Web-Service istanziata da utilizzare nelle chiamate alla Web-service (WS-Address);

Le funzionalità principali di questo modulo dunque sono:

- Fornire l'elenco dei Web-services disponibili, per ogni Web-service viene fornita la descrizione della Web-Service;
- Fornire i profili contrattuali con cui ogni Web-service puo' essere attivato. Da notare che ogni profilo contrattuale determina la modalità di esecuzione del componente che implementa la funzionalità del servizio esposto e che ad ogni profilo contrattuale sono associati un insieme di caratteristiche qualitatitative (opzioni del contratto) da determinare al momento della sottomissione del contratto.
- Negoziazione del contratto. I Manager delle applicazioni client devono selezionare il profilo contrattuale tra quelli disponibili e specificare i valori delle opzioni contrattuali. Una volta sottomesso, il contratto viene analizzato dal notaio per verificarne la validità, infatti viene verificata:
    - la compatibilità del profilo contrattuale selezionato per il servizio che si intende utilizzare;
    - la compatibilità delle opzioni contrattuali con il profilo contratto selezionato;
    - validità della opzione "PowerConstraint" (se presente) : tale vincolo non puo' essere applicato a nodi fisici del grafo astratto del componente e a nodi astratti in cui è specificato un vincolo di "Resource Reservation";

- validità della opzione “ResourceReseration” (se presente): tale vincolo non può essere applicato a nodi astratti del componente in cui è specificato un vincolo di “PowerConstraint”. Inolte si deve verificare se i nodi specificati sono montati sulla VPG.

- Fornire lo stato di un contratto sottomesso (numero di chiamate in corso al Web-service, scadenza del contratto e i valori di tutte le opzioni contrattuali sottomesse) ;
- Metodo per poter chiudere un contratto. La chiusura del contratto implica l’eliminazione di tutte le chiamate alla Web-service non ancora in esecuzione, le chiamate in esecuzione invece, verranno completate.

5.2 WorkItemList Handler

Questo modulo ha il compito di aggiornare la coda delle chiamate alle Web-services (WorkItems). Come già accennato, il server espone un insieme di Web-Services fittizie le cui chiamate vengono intercettate da un demultiplexer. Tutte le chiamate intercettate vengono messe in una coda in attesa di essere schedulate per essere eseguite, ciò viene realizzato attraverso una porta funzionale Web-Serivice con cui il demultiplexer può:

- sottomettere in coda una chiamata: da sottolineare che durante questa operazione viene verificata l’esistenza di un contratto valido associato alla chiamata;
- ottenere lo stato di una chiamata;
- ottenere la busta SOAP di ritorno (nel caso in cui l’elaborazione è finita);

Come si evince dall’interfaccia funzionale, questo modulo ha anche lo scopo di tenere traccia tutte le informazioni legate al WorkItem generato dalla chiamata (si rimanda al paragrafo 4.5 per un maggione dettaglio). Lo stato di ogni chiamata viene aggiornato costantemente e al suo completamento viene memorizzata la busta SOAP di risposta.

Il modulo è un parmode con tipologia none, ciò perché è necessaria la presenza di più worker, uno di essi è dedicato all’ascolto dei comandi della porta Web Service e gli altri sono attivati dal completamento dei WorkItem e hanno lo scopo di aggiornarne lo stato.

5.3 Scheduler

Lo scheduler ha il compito di analizzare la lista delle chiamate per selezionare il prossimo WorkItem da eseguire. Tale scelta viene eseguita in base a due fattori findamentali:

- **Priorità**: ogni WorkItem possiede una priorità di scheduling. Tale priorià e uguale per tutti al momento del loro accodamento e aumenta ogni volta che un dato WorkItem viene rischedulato in seguito a degli errori che non hanno permesso il completamento dell'elaborazione da parte del componente (per quelle sitauzioni in cui il JobInspector decide di rischedulare il WorkItem).
- **Tempo di accodamento**: a parità di priorità viene applicata la potilica FIFO: First In First Out, per cui avranno la precedenza i WorkItems che sono in coda da più tempo.

Purtroppo la scelta del WorkItem da schedulare viene effettuata ignorando lo stato delle risorse disponibili. Ciò implica che, una volta schedulato un WorkItem, se non esistono le risorse disponibili per eseguire il componente, il WorkItem selezionato si troverà in attesa che si renderanno disponibili le risorse necessarie per soddifare il contratto. Il risultato è che lo scheduling viene bloccato.

I Componenti gestiti sono di natura parallela, ciò significa che al momento dell'esecuzione si deve stabilire il grafo fisico dei processi da lanciare [11]. Compito di questo modulo è anche quello di determinare tale grafo fisico e di stabilire il mapping sulle risorse. La costruzione del grafo fisico avviene in base ai seguenti fattori:

- **Profilo di contratto**: le risorse disponibili vengono filtrate in base al profilo contrattuale del WorkItem da schedulare. Infatti esistono dei profili contrattuali che prevedono l'uso condiviso di risorse (ECONOMY, LOW_COSTI) e profili che invece impongono l'uso esclusivo delle risorse co-reservate.(STANDARD , BUSINESS, BEST_EFFORT).
- **Vincoli sul grafo astratto** del componente: alcuni componenti potrebbero avere dei vincoli di mapping, si pensi ad esempio ad un processo del componente che deve acquisire dei dati da un dispositivo elettronico presente in una macchina particolare.
- Opzione contrattuale "**ResourceReservation**": questa opzioni impone dei vincoli sulla risorsa fisica che si intende utilizzare per un deternato nodo del grafo del componente. Per fare un esempio questa opzione potrebbe essere utilizzata se non si vogliono distribuire dei dati di elaborazione da ritenere sensibili per la griglia ma si volgiono confinare all'interno di macchine considerate di fiducia.
- Opzione contrattuale "**PowerConstraint**": questa opzione vincola la potenza effettiva delle macchine da utilizzare per i processi del grafo del componente.

Al fine di poter adempiere ai propri compiti il modulo necessita della conoscenza dello stato corrente delle risorse montate e immediatamente disponibili sulla griglia, ciò viene fatto grazie all'interfaccia della macchina virtuale di griglia (VPG [8]).

5.4 WorkItemsHandler

Questo modulo ha il duplice compito di:

- gestire l'intero ciclo di vita del componente che dovrà eseguire il WorkItem schedulato;
- attuare le politiche di enforcement dettate dall'Inspector.

Al fine di poter gestire in maniera concorrente più WorkItem il modulo è stato realizzato con un parmode con tipologia "none". Ad ogni worker viene assegnato dunque il compito di gestire un workitem schedulato. Tale gestione comporta l'attuazione di una serie di fasi che iniziano con la configurazione e l'attivazione del componente fino ad attendere il completamento dell'esecuzione in modo da poter effettuare le operazioni di post-completamento del workitem. Se non ci sono Worker disponibili lo scheduler interrompe il proprio operato in attesa che si rendano disponibili dei workers.

Uno dei worker viene impegnato per l'ascolto degli eventi con cui il JobInspector emana le direttive di enforcement per modificare la situazione corrente. Compito di questo worker è dunque quello di fare da listener di tali eventi e attuare le direttive dettate dall'Inspector.

5.5 Strutture dati

Durante la realizzazione dell'applicazione si è dovuti ricorrere all'utilizzo della memoria "Reference" per poter disporre di una zona di memoria distribuita (e unica) tra i vari processi. In figura 5 è riassunta la strutura logica di tale memoria. Come si evince dalla figura in memoria sono memorizzate quattro tabelle, ognuna di tale tabella è stata realizzata attraverso un vettore di strutture dati, in seguito sono riassunte le caratteristiche principali di ognuna di tali tabelle.

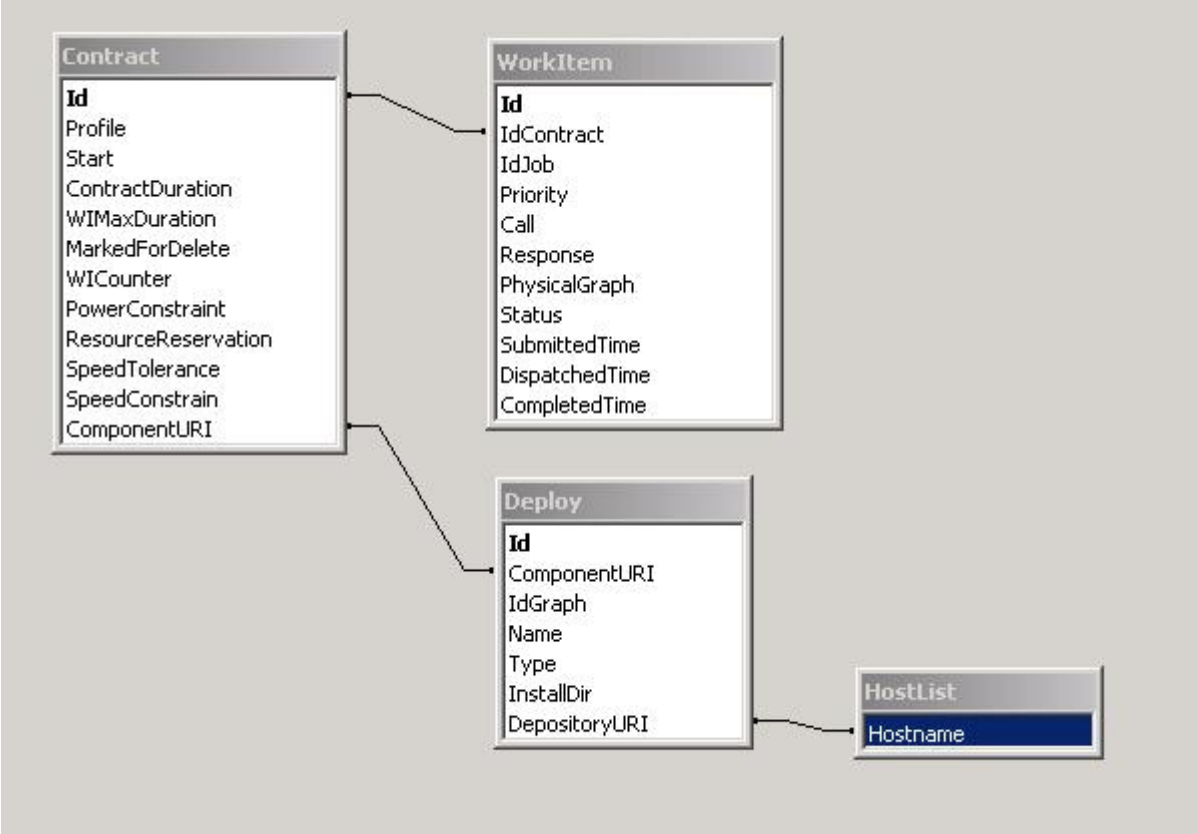

Figura 5. Struttura della memoria distribuita dell'applicazione

Contract. Tabella in cui vengono memorizzati i contratti concordati tra il modulo "Notary" e le applicazioni clienti. Attributi della tabella sono:

- **Id.** Identificatore del contratto, esso corrisponde all'identificatore dell'instanza della Web-Service oggetto del contratto;
- **Profile**. Profilo del contratto;
- **Start.** Data di sottomissione del contratto;
- **WICounter**. Numero di chiamate alla Web-Service in corso;
- **ContractDuration.** Tempo oltre la quale il contratto si considera non più valido. Necessario per realizzare il meccanismo di *"Scheduled Desctruction"* [wsrf];
- **MarkedForDelete**. Flag che indica che il contratto è scaduto;
- **PowerConstraint, ResourceResevation, SpeedConstraint.** Memorizzano i valori delle rispettive opzioni contrattuali;
- **ComponentURI**. URI del componente che realizza la funzionalità del Web-Service oggetto del contratto.

WorkItem. Tabella in cui vengono memorizzati i WorkItem generati dalle chiamate alle Web-Service esposte dal server. Attributi del WotkItem sono:

- **Id**. Identificatore assegnato al WorkItem;
- **IdContract**. Identificarore del contratto;
- **IdJob**. Identificatore del grafo dei processi lanciato sui nodi della griglia;
- **Priority**. Priorità del WorkItem;
- **Call**. Busta SOAP di chiamata al Web-Service fittizio;
- **Response**. Busta SOAP di risposta del Web-Service fittizio;
- **PhysicalGraph**. Grafo fisico del componente lanciato;
- **SubmittedTime**, **DispatchedTime**, **CompletedTime**. Date di accodamento, dispatching e completamento del WorkItem;
- **Status**. Stato corrente del WorkItem (possibili stati sono SUBMITTED, DISPATCHED, INITED, RUNNING, ACTIVE, FAILED, DONE, SUSPENDED, NOT_VALID, SYSTEM_ERROR);

Deploy. Tabella in cui vengono memorizzate le informazioni di deployment dei file necessari per l'esecuzione dei componenti (eseguibili, librerie, files, etc). Ogni componente genera un un pool di files distributo lungo le risorse di griglia, tale pool viene aumentato incrementalmente ogni volta che un si determina il mapping di un componente su risorse non ancora dallo esso utilizzate. Gli attributi della tabella sono:

- Identificatore del file;
- Identificatore del pool di file distribuito sulla griglia;
- Identificatore del componente a cui il pool si riferisce;
- Tipo di file (EXE, DLL, GENERIC)
- Directory di installazione;
- Locazione del file (necessario per effettuare le operazioni di FTP sui nodi di griglia);
- Lista di nodi su cui il file è attualmente presente;

5.6 Note sull'utilizzo di ASSIST 1.3

La tecnologia ASSIST è stata utilizzata per il coordinamento di moduli applicativi realizzati in conformità con il processo di integrazione [12]. Data la natura di attivazione RPC di tali moduli si è reso necessario l'impiego di un meccanismo in grado di realizzare un protocollo RPC attraverso gli Stream Data-Flow di ASSIST. Questi ultimi permettono la comunicazione remota in un solo verso, per cui possono essere utilizzate per realizzare chiamate RPC di tipo "one-way", cioè per funzioni che non possiedono un valore di ritorno e che non abbiano parametri di chianata passati per riferimento (come ad esempio i parametri "ref" delle Web Service). L'utilizzo di due stream separati, uno per la chiamata e uno per la risposta nel verso contrario non è attuabile poiché lo stream di ritorno attiverebbe una nuova "proc" diversa dalla "proc" chiamante, ed addirittura su su un altro thread/processo, rendendo inconsistente il protocollo tra il modulo chiamante e quello servente.

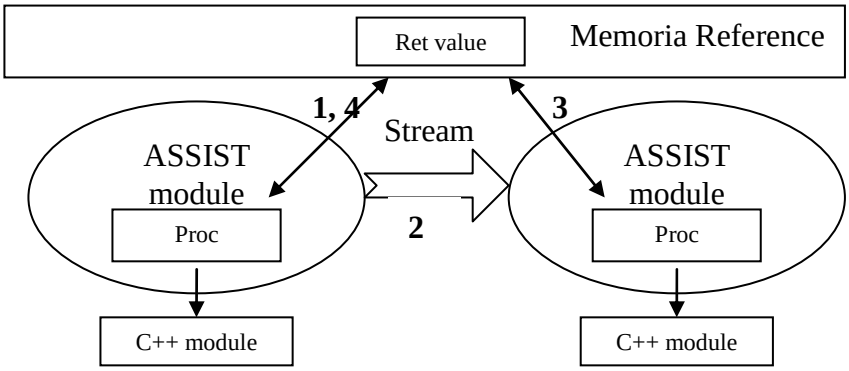

Figura 6. Integrazione di moduli in ASSIST

In figura 5 è illustrata la soluzione adottata. In pratica la chiamata viene attivata da uno strema, mentre il valore di ritorno viene restiuito grazie alla memoria reference. Lo Stream dunque viene utilizzato come una "busta" contenete i parametri di chiamata ed il riferimento in memoria dove il modulo servente deve inserire i parametri di ritorno. Informazioni necessarie da inserire nello stream sono:

- identificatore asseganto alla chiamata
- riferimento alla memoria reference ove inserire il risulato (di tipo ref_t);
- dimensione della memoria reference ove il risultato della chiamata sarà inserito;
- Paramentri della chiamata;

Data la natura asincrona del modello DataFlow viene utilizzato un flag che indica al modulo cliente la fine dell'elaborazione del modulo servente al fine di garantire la consistenza della chiamata. In tal modo il modulo chiamante, prima di effettuare la chiamata crea il riferimento in memoria per la risposta e/o il flag di terminazione (1) e non appena inviato lo stream per sottomettere la chiamata (2), si mette in attesa della terminazione dell'esecuzione remota monitorando sul cambiamentto di stato di tale flag (4). Il modulo servente dunque

- legge dallo stream di ingresso gli argomenti della chiamata(2);
- effettua la chiamata alla proc che chiama a sua volta il modulo applicativo;
- inserire il risultato (o il flag sull'esito dell'operazione) nel riferimento alla memoria(3).

Ovviamente se il modulo applicativo possiede diverse funzionalità il modulo ASSIST conterrà uno stream di ingresso per ogni funzionalità da esporre sfruttando il non-determinismo sugli stream.

Durante la realizzazione dell'applicazione ci si è trovati davanti ad una serie di limitazioni insiti nell'utilizzo della versione1.3 di ASSIST , attualmente in beta test.

- Insuffufficiente supporto del tool assist-WS (V1.2rc3 e V.1.3), alla costruzione di porte provide verso l'esterno.. Abbiamo creato tramite codice utente all'interno delle proc la porta provide SOAP. Ogni modulo ASSIST che contiene all'interno una tale porta deve avere una proc dedicata che faccia da "listener" dei comandi.
- I dati contenuti negli stream e le variabili shared hanno lunghezza fissa.
- Non è possibile accedere in maniera concorrente da parte dei processori virtuali di uno steso parmod ad uno stesso attributo del parmod,. Gli attributi infatti possono essere partizionati (topologia array) o replicati (topologia array e none) tra i processori virtuali ma non condivisi. E' necessario l'utilizzo di variabili globali di tipo reference, ma la gestione della sincronizzazione degli accessi è a carico del codice utente. La gestione di tale tipo di variabili comporta però dei malfuzinamenti nell'applicazine (vedi punto successivo);
- Ci sono delle difficoltà da parte del compilatore nell'utilizzo delle variabili "shared", si verificano dei casi in cui le proc che accedono a variabili shared non completano la propria esecuzioe e sembrano andate in deadlock;
- Le variabili shared possono essere solo ti tipo ARRAY

6 Appendice A:

Descrizione delle porte funzionali Component Administrator

6.1 Notary

```
<?xml version="1.0" encoding="UTF-8"?>
<WSDL:definitions xmlns:tns="GRACENotary.xsd" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:Notary="GRACENotary.xsd"
xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="GRACENotary.xsd" name="GRACENotary">
        <WSDL:types>
                <schema targetNamespace="GRACENotary.xsd" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:Notary="GRACENotary.xsd"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
                        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
                        <!-- operation request element -->
                        <element name="GetComponentList">
                                <complexType>
                                        <sequence>
                                                <element name="userName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                                <element name="pwd" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <!-- operation response element -->
                        <element name="GetComponentListResponse">
                                <complexType>
                                        <sequence>
                                                <element name="result1" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <!-- operation request element -->
                        <element name="GetContractProfiles">
                                <complexType>
                                        <sequence>
                                                <element name="descriptionURI" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                                <element name="userName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                                <element name="pwd" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <!-- operation response element -->
                        <element name="GetContractProfilesResponse">
                                <complexType>
                                        <sequence>
                                                <element name="result2" type="xsd:int" minOccurs="1"
maxOccurs="unbounded"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <!-- operation request element -->
                        <element name="SubmittContract">
                                <complexType>
                                        <sequence>
```

```
                                        <element name="userName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                        <element name="pwd" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                        <element name="contract" type="Notary:ContractRecord"
minOccurs="1" maxOccurs="1"/>
                                </sequence>
                        </complexType>
                </element>
                <!-- operation response element -->
                <element name="SubmittContractResponse">
                        <complexType>
                                <sequence>
                                        <element name="result3" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                </sequence>
                        </complexType>
                </element>
                <!-- operation request element -->
                <element name="DestroyContract">
                        <complexType>
                                <sequence>
                                        <element name="userName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                        <element name="pwd" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                        <element name="contractId" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                </sequence>
                        </complexType>
                </element>
                <!-- operation response element -->
                <element name="DestroyContractResponse">
                        <complexType>
                                <sequence>
                                        <element name="result4" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                                </sequence>
                        </complexType>
                </element>
                <!-- operation request element -->
                <element name="GetContractInfo">
                        <complexType>
                                <sequence>
                                        <element name="contractId" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                                        <element name="userName" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                        <element name="pwd" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                </sequence>
                        </complexType>
                </element>
                <!-- operation response element -->
                <complexType name="ContractRecord">
                        <sequence>
                                <element name="Profile" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                                <element name="ContractDuration" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                                <element name="WIMaxDuration" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
                                <element name="PowerConstraint" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                <element name="SpeedConstraint" type="xsd:float" minOccurs="1"
maxOccurs="1"/>
                                <element name="SpeedTolerance" type="xsd:float" minOccurs="1"
maxOccurs="1"/>
                                <element name="ComponentURI" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
                                <element name="ResourceReservation" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                        </sequence>
                </complexType>
                <element name="ContractRecord" type="Notary:ContractRecord"/>
        </schema>
    </WSDL:types>
    <message name="GetComponentListRequest">
```

```
		<part name="parameters" element="Notary:GetComponentList"/>
	</message>
	<message name="GetComponentListResponse">
		<part name="parameters" element="Notary:GetComponentListResponse"/>
	</message>
	<message name="GetContractProfilesRequest">
		<part name="parameters" element="Notary:GetContractProfiles"/>
	</message>
	<message name="GetContractProfilesResponse">
		<part name="parameters" element="Notary:GetContractProfilesResponse"/>
	</message>
	<message name="SubmittContractRequest">
		<part name="parameters" element="Notary:SubmittContract"/>
	</message>
	<message name="SubmittContractResponse">
		<part name="parameters" element="Notary:SubmittContractResponse"/>
	</message>
	<message name="DestroyContractRequest">
		<part name="parameters" element="Notary:DestroyContract"/>
	</message>
	<message name="DestroyContractResponse">
		<part name="parameters" element="Notary:DestroyContractResponse"/>
	</message>
	<message name="GetContractInfo">
		<part name="parameters" element="Notary:GetContractInfo"/>
	</message>
	<message name="ContractRecord">
		<part name="parameters" element="Notary:ContractRecord"/>
	</message>
	<portType name="rpc">
		<operation name="GetComponentList">
			<documentation>Service definition of function
Notary__GetComponentList</documentation>
			<input message="tns:GetComponentListRequest"/>
			<output message="tns:GetComponentListResponse"/>
		</operation>
		<operation name="GetContractProfiles">
			<documentation>Service definition of function
Notary__GetContractProfiles</documentation>
			<input message="tns:GetContractProfilesRequest"/>
			<output message="tns:GetContractProfilesResponse"/>
		</operation>
		<operation name="SubmittContract">
			<documentation>Service definition of function
Notary__SubmittContract</documentation>
			<input message="tns:SubmittContractRequest"/>
			<output message="tns:SubmittContractResponse"/>
		</operation>
		<operation name="DestroyContract">
			<documentation>Service definition of function
Notary__DestroyContract</documentation>
			<input message="tns:DestroyContractRequest"/>
			<output message="tns:DestroyContractResponse"/>
		</operation>
		<operation name="GetContractInfo">
			<documentation>Service definition of function
Notary__GetContractInfo</documentation>
			<input message="tns:GetContractInfo"/>
			<output message="tns:ContractRecord"/>
		</operation>
	</portType>
	<binding name="GRACENotary" type="tns:rpc">
		<SOAP:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
		<operation name="GetComponentList">
			<SOAP:operation/>
			<input>
				<SOAP:body use="literal"/>
			</input>
			<output>
				<SOAP:body use="literal"/>
			</output>
		</operation>
		<operation name="GetContractProfiles">
			<SOAP:operation/>
			<input>
				<SOAP:body use="literal"/>
			</input>
			<output>
```

```
                        <SOAP:body use="literal"/>
                    </output>
                </operation>
                <operation name="SubmittContract">
                    <SOAP:operation/>
                    <input>
                        <SOAP:body use="literal"/>
                    </input>
                    <output>
                        <SOAP:body use="literal"/>
                    </output>
                </operation>
                <operation name="DestroyContract">
                    <SOAP:operation/>
                    <input>
                        <SOAP:body use="literal"/>
                    </input>
                    <output>
                        <SOAP:body use="literal"/>
                    </output>
                </operation>
                <operation name="GetContractInfo">
                    <SOAP:operation/>
                    <input>
                        <SOAP:body use="literal"/>
                    </input>
                    <output>
                        <SOAP:body use="literal"/>
                    </output>
                </operation>
        </binding>
        <service name="GRACENotary">
                <documentation>gSOAP 2.7.2 generated service definition</documentation>
                <port name="GRACENotary" binding="tns:GRACENotary">
                    <SOAP:address location="http://localhost:80"/>
                </port>
        </service>
</WSDL:definitions>
```

6.2 WorkItem List Handler

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="GRACEWI"
 targetNamespace="urn:GRACEWI"
 xmlns:tns="urn:GRACEWI"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:WIListHandler="urn:GRACEWI"
 xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
 xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>
 <documentation>
</documentation>
<schema targetNamespace="urn:GRACEWI"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:WIListHandler="urn:GRACEWI"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <!-- operation request element -->
  <element name="SubmittWorkItem">
   <complexType>
    <sequence>
     <element name="contractId" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <element name="call" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
```

```
  </element>
  <!-- operation response element -->
  <element name="SubmittWorkItemResponse">
   <complexType>
    <sequence>
     <element name="resultSubmitt" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation request element -->
  <element name="GetWIStatus">
   <complexType>
    <sequence>
     <element name="idWI" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="GetWIStatusResponse">
   <complexType>
    <sequence>
     <element name="resultGetStatus" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation request element -->
  <element name="GetWIResult">
   <complexType>
    <sequence>
     <element name="idWI" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
  <!-- operation response element -->
  <element name="GetWIResultResponse">
   <complexType>
    <sequence>
     <element name="resultGetWI" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </sequence>
   </complexType>
  </element>
 </schema>

</types>

<message name="SubmittWorkItemRequest">
 <part name="parameters" element="WIListHandler:SubmittWorkItem"/>
</message>

<message name="SubmittWorkItemResponse">
 <part name="parameters" element="WIListHandler:SubmittWorkItemResponse"/>
</message>

<message name="GetWIStatusRequest">
 <part name="parameters" element="WIListHandler:GetWIStatus"/>
</message>

<message name="GetWIStatusResponse">
 <part name="parameters" element="WIListHandler:GetWIStatusResponse"/>
</message>

<message name="GetWIResultRequest">
 <part name="parameters" element="WIListHandler:GetWIResult"/>
</message>

<message name="GetWIResultResponse">
 <part name="parameters" element="WIListHandler:GetWIResultResponse"/>
</message>

<portType name="rpc">
 <operation name="SubmittWorkItem">
  <documentation>Service definition of function WIListHandler__SubmittWorkItepcf2 </documentation>
  <input message="tns:SubmittWorkItemRequest"/>
  <output message="tns:SubmittWorkItemResponse"/>
 </operation>
 <operation name="GetWIStatus">
  <documentation>Service definition of function WIListHandler__GetWIStatus</documentation>
  <input message="tns:GetWIStatusRequest"/>
```

```
  <output message="tns:GetWIStatusResponse"/>
 </operation>
 <operation name="GetWIResult">
  <documentation>Service definition of function WIListHandler__GetWIResult</documentation>
  <input message="tns:GetWIResultRequest"/>
  <output message="tns:GetWIResultResponse"/>
 </operation>
</portType>

<binding name="GRACEWI" type="tns:rpc">
 <SOAP:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="SubmittWorkItem">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
  <output>
   <SOAP:body use="literal"/>
  </output>
 </operation>
 <operation name="GetWIStatus">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
  <output>
   <SOAP:body use="literal"/>
  </output>
 </operation>
 <operation name="GetWIResult">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="literal"/>
  </input>
  <output>
   <SOAP:body use="literal"/>
  </output>
 </operation>
</binding>

<service name="GRACEWI">
 <documentation>gSOAP 2.7.2 generated service definition</documentation>
 <port name="GRACEWI" binding="tns:GRACEWI">
  <SOAP:address location="http://localhost:80"/>
 </port>
</service>

</definitions>
```

7 Appendice B: Codice ASSIST-CL del Component Administrator

```
#include "StreamStructures.h"

/*************************************/
/*                                   */
/*             Main                  */
/*                                   */
/*************************************/

generic main()
{
        stream command_NoArgs refCTable;
        stream command_RefArg InitScheduler;

        stream command_RefArg ScheduledWI;
        stream command_RefArg StartListener;
        stream command_RefArg WorkItemCompleted;

        Notary(output_stream refCTable);

        WorkItemsListHandler(input_stream refCTable, WorkItemCompleted output_stream InitScheduler);

        Scheduler(input_stream InitScheduler output_stream StartListener, ScheduledWI);
        WorkItemsHandler(input_stream StartListener, ScheduledWI output_stream WorkItemCompleted);
}

/*****************************************************/
/*                                                   */
/*                    Notary                         */
/*                                                   */
/*****************************************************/

parmod Notary(output_stream command_NoArgs refCTable)
{
        topology one vp;

        attribute ref_t ContractTable  onto vp;

        attribute bool started;

        //TODO: control the end

        init
        {
                started = false;
        }

        do input_section
        {
                guard1: on , !started ,
                {
                        operation
                        {
                                started = true;
                        }<use {started}>
                }


                guard2: on , started ,
                {
                }


        } while (true)

        virtual_processors
        {
                elab1 (in guard1 out refCTable)
                {
```

```
                        VP
                        {
                                InitContractTable(in ContractTable out refCTable);
                        }
                }

                elab2 (in guard2 )
                {
                        VP
                        {
                                NotaryAcceptIncomingCommand(in ContractTable);
                        }
                }
        }

        output_section
        {
                collects refCTable from ANY vp;
        }
}

/*******************************************************/
/*                                                     */
/*                    Notary-InitContractTable         */
/*                                                     */
/*******************************************************/
proc InitContractTable(in  ref_t ContractTable out command_NoArgs refCTable)
path<MODULES_PATH_INC, MODULES_PATH_LIB, SOAP_LIB>
inc<"unistd.h", "stdio.h","grace_structdefinition.h">
lib<"libwinotary.so", "libstdsoap2.a">
$c++{
        ContractRecord rec;
        strcpy(rec.Id, "-1");

        // allocate memory
        int recSize = sizeof(ContractRecord) ;
        REF_New (&ContractTable, recSize* DIM_LIST, REF_Default,0);

        for(int i=0; i < DIM_LIST ; i++)
        {
                // store record
                REF_D_Write (&ContractTable, &rec, i* recSize, recSize);
        }

        refCTable.Ref = ContractTable;

        printf(" ----------\tContractTable inited\t----------------\n");

}c++$

/*******************************************************/
/*                                                     */
/*             Notary-AcceptIncomingCommand            */
/*                                                     */
/*******************************************************/
proc NotaryAcceptIncomingCommand(in ref_t ContractTable)
path<MODULES_PATH_INC, MODULES_PATH_LIB, SOAP_LIB, LIBRARY_PATH_LIB>
inc< "stdio.h",  "time.h", "grace_structdefinition.h", "notary_module.h">
lib<"libwinotary.so", "libstdsoap2.a", "libUtilities.a">
$c++{

        ContractRecord contractList[DIM_LIST];
        REF_Lock_t     lock;
        int            size = sizeof(ContractRecord);
        //string              c;

        static int newContractId = 0;

        sleep(5);


        if (NotaryService::Connect(NOTARY_PROVIDE_PORT)< 0)
        {
                printf ("NotaryService: error during port opening\n");
                return;
        }

        while(1)
```

```
	{
		printf("Notary: Listening incoming command\n");

		string cmd = NotaryService::GetCommand() ;

		if (cmd == "GetComponentList")
		{
			printf("GetComponentList called by user = '%s' with pwd = '%s' \n",
				NotaryService::GetUserName().c_str(),
NotaryService::GetPassword().c_str());

			//TODO: Verify logging information

			printf("components uri = %s\n", COMPONENTS_URI);

			//obtain available componentList : TODO: read WSDL URI of each service
			vector<string> cList = NotaryService::ReadComponentList(COMPONENTS_URI);
			NotaryService::SetComponentList(cList);

		}else

		if (cmd == "SubmittContract")
		{
			printf("SubmittContract called by user = '%s' with pwd = '%s' \n",
				NotaryService::GetUserName().c_str(),
NotaryService::GetPassword().c_str());

			// (TODO) verify if the contract is Valid
			if (!NotaryService::IsContractValid())
				NotaryService::SetContractId("-1");
			else
			{
				ContractRecord *contract = NotaryService::GetContract();

				// set contract properties
				contract->Start = time(0);
				contract->MarkedForDelete = false;
				contract->WICounter = 0;

				REF_Lock(&ContractTable, &lock);
				REF_Read(&ContractTable, &contractList);

				bool OK = false;

				for(int i =0; i< DIM_LIST;i++)
				{
					if (!strcmp(contractList[i].Id, "-1"))
					{
						sprintf(contract->Id,"c_%d", ++newContractId);
						REF_D_Write(&ContractTable, contract, size * i, size);
						printf("-New contract with id = '%s' stored\n",
contract->Id);
						NotaryService::SetContractId(contract->Id);
						OK = true;
						break;
					}
				}

				if (!OK)
					NotaryService::SetContractId("-2");

				REF_Unlock(&ContractTable, &lock);
				delete contract;
			}

		}else

		if (cmd == "GetContractProfiles")
		{
			printf("GetProfiles called for component '%s' by user = '%s' with pwd = '%s'
\n",
				NotaryService::GetDescriptionURI().c_str(),
NotaryService::GetUserName().c_str(),
				NotaryService::GetPassword().c_str());

			// read contract profiles
			vector<long> profiles = NotaryService::ReadProfiles(PROFILES,
NotaryService::GetDescriptionURI()) ;
```

```
                printf("profile file = %s\nProfile count = %d\n", PROFILES, profiles.size());

                 NotaryService::SetContractProfiles(profiles);

        }else

        if (cmd == "GetContractInfo")
        {
                printf("GetContractInfo called by user = '%s' with pwd = '%s' \n",
                        NotaryService::GetUserName().c_str(),
NotaryService::GetPassword().c_str());

                ContractRecord c;

                char cId[DIM_ITEM];
                strcpy(cId, NotaryService::GetContractId().c_str());

                REF_Lock(&ContractTable, &lock);
                REF_Read(&ContractTable, &contractList);
                REF_Unlock(&ContractTable, &lock);

                bool OK = false;

                for(int i =0; i< DIM_LIST;i++)
                {
                        if (!strcmp(contractList[i].Id, cId))
                        {
                                NotaryService::SetContract(contractList[i]);
                                OK = true;
                                break;
                        }
                }

                if (!OK)
                {
                        // set empty contract
                        ContractRecord c;
                        strcpy(c.Id,"-1");
                        c.Profile = -1;
                        c.Start = 0;
                        c.ContractDuration = 0;
                        c.WIMaxDuration = 0;
                        c.WICounter = 0;
                        strcpy(c.ResourceReservation, "");
                        strcpy(c.PowerConstraint,"");
                        strcpy(c.ComponentURI,"");
                        c.SpeedConstraint = 0;
                        c.SpeedTolerance = 0;
                        NotaryService::SetContract(c);
                }

        }else

        if (cmd == "DestroyContract")
        {
                printf("DestroyContract called by user = '%s' with pwd = '%s' \n",
                        NotaryService::GetUserName().c_str(),
NotaryService::GetPassword().c_str());

                // mark the contract for delete
                ContractRecord c;

                char cId[DIM_ITEM];
                strcpy(cId, NotaryService::GetContractId().c_str());
                NotaryService::SetLongResult(-1);

                REF_Lock(&ContractTable, &lock);
                REF_Read(&ContractTable, &contractList);

                for(int i =0; i< DIM_LIST;i++)
                {
                        if (!strcmp(contractList[i].Id, cId))
                        {
                                printf("Marking contract with id = '%s' for deleting.\n",
contractList[i].Id);

                                contractList[i].MarkedForDelete = true;
```

```
                                        if (contractList[i].WICounter == 0)
                                        {
                                                strcpy(contractList[i].Id, "-1");
                                                printf("Contract deleted. No Work Item related\n");
                                        }

                                        REF_D_Write(&ContractTable, &contractList[i], size * i, size);

                                        NotaryService::SetLongResult(1);

                                        break;
                                }
                        }

                        REF_Unlock(&ContractTable, &lock);

                }else
                        continue;

                NotaryService::SendResponse();

        }


        NotaryService::Fini();


}c++$

/*************************************/
/*                                   */
/*     WorkItemsListHandler          */
/*                                   */
/*************************************/
parmod WorkItemsListHandler(input_stream command_NoArgs refCTable, command_RefArg WorkItemCompleted
output_stream command_RefArg InitScheduler)
{
        topology none vp;


        do input_section
        {
                guard1: on, , refCTable
                {
                        distribution refCTable on_demand to vp;

                }

                guard2: on, , WorkItemCompleted
                {
                        distribution WorkItemCompleted on_demand to vp;
                }

        } while (true)

        virtual_processors
        {
                elab1 (in guard1 out InitScheduler)
                {
                        VP
                        {
                                InitWIListHanlder(in refCTable output_stream InitScheduler);
                        }
                }

                elab2 (in guard2)
                {
                        VP
                        {
                                StoreWI(in WorkItemCompleted);
                        }
                }
        }

        output_section
        {
                collects InitScheduler from ANY vp;
```

```
	}

}

/*****************************************************/
/*                                                   */
/*            WIListHandler-Init                     */
/*                                                   */
/*****************************************************/
proc InitWIListHanlder(in command_NoArgs refCTable output_stream command_RefArg InitScheduler)
path<MODULES_PATH_INC, MODULES_PATH_LIB, LIBRARY_PATH_LIB, SOAP_LIB>
inc<"unistd.h", "stdio.h", "grace_structdefinition.h", "wilisthandler_module.h">
lib<"libwilisthandler.so", "libstdsoap2.a">
$c++{
	printf("------------\tInitWITable called\t--------\n");

	ref_t ContractTable = refCTable.Ref;
	ref_t WITable;
	ref_t WICount;

	int count = 0;

	REF_New (&WICount, sizeof(int), REF_Default,0);
	REF_Write (&WICount, &count);

	WorkItemRecord rec;
	strcpy(rec.Id, "-1");

	// allocate memory
	int recSize = sizeof(WorkItemRecord);
	REF_New (&WITable, recSize*DIM_LIST, REF_Default,0);

	for(int i=0; i < DIM_LIST ; i++)
	{
		// store record
		REF_D_Write (&WITable, &rec, i* recSize, recSize);
	}

	// fill output stream
	command_RefArg cmd;
	cmd.Arg1 = ContractTable;
	cmd.Arg2 = WITable;
	cmd.Arg3 = WICount;

	assist_out(InitScheduler, cmd);

	printf("\tWIList Handler inited\n");

	static int newWIId = 0;
	char   tmpWIId[DIM_ITEM];

	string contractId;

	string response;
	int    wiStatus;
	bool   found = false;

	REF_Lock_t     lockWI, lockC, lock;
	WorkItemRecord WIRec;
	WorkItemRecord WIList[DIM_LIST];
	ContractRecord contract;
	ContractRecord contractList[DIM_LIST];

	int    sizeWI = sizeof(WorkItemRecord);
	int    sizeC = sizeof(ContractRecord);


	printf ("----------\tWIAcceptIncomingCommand\t----------------\n");

	if (WIListHandlerService::Connect(WIHANLDER_PROVIDE_PORT)< 0)
	{
		printf ("WIListHandlerService: error during port opening\n");
		return;
	}

	while(1)
	{
		printf("WIListHandlerService: Listening incoming command\n");
```

```
		string cmd = WIListHandlerService::GetCommand() ;

		if (cmd == "SubmittWorkItem")
		{
			contractId = WIListHandlerService::GetContractId();

			printf("WIListHandlerService:SubmittWorkItem called  with idContract = '%s'
and call = '%s' \n",
				contractId.c_str(), WIListHandlerService::GetCall().c_str());

			// verify if the contract exists
			REF_Lock(&ContractTable, &lock);
			REF_Read(&ContractTable, &contractList);

			for(int i =0; i< DIM_LIST;i++)
			{
				if (!strcmp(contractList[i].Id, contractId.c_str()))
				{
					printf("WIListHandlerService: contract found. OK\n");

					contractList[i].WICounter++;
					found = true;
					REF_D_Write(&ContractTable, &contractList[i], sizeC * i,
sizeC);
					break;
				}
			}

			REF_Unlock(&ContractTable, &lock);

			if (!found)
			{
				printf("WIListHandlerService: invalid contract Id\n");
				WIListHandlerService::SetWIId("-1");
			}
			else
			{
				// create record
				sprintf(tmpWIId,"wi_%d", newWIId++);

				strcpy(WIRec.Id, tmpWIId);
				strcpy(WIRec.IdContract, contractId.c_str());
				strcpy(WIRec.Call, WIListHandlerService::GetCall().c_str());
				strcpy(WIRec.PhysicalGraph, "");
				strcpy(WIRec.Response,"");
				WIRec.Status = WI_SUBMITTED;
				WIRec.Priority = 0;
				WIRec.SubmittedTime = time(0);
				WIRec.DispatchedTime = -1;
				WIRec.CompletedTime = -1;

				// store Work Item in the registry
				REF_Lock(&WITable, &lockWI);
				REF_Read(&WITable, WIList);

				found = false;

				for(int i=0; i<DIM_LIST; i++)
				{
					if (!strcmp(WIList[i].Id, "-1"))
					{
						REF_D_Write(&WITable, &WIRec, sizeWI *i, sizeWI);
						printf("\tIListHandlerService: Stored WorkItem at
position '%d' with id = '%s' \n", i , tmpWIId);
						found = true;
						break;
					}
				}

				REF_Unlock(&WITable, &lockWI);

				// if there are not enough space in the  WI Table
				if (!found)
				{
					printf("WIListHandlerService: there are not enough space in the
WI Table\n");
					WIListHandlerService::SetWIId("-2");
```

```
                        }
                        else
                        {
                                WIListHandlerService::SetWIId(tmpWIId);

                                // update WICount
                                int wiCount;
                                REF_Lock(&WICount, &lock);
                                REF_Read(&WICount, &wiCount);
                                wiCount++;
                                REF_Write(&WICount, &wiCount);
                                REF_Unlock(&WICount, &lock);
                        }
                }

        }else

        if (cmd == "GetWIStatus")
        {
                string wiId = WIListHandlerService::GetWIId();

                printf("WIListHandlerService:GetWIStatus called for idWorkItem = '%s'\n",
wiId.c_str());

                // search Work Item in the registry
                REF_Lock(&WITable, &lockWI);
                REF_Read(&WITable, WIList);

                found = false;

                for(int i=0; i<DIM_LIST; i++)
                {
                        if (!strcmp(WIList[i].Id, wiId.c_str()))
                        {
                                wiStatus = WIList[i].Status;
                                printf("\tWIListHandlerService: WorkItem '%s' status = '%d'
\n", wiId.c_str(), wiStatus);
                                found = true;
                                break;
                        }
                }

                REF_Unlock(&WITable, &lockWI);

                if (found)
                {
                        WIListHandlerService::SetLongResult(wiStatus);
                }
                else
                {
                        WIListHandlerService::SetLongResult(-1);
                        printf("\tWIListHandlerService: WorkItem '%s' not found\n",
wiId.c_str());
                }

        }else

        if (cmd == "GetWIResult")
        {
                string wiId = WIListHandlerService::GetWIId();

                printf("WIListHandlerService:GetWIResult called for idWorkItem = '%s'\n",
                        wiId.c_str());

                // search Work Item in the registry
                REF_Lock(&WITable, &lockWI);
                REF_Read(&WITable, WIList);

                found = false;

                for(int i=0; i<DIM_LIST; i++)
                {
                        if (!strcmp(WIList[i].Id, wiId.c_str()))
                        {
                                REF_D_Write(&WITable, &WIRec, sizeWI *i, sizeWI);
                                printf("\tWIListHandlerService: WorkItem '%s' response = '%s'
\n", wiId.c_str(), WIList[i].Response);
```

```
                                response = WIList[i].Response;
                                found = true;
                                break;
                        }
                }

                REF_Unlock(&WITable, &lockWI);

                if (found)
                        WIListHandlerService::SetResponse(response);
                else
                {
                        printf("\tWIListHandlerService: WorkItem '%s' not found\n",
wiId.c_str());
                        WIListHandlerService::SetResponse("-1");
                }

        }else
                continue;

        WIListHandlerService::SendResponse();

}


WIListHandlerService::Fini();

}c++$

proc StoreWI(in command_RefArg WorkItemCompleted)
inc<"stdio.h">
$c++{

        printf ("----------- WorkItem completed -----------------\n");

        REF_Reference_t  ContractTable = WorkItemCompleted.Arg1;
        REF_Reference_t  WITable = WorkItemCompleted.Arg2;
        REF_Reference_t  WICount = WorkItemCompleted.Arg3;

        REF_Lock_t     lock, lockWI, lockC;
        WorkItemRecord WIRec;
        int            count;
        ContractRecord contract;
        ContractRecord contractList[DIM_LIST];
        int sizeC = sizeof(ContractRecord);

        // get termination status
        int sizeWI = sizeof(WorkItemRecord);
        int index = WorkItemCompleted.Id;
        int status = WorkItemCompleted.Code;

        // get WorkItem Record
        REF_D_Lock(&WITable, sizeWI *index, sizeWI, &lockWI);
        REF_D_Read(&WITable, &WIRec, sizeWI *index, sizeWI);

        switch (status)
        {
        case  WI_END: WI_ABT:WI_SERR:WI_NV:

                //update WorkItem status
                if (status == WI_END)
                        WIRec.Status = WI_DONE;
                else if (status == WI_ABT)
                        WIRec.Status =  WI_FAILED;
                else if (status == WI_SERR)
                        WIRec.Status =  WI_SYSTEM_ERROR;
                else if (status == WI_NV)
                        WIRec.Status =  WI_NOT_VALID;

                // update contract record
                REF_Lock(&ContractTable, &lockC);
                REF_Read(&ContractTable, &contractList);

                for(int i =0; i< DIM_LIST;i++)
                {
                        if (!strcmp(contractList[i].Id, WIRec.IdContract))
                        {
                                printf("Updating contract with id = '%s'\n",  contractList[i].Id);
```

```
                        contractList[i].WICounter--;

                        // if the Contract is marked for delete and there are no WorkItem
                        // that refer to it then remove the contract
                        if ((contract.WICounter == 0) && (contract.MarkedForDelete))
                                strcpy(contract.Id , "-1");

                        REF_D_Write(&ContractTable, &contractList[i], sizeC * i, sizeC);

                        break;
                }
        }

        REF_Unlock(&ContractTable, &lockC);

    case WI_RSKD:

        // update WICount
        REF_Lock(&WICount, &lock);
        REF_Read(&WICount, &count);
        count++;
        REF_Write(&WICount, &count);
        REF_Unlock(&WICount, &lock);

        // re-submitt the work item
        WIRec.Priority++;
        WIRec.Status = WI_SUBMITTED;

    default:
        printf ("WorkItem status ('%d') not recognized\n", status);
        break;
    }

    // store the WorkItem in the registry
    REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);
    REF_Unlock(&WITable, &lockWI);

}c++$

/*********************************************/
/*      Scheduler                            */
/*      Effettua lo Scheduling delle chiamate       */
/*      in base allo stato delle risorse     */
/*      ed al contratto sottomesso           */
/*********************************************/
parmod Scheduler(input_stream  command_RefArg InitScheduler output_stream command_RefArg
StartListener, command_RefArg ScheduledWI)
{
    topology one vp;

    attribute ref_t ContractTable onto vp;
    attribute ref_t WITable onto vp;
    attribute ref_t DeployTable onto vp;
    attribute ref_t WICount onto vp;

    attribute bool started;

    init
    {
        started = false;
    }

    do input_section
    {

        guard1: on , !started, InitScheduler
        {
            distribution InitScheduler broadcast to vp;

            operation
            {
                started = true;
            }<use {started}>
        }

        guard2: on , started,
        {
```

```
		}

	} while (true)

	virtual_processors
	{
		elab1 (in guard1 out StartListener)
		{
			VP
			{
				SetTables(in InitScheduler, ContractTable, WITable, DeployTable,
WICount out StartListener);
			}
		}

		elab2 (in guard2 out ScheduledWI)
		{
			VP
			{
				Scheduling(in ContractTable, WITable, DeployTable, WICount
output_stream ScheduledWI);
			}
		}
	}

	output_section
	{
		collects StartListener from ANY vp;
		collects ScheduledWI from ANY vp;
	}
}

proc SetTables(in command_RefArg InitScheduler, ref_t ContractTable, ref_t WITable, ref_t
DeployTable, ref_t WICount out command_RefArg StartListener)
path<MODULES_PATH_INC, MODULES_PATH_LIB, LIBRARY_PATH_LIB, SOAP_LIB>
inc<"unistd.h", "stdio.h", "wischeduler_module.h","grace_structdefinition.h">
lib<"libwischeduler.so">
$c++{
	printf("----------\tSetTables called\t----------\n");

	ContractTable = InitScheduler.Arg1;
	WITable = InitScheduler.Arg2;
	WICount = InitScheduler.Arg3;

	int recSize = sizeof(DeployRecord);
	REF_New (&DeployTable, recSize*DIM_LIST, REF_Default,0);

	DeployRecord rec;
	strcpy(rec.Id, "-1");

	for(int i=0; i < DIM_LIST ; i++)
	{
		// store record
		REF_D_Write (&DeployTable, &rec, i* recSize, recSize);
	}

	// send command to start the enforcement event listener
	StartListener.Id = 1;
	StartListener.Arg1 = ContractTable;
	StartListener.Arg2 = WITable;
	StartListener.Arg3 = DeployTable;


}c++$

/*****************************************************************
*	Esegue lo scheduling della WorkItemsList e
*	invia il prossimo WorkItem da eseguire
*****************************************************************/
proc Scheduling(in ref_t ContractTable, ref_t WITable,  ref_t DeployTable, ref_t WICount
	output_stream  command_RefArg ScheduledWI)
path<MODULES_PATH_INC, MODULES_PATH_LIB, LIBRARY_PATH_LIB, SOAP_LIB>
inc<"unistd.h", "stdio.h", "wischeduler_module.h","grace_structdefinition.h">
lib<"libwischeduler.so", "libsedl_descriptionengine.so", "libVPGproxy.a", "libstdsoap2.a",
"libUtilities.a">
$c++{

	printf ("-------\tScheduling.....\t---------\n");
```

```
	sleep(25);

	//variabili di lavoro
	char		tmpWIId[DIM_ITEM];
	int		index = 0;
	int		count;
	long		contractIndex;
	REF_Lock_t	lock;
	WorkItemRecord WI;
	WorkItemRecord WorkItemList[DIM_LIST];
	ContractRecord ContractList[DIM_LIST];

	// verify if there are WI to dispatch
	REF_Lock(&WICount, &lock);
	REF_Read(&WICount, &count);
	REF_Unlock(&WICount, &lock);

	if (count == 0)
	{
		printf("No WorkItems in the list\n");
		return;
	}

	command_RefArg cmd;

	// read contract table
	REF_Lock(&ContractTable, &lock);
	REF_Read (&ContractTable, &ContractList);
	REF_Unlock(&ContractTable, &lock);

	// read WI table and lock
	REF_Lock(&WITable, &lock);
	REF_Read (&WITable, &WorkItemList);

	printf("\tcalling scheduler_module::ScheduleNextWI\n");

	//submitt the call to the scheduler module
	index = scheduler_module::ScheduleNextWI(WorkItemList, ContractList, DIM_LIST,
VPG_MASTER_URI, WI);

	if (index >= 0)
	{
		printf("\tdispatched WorkItem: position ='%d', identifier= '%s'\n",index, WI.Id);

		// store scheduled WorkItem
		int recSize = sizeof(WorkItemRecord);
		REF_D_Write (&WITable, &WI, index*recSize, recSize);
		REF_Unlock(&WITable, &lock);

		// decrement the counter
		REF_Lock(&WICount, &lock);
		REF_Read(&WICount, &count);
		count--;
		REF_Write(&WICount, &count);
		REF_Unlock(&WICount, &lock);

		// fill the output stream
		REF_Reference_t refRes;
		long	result = 0;
		int	sizeResponse = sizeof(long);

		if (!REF_Check(&refRes))
			REF_New (&refRes, sizeResponse, REF_Default,0);

		// write the flag in the memory
		REF_D_Write (&refRes, &result, 0, sizeResponse);

		cmd.Id = index;
		cmd.Size = sizeResponse;
		cmd.Ref = refRes;
		cmd.Arg1 = ContractTable;
		cmd.Arg2 = WITable;
		cmd.Arg3 = DeployTable;
		cmd.Arg4 = WICount;

		assist_out(ScheduledWI, cmd);

		// attende che il nuovo WorkItem inizi la sua esecuzione
```

```
		while(result == 0)
		{
			REF_Read (&(refRes), &result);
			//sleep(RES_TIMEOUT);
		}

	}
	else
	{
		printf("\nNo item dispatched. Error during dispatching. Error code: %d !!!\n",
index);
		REF_Unlock(&WITable, &lock);
	}

}c++$

/*************************************/
/*		WorkItemsHandler		*/
/*				*/
/*************************************/
parmod WorkItemsHandler(input_stream command_RefArg StartListener, command_RefArg ScheduledWI
output_stream command_RefArg WorkItemCompleted)
{
	topology none vp;

	do input_section
	{
		guard1: on, , StartListener
		{
			distribution StartListener on_demand to vp;
		}

		guard2: on, , ScheduledWI
		{
			distribution ScheduledWI on_demand to vp;
		}

	} while (true)

	virtual_processors
	{
		elab1 (in guard1)
		{
			VP
			{
				ListenEnforcement(in StartListener);
			}
		}

		elab2 (in guard2 out WorkItemCompleted)
		{
			VP
			{
				HandleWorkItem(in ScheduledWI output_stream WorkItemCompleted);
			}
		}
	}

	output_section
	{
		collects WorkItemCompleted from ANY vp;
	}
}


proc ListenEnforcement(in command_RefArg StartListener)
path<MODULES_PATH_INC, MODULES_PATH_LIB, LIBRARY_PATH_LIB, SOAP_LIB>
inc<"unistd.h", "stdio.h", "wischeduler_module.h","grace_structdefinition.h">
lib<"libwischeduler.so", "libsedl_descriptionengine.so">
$c++{

	//variabili di lavoro
	char		jobId[DIM_ITEM];
	long		sizeWI = sizeof(WorkItemRecord);
	WorkItemRecord  wiList[DIM_LIST];
	REF_Lock_t	lockWI;

	REF_Reference_t  ContractTable = StartListener.Arg1;
```

```
	REF_Reference_t  WITable = StartListener.Arg2;
	REF_Reference_t  DeployTable = StartListener.Arg3;

	if (scheduler_module::StartEventsListener(EVENT_SERVICE_URI)< 0)
	{
		printf ("WorkItemsHandler module: error during Event handler start-up\n");
		return;
	}

	while(1)
	{
		printf("WorkItemsHandler module: Listening incoming events\n");

		scheduler_module::EventInfo info;

		scheduler_module::ListenEvents(info) ;

		// search WorkItem record
		REF_Lock(&WITable, &lockWI);
		REF_Read(&WITable, &wiList);

		for(int i =0; i< DIM_LIST;i++)
		{
			if (!strcmp(wiList[i].IdJob, info.IdJob.c_str()))
			{
				printf("WorkItemsHandler module: WorkItem found. OK\n");

				// create enforcement directive...
				scheduler_module::ReactToEvent(info, wiList[i]);

				REF_D_Write(&WITable, &wiList[i], sizeWI * i, sizeWI);
				break;
			}
		}

		REF_Unlock(&WITable, &lockWI);
	}

}c++$

/*************************************/
/*                                   */
/*     WorkItemsHandler-HandleWorkItem      */
/*                                   */
/*************************************/
proc HandleWorkItem(in command_RefArg ScheduledWI output_stream command_RefArg WorkItemCompleted)
path<MODULES_PATH_INC, MODULES_PATH_LIB, LIBRARY_PATH_LIB, SOAP_LIB>
inc<"unistd.h", "stdio.h", "wihandler_module.h","grace_structdefinition.h">
lib<"libwihandler.so", "libsedl_descriptionengine.so", "libVPGproxy.a", "libstdsoap2.a",
"libUtilities.a">
$c++{
	printf("--------\tHandleWorkItem called\t--------\n");

	long result = 1;

	static int minPort= 20000;
	static int maxPort= 21000;

	minPort += 10;
	if (minPort >maxPort)
		minPort = 20000;

	// write the flag in the memory: it states that the command is dispatched
	REF_Write(&ScheduledWI.Ref, &result);

	// get shared tables
	REF_Reference_t  ContractTable = ScheduledWI.Arg1;
	REF_Reference_t  WITable = ScheduledWI.Arg2;
	REF_Reference_t  DeployTable = ScheduledWI.Arg3;

	long index = ScheduledWI.Id;
	int sizeWI = sizeof (WorkItemRecord);
	REF_Lock_t     lockWI;
	WorkItemRecord WIRec;

	// init output stream
	command_RefArg WICompl;
	WICompl.Id = index;
```

```
	WICompl.Arg1 = ScheduledWI.Arg1;
	WICompl.Arg2 = ScheduledWI.Arg2;
	WICompl.Arg3 = ScheduledWI.Arg4;

	// read the WorkItem: lock the registry
	REF_D_Lock(&WITable, sizeWI *index, sizeWI, &lockWI);

	// get the WorkItem from the registry
	REF_D_Read (&WITable, &WIRec, sizeWI *index, sizeWI);

	// read the ContractRecord
	REF_Lock_t     lockC;
	ContractRecord  contract;
	ContractRecord  contractList[DIM_LIST];
	int sizeC = sizeof(ContractRecord);

	// read the contract
	REF_Lock(&ContractTable, &lockC);
	REF_Read(&ContractTable, &contractList);

	bool found = false;

	for(int i = 0; i< DIM_LIST; i++)
	{
		if (!strcmp(contractList[i].Id, WIRec.IdContract))
		{
			printf("contract with id '%s' found\n",  contractList[i].Id);

			REF_D_Read (&ContractTable, &contract, sizeC * i, sizeC);

			found = true;
			break;
		}
	}

	REF_Unlock(&ContractTable, &lockC);

	// if the contract isn't available...
	if (!found)
	{
		//...exit
		printf("WIListHandler: error, contract not found.\n");
		// store the WorkItem in the registry
		REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);
		REF_Unlock (&WITable, &lockWI);

		// fill output stream
		WICompl.Code = WI_NV;
		assist_out(WorkItemCompleted,WICompl);
		return;
	}

	string installDir = ROOT_DEPLOY_DIR;
	string cURI = contract.ComponentURI;
	string compName =  cURI.substr(cURI.find_last_of("/")+1, (cURI.find_last_of(".")-
cURI.find_last_of("/")-1));
	installDir += "/" + compName;

	printf("WIHandler_module: Installdir = '%s'\n", installDir.c_str());

	int retCode = WIHandler_module::Init(VPG_MASTER_URI, EVENT_SERVICE_URI, contract, minPort,
maxPort );

	printf("WIHandler_module::Init : Returned value is : '%d'\n", retCode);

	if ( retCode <0)
	{
		// store the WorkItem in the registry
		REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);
		REF_Unlock (&WITable, &lockWI);

		//fill the output stream
		WICompl.Code = WI_SERR;
		assist_out(WorkItemCompleted,WICompl);
		return;
	}

	DeployRecord   DeployList[DIM_LIST];
```

```
REF_Lock_t     lockD;

// read the deploy table
REF_Lock (&DeployTable, &lockD);
REF_Read (&DeployTable, &DeployList);

retCode = WIHandler_module::HandleDeployment(WIRec, DeployList, installDir);

printf("HandleDeployment: Returned value is : '%d'\n", retCode);

REF_Write (&DeployTable, &DeployList);
REF_Unlock (&DeployTable, &lockD);

if (retCode < 0)
{
        // store the WorkItem in the registry
        REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);
        REF_Unlock (&WITable, &lockWI);

        //fill the output stream
        WICompl.Code = WI_SERR;
        assist_out(WorkItemCompleted,WICompl);
        printf("assist_out done\n");

        return;
        printf("return done\n");
}

retCode = WIHandler_module::HandleEnactment(WIRec);

// update WI status
WIRec.Status = WI_INITED;
REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);

printf("HandleEnactment: Returned value is : '%d'\n", retCode);

if (retCode < 0)
{
        // store the WorkItem in the registry
        REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);
        REF_Unlock (&WITable, &lockWI);

        //fill the output stream
        WICompl.Code = WI_SERR;
        assist_out(WorkItemCompleted,WICompl);
        printf("assist_out done\n");

        return;
        printf("return done\n");
}

//TODO: wait that the component is up
sleep(2);

// update WI status
WIRec.Status = WI_RUNNING;
REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);

retCode =  WIHandler_module::HandleCall(WIRec);

printf("HandleCall: Returned value is : '%d'\n", retCode);

if (retCode < 0)
{
        // store the WorkItem in the registry
        REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);
        REF_Unlock (&WITable, &lockWI);

        //fill the output stream
        WICompl.Code = WI_SERR;
        assist_out(WorkItemCompleted,WICompl);
        printf("assist_out done\n");

        return;
        printf("return done\n");
}

retCode = WIHandler_module::Wait(WIRec);
```

```
        printf("Returned value is : '%d'\n", retCode);

        WICompl.Code = retCode;

        // store the WorkItem in the registry
        REF_D_Write (&WITable,  &WIRec, sizeWI *index, sizeWI);
        REF_Unlock (&WITable, &lockWI);

        WIHandler_module::Fini();

        assist_out(WorkItemCompleted,WICompl);

}c++$
```

8 Appendice C: ComponentAdministrator Namespace Documentation

8.1 NotaryService Namespace Reference

8.1.1 Classes

class **ContractHandler**
struct **Notary__ContractRecord**

8.1.2 Typedefs

typedef long **xsd__int**

8.1.3 Functions

bool **IsContractValid** ()
vector< string > **ReadComponentList** (string repURL)
vector< long > **ReadProfiles** (string docURL, string compName)
int **Connect** (int portNumber)
int **SendResponse** ()
void **Fini** ()
string **GetCommand** ()
int **SetLongResult** (long result)
ContractRecord * **GetContract** ()
int **SetContract** (ContractRecord contract)
string **GetUserName** ()
string **GetPassword** ()
string **GetDescriptionURI** ()
string **GetContractId** ()
int **SetContractId** (string id)
int **SetContractProfiles** (vector< long > contractProfiles)
int **SetComponentList** (vector< string > componentList)
int **Notary__GetComponentList** (std::string userName, std::string pwd, std::vector< std::string > &result1)
int **Notary__GetContractProfiles** (std::string descriptionURI, std::string userName, std::string pwd, std::vector< **xsd__int** > &result2)
int **Notary__SubmittContract** (std::string userName, std::string pwd, struct **Notary__ContractRecord** contract, std::string &result3)
int **Notary__DestroyContract** (std::string userName, std::string pwd, std::string contractId, **xsd__int** &result4)
int **Notary__GetContractInfo** (std::string contractId, std::string userName, std::string pwd, struct **Notary__ContractRecord** &result5)

8.1.4 Typedef Documentation

8.1.4.1 typedef long NotaryService::xsd__int

8.1.5 Function Documentation

8.1.5.1 int NotaryService::Connect (int *portNumber*)

Open a port to listen incoming SOAP command

8.1.5.1.1 Parameters:

portNumber the number of port to open

8.1.5.1.2 Returns:

1 all done, -1 port bind failed

8.1.5.2 void NotaryService::Fini ()

8.1.5.3 string NotaryService::GetCommand ()

Get the name of called command

8.1.5.3.1 Returns:

the called command

8.1.5.4 ContractRecord* NotaryService::GetContract ()

Get the "ContractRecord" param of the remote call/response

8.1.5.5 string NotaryService::GetContractId ()

Get the "ContractId" param of the remote call/response

8.1.5.6 string NotaryService::GetDescriptionURI ()

Get the "DescriptionURI" param of the remote call/response

8.1.5.7 string NotaryService::GetPassword ()

Get the "Password" param of the remote call/response

8.1.5.8 string NotaryService::GetUserName ()

Get the "UserName" param of the remote call/response

8.1.5.9 bool NotaryService::IsContractValid ()

Verify if a contract is valid. This means to analize the compatibility of contract options

8.1.5.9.1 Returns:

true if the contract is valid, false otherwise

8.1.5.10 int NotaryService::Notary__DestroyContract (std::string *userName*, std::string *pwd*, std::string *contractId*, xsd__int & *result4*)

Distrugge un contratto.

8.1.5.10.1 Parameters:

userName account utente
pwd password dell'utente
contractId identificatore del contratto

8.1.5.11 int NotaryService::Notary__GetComponentList (std::string *userName*, std::string *pwd*, std::vector< std::string > & *result1*)

Restituisce l'elenco dei componenti disponibili (URI della descrizione WSDL)

8.1.5.11.1 Parameters:

userName account utente
pwd password dell'utente

8.1.5.11.2 Returns:

lista delle URI

8.1.5.12 int NotaryService::Notary__GetContractInfo (std::string *contractId*, std::string *userName*, std::string *pwd*, struct Notary__ContractRecord & *result5*)

Ritorna informazioni su una istanza WS

8.1.5.12.1 Parameters:

userName account utente
pwd password dell'utente

8.1.5.13 int NotaryService::Notary__GetContractProfiles (std::string *descriptionURI*, std::string *userName*, std::string *pwd*, std::vector< xsd__int > & *result2*)

Restituisce i template di contratti disponibili per una determinto componente

8.1.5.13.1 Parameters:

userName account utente
pwd password dell'utente
descriptionURI URI di descrizione della WS

8.1.5.13.2 Returns:

lista di profili di contratti disponibili per un determinato componente

8.1.5.14 int NotaryService::Notary__SubmittContract (std::string *userName*, std::string *pwd*, struct Notary__ContractRecord *contract*, std::string & *result3*)

Sottoscrizione di un contratto, ogni contratto è riferito ad un particolare metodo del componente

8.1.5.14.1 Parameters:

userName account utente
pwd password dell'utente
contract scontratto di utilizzo del metodo che si intende utilizzare

8.1.5.14.2 Returns:

returned value are:
l'identificatore del contratto :(ws-addressing end point) "-1". componente non disponibile "-2". Contratto non valido o incompatibile col componente

8.1.5.15 vector<string> NotaryService::ReadComponentList (string *repURL*)

Read the list of availables components from specified URL

8.1.5.15.1 Parameters:

repURL the repository URL where is located component descriptors

8.1.5.16 vector<long> NotaryService::ReadProfiles (string *docURL*, string *compName*)

Read availables contract profiles for a specified Component

8.1.5.16.1 Parameters:

docURL URL of the document where availables profiles are indicated
component name

8.1.5.16.2 Returns:

a list of profiles

8.1.5.17 int NotaryService::SendResponse ()

Send the response to client application

8.1.5.17.1 Returns:

returned value are: 0. done -1. generic error

8.1.5.18 int NotaryService::SetComponentList (vector< string > *componentList*)

Set the "ComponentList" param of the remote call/response

8.1.5.19 int NotaryService::SetContract (ContractRecord *contract*)

Set the "ContractRecord" param of the remote call/response

8.1.5.20 int NotaryService::SetContractId (string *id*)

Set the "ContractId" param of the remote call/response

8.1.5.21 int NotaryService::SetContractProfiles (vector< long > *contractProfiles*)

Set the "ContractProfiles" param of the remote call/response

8.1.5.22 int NotaryService::SetLongResult (long *result*)

Set a long value to return to client application

8.1.5.22.1 Parameters:

return value of the call

8.1.5.22.2 Returns:

returned value are: 0. done -1. error

8.1.5.23

8.2 scheduler_module Namespace Reference

8.2.1 Classes

class **WIEventHandler**
class **WIScheduler**
struct **EventInfo**

8.2.2 Functions

int **ScheduleNextWI** (WorkItemRecord WorkItemList[], ContractRecord ContractList[], int dimList, string serverURI, WorkItemRecord &WI)
int **StartEventsListener** (string masterPortEvent)
int **ListenEvents** (**EventInfo** &info)
int **SendEvent** (**EventInfo** info)
int **ReactToEvent** (**EventInfo** &info, WorkItemRecord &WIRec)

8.2.3 Detailed Description

8.2.3.1.1 Author:

assist user

8.2.4 Function Documentation

8.2.4.1 int scheduler_module::ListenEvents (EventInfo & *info*)

Get an event sent by the JobInspector

8.2.4.2 int scheduler_module::ReactToEvent (EventInfo & *info*, WorkItemRecord & *WIRec*)

Analize an incoming event, modify the work-item status and create the outcoming event

8.2.4.3 int scheduler_module::ScheduleNextWI (WorkItemRecord *WorkItemList*[], ContractRecord *ContractList*[], int *dimList*, string *serverURI*, WorkItemRecord & *WI*)

Schedule the next Work Item from the WotkItem list. It modify the WI description

8.2.4.3.1 Parameters:

WorkItemList the list of WorkItem
WorkItem description of dispatched WorkItem
dimList lenght of the list
WI WorkItem Record modified by the module

serverURI address of VPGMaster port

8.2.4.3.2 Returns:

the position of the WorkItem Dispatched. The follow error code on error: -1. error during connection with VPGMaster -2. error during the comunication with VPGMaster -3. not enough resources are available -4. there isn't WorkItm to dispatch -5. Error during contract description handling -6. error during WorkItem description handling -7. Contract option not valid -8. No workItem to dispatch

8.2.4.4 int scheduler_module::SendEvent (EventInfo *info*)

Send an enforcement event for a component

8.2.4.5 int scheduler_module::StartEventsListener (string *masterPortEvent*)

Start the listener of enforcement events

8.3 WIHandler_module Namespace Reference

8.3.1 Classes

class **WIHandler**

8.3.2 Functions

int **Init** (string serverURI, string eventEngineURI, ContractRecord contract, int minPort, int maxPort)
int **HandleDeployment** (WorkItemRecord &WIRec, DeployRecord deployTable[], string installDir)
int **HandleEnactment** (WorkItemRecord &WIRec)
int **HandleCall** (WorkItemRecord &WIRec)
int **Wait** (WorkItemRecord &WIRec)
int **Fini** ()

8.3.3 Function Documentation

8.3.3.1 int WIHandler_module::Fini ()

8.3.3.2 int WIHandler_module::HandleCall (WorkItemRecord & *WIRec*)

8.3.3.3 int WIHandler_module::HandleDeployment (WorkItemRecord & *WIRec*, DeployRecord *deployTable*[], string *installDir*)

8.3.3.4 int WIHandler_module::HandleEnactment (WorkItemRecord & *WIRec*)

8.3.3.5 int WIHandler_module::Init (string *serverURI*, string *eventEngineURI*, ContractRecord *contract*, int *minPort*, int *maxPort*)

Handle the lifecycle of the WorkItem

8.3.3.5.1 Parameters:

WorkItem description. It will be modified
contract the copy of the contract of the WorkItem

8.3.3.5.2 Returns:

returned value are: 0. WI_END - WorkItem execution ends correctly 1. WI_RSKD - WorkItem execution ends incorrectly, it is to reschedule 2. WI_ABT - WorkItem execution ends incorrectly, it aborts -2. an error occurs during deployment of Component -3. an error occurs during enactment of Component -4. an error occurs during call -5. Error during parsing of physical graph -6. Error during parsing of Component Description

8.3.3.6 int WIHandler_module::Wait (WorkItemRecord & *WIRec*)

8.4 WIListHandlerService Namespace Reference

8.4.1 Typedefs

typedef long **xsd__int**

8.4.2 Functions

int **Connect** (int portNumber)
string **GetCommand** ()
string **GetWIId** ()
int **SetWIId** (string idWI)
string **GetContractId** ()
int **SetContractId** (string idContract)
int **SetLongResult** (long result)
string **GetCall** ()
int **SetCall** (string call)
string **GetResponse** ()
int **SetResponse** (string response)
int **SendResponse** ()
void **Fini** ()
int **WIListHandler__SubmittWorkItem** (std::string contractId, std::string call, std::string &resultSubmitt)
int **WIListHandler__GetWIStatus** (std::string idWI, **xsd__int** &resultGetStatus)
int **WIListHandler__GetWIResult** (std::string idWI, std::string &resultGetWI)

8.4.3 Typedef Documentation

8.4.3.1 typedef long WIListHandlerService::xsd__int

8.4.4 Function Documentation

8.4.4.1 int WIListHandlerService::Connect (int *portNumber*)

Open a port to listen incoming SOAP command

8.4.4.1.1 Parameters:

portNumber the number of port to open

8.4.4.1.2 Returns:

1 all done, -1 port bind failed

8.4.4.2 void WIListHandlerService::Fini ()

8.4.4.3 string WIListHandlerService::GetCall ()

Get the "Call" param of the remote call/response

8.4.4.4 string WIListHandlerService::GetCommand ()

8.4.4.5 string WIListHandlerService::GetContractId ()

Get the "ContractId" param of the remote call/response

8.4.4.6 string WIListHandlerService::GetResponse ()

Get the "Response" param of the remote call/response

8.4.4.7 string WIListHandlerService::GetWIId ()

Get the "WorkItemId" param of the remote call/response

8.4.4.8 int WIListHandlerService::SendResponse ()

8.4.4.9 int WIListHandlerService::SetCall (string *call*)

Set the "Call" param of the remote call/response

8.4.4.10 int WIListHandlerService::SetContractId (string *idContract*)

Set the "ContractId" param of the remote call/response

8.4.4.11 int WIListHandlerService::SetLongResult (long *result*)

Set the value (long) of the response

8.4.4.12 int WIListHandlerService::SetResponse (string *response*)

Get the "Response" param of the remote call/response

8.4.4.13 int WIListHandlerService::SetWIId (string *idWI*)

Set the "WorkItemId" param of the remote call/response

8.4.4.14 int WIListHandlerService::WIListHandler__GetWIResult (std::string *idWI*, std::string & *resultGetWI*)

restituisce la busta di risposta. Se il WI non ha terminato restituisce una stringa vuota;

8.4.4.15 int WIListHandlerService::WIListHandler__GetWIStatus (std::string *idWI*, xsd__int & *resultGetStatus*)

Restituisce lo stato di elaborazione della chiamata (vedi stato di un WorkItem)

8.4.4.16 int WIListHandlerService::WIListHandler__SubmittWorkItem (std::string *contractId*, std::string *call*, std::string & *resultSubmitt*)

Sottomette una chiamata ad un modulo istanziato durante la sottomissione del contratto. Ritorna un identificatore di chiamata o un codice di errore

9 Appendice D: ComponentAdministrator Class Documentation

9.1 NotaryService::ContractHandler Class Reference

`#include <contracthandler.h>`

9.1.1 Public Member Functions

ContractHandler ()
~ContractHandler ()

9.1.2 Static Public Member Functions

static bool **IsContractValid** (ContractRecord contract)
static vector< string > **ReadComponentList** (string repURL)
static vector< long > **ReadProfiles** (string docURL, string compName)

9.1.3 Detailed Description

This class manages the contracts. It reads profiles availables for a component and analize the validity of submitted contracts

---

9.1.4 Constructor & Destructor Documentation

9.1.4.1 NotaryService::ContractHandler::ContractHandler ()

Init the class

9.1.4.2 NotaryService::ContractHandler::~ContractHandler ()

Release resources

9.1.5 Member Function Documentation

9.1.5.1 static bool NotaryService::ContractHandler::IsContractValid (ContractRecord *contract*) [static]

Verify if a contract is valid. This means to analize the compatibility of contract options

9.1.5.1.1 Returns:

true if the contract is valid, false otherwise

9.1.5.2 static vector<string> NotaryService::ContractHandler::ReadComponentList (string *repURL*) [static]

Read the list of availables components from specified URL

9.1.5.2.1 Parameters:

repURL the repository URL where is located component descriptors

9.1.5.3 static vector<long> NotaryService::ContractHandler::ReadProfiles (string *docURL*, string *compName*) [static]

Read availables contract profiles for a specified Component

9.1.5.3.1 Parameters:

docURL URL of the document where availables profiles are indicated
component name

9.1.5.3.2 Returns:

a list of profiles

The documentation for this class was generated from the following file:
E:/tmp/Administrator-doxygen/**contracthandler.h**

9.1.5.4

9.2 scheduler_module::EventInfo Struct Reference

`#include <wischeduler_module.h>`

9.2.1 Public Attributes

string **EventName**
string **IdJob**
string **IdNode**
string **PhysicalGraph**
float **DeltaPower**

9.2.2 Detailed Description

This struct contain the dataset sent inside the enforcement event QOS_INC

9.2.3 Member Data Documentation

9.2.3.1 float scheduler_module::EventInfo::DeltaPower

9.2.3.2 string scheduler_module::EventInfo::EventName

9.2.3.3 string scheduler_module::EventInfo::IdJob

9.2.3.4 string scheduler_module::EventInfo::IdNode

9.2.3.5 string scheduler_module::EventInfo::PhysicalGraph

The documentation for this struct was generated from the following file:

E:/tmp/Administrator-doxygen/**wischeduler_module.h**

9.3 NotaryService::Notary__ContractRecord Struct Reference

`#include <NotaryService.h>`

9.3.1 Public Attributes

xsd__int Profile
xsd__int ContractDuration
xsd__int WIMaxDuration
std::string **PowerConstraint**
float **SpeedConstraint**
float **SpeedTolerance**
std::string **ComponentURI**
std::string **ResourceReservation**

9.3.2 Member Data Documentation

9.3.2.1 std::string NotaryService::Notary__ContractRecord::ComponentURI

9.3.2.2 xsd__int NotaryService::Notary__ContractRecord::ContractDuration

9.3.2.3 std::string NotaryService::Notary__ContractRecord::PowerConstraint

9.3.2.4 xsd__int NotaryService::Notary__ContractRecord::Profile

9.3.2.5 std::string NotaryService::Notary__ContractRecord::ResourceReservation

9.3.2.6 float NotaryService::Notary__ContractRecord::SpeedConstraint

9.3.2.7 float NotaryService::Notary__ContractRecord::SpeedTolerance

9.3.2.8 xsd__int NotaryService::Notary__ContractRecord::WIMaxDuration

---

The documentation for this struct was generated from the following file:

E:/tmp/Administrator-doxygen/**NotaryService.h**

9.4 scheduler_module::WIEventHandler Class Reference

`#include <wieventhandler.h>`

9.4.1 Public Member Functions

~WIEventHandler ()
void **EventOccur** (const string &event, const AL_DataSet &data)
void **Init** (string masterEventPort)
void **TriggerEvent** (string poolName, string eventName, string stringDataName, string stringDataValue)
void **TriggerEvent** (string poolName, string eventName, string stringDataName1, string stringDataValue1, string stringDataName2, string stringDataValue2, string stringDataName3, string stringDataValue3)
void **InstallHandlers** ()
string **Listen** ()
void **SetEventName** (string EventName)
void **SetJobId** (string JobId)
void **SetNodeId** (string NodeId)
void **SetPhysicalGraph** (string PhysicalGraph)
void **SetDeltaPower** (float DeltaPower)
string **GetEventName** ()
string **GetJobId** ()
string **GetNodeId** ()
string **GetPhysicalGraph** ()
float **GetDeltaPower** ()

9.4.2 Static Public Member Functions

static **WIEventHandler** * **GetInstance** ()

9.4.3 Private Member Functions

WIEventHandler ()
void **Register** (string eventPool)
void **ManageEnd** (const AL_DataSet &data)
void **ManageStart** (const AL_DataSet &data)
void **Notify** (const AL_DataSet &data)

9.4.4 Private Attributes

AL_Notifier< **WIEventHandler** > **notifier**
The event Notifier.

AL_Dispatcher< **WIEventHandler** > **dispatcher**
The event Dispatcher.

string **_eventName**
string **_jobId**
string **_nodeId**
string **_physicalGraph**
float **_deltaPower**

9.4.5 Static Private Attributes

static **WIEventHandler** * **_instance**

9.4.6 Constructor & Destructor Documentation

9.4.6.1 scheduler_module::WIEventHandler::~WIEventHandler ()

9.4.6.2 scheduler_module::WIEventHandler::WIEventHandler () `[private]`

9.4.7 Member Function Documentation

9.4.7.1 void scheduler_module::WIEventHandler::EventOccur (const string & *event*, const AL_DataSet & *data*)

Handles the incoming event

9.4.7.1.1 Parameters:

event the name of the event
data the dataset inside the event

9.4.7.2 float scheduler_module::WIEventHandler::GetDeltaPower () [inline]

9.4.7.3 string scheduler_module::WIEventHandler::GetEventName () [inline]

9.4.7.4 static WIEventHandler* scheduler_module::WIEventHandler::GetInstance () [static]

Get the unique instance of the class (it implement the "singletone pattern")

9.4.7.5 string scheduler_module::WIEventHandler::GetJobId () [inline]

9.4.7.6 string scheduler_module::WIEventHandler::GetNodeId () [inline]

9.4.7.7 string scheduler_module::WIEventHandler::GetPhysicalGraph () [inline]

9.4.7.8 void scheduler_module::WIEventHandler::Init (string *masterEventPort*)

Initialize the event system

9.4.7.9 void scheduler_module::WIEventHandler::InstallHandlers ()

Install the observer for an event

9.4.7.10 string scheduler_module::WIEventHandler::Listen ()

start the listener of incoming events

9.4.7.11 void scheduler_module::WIEventHandler::ManageEnd (const AL_DataSet & *data*) [private]

9.4.7.12 void scheduler_module::WIEventHandler::ManageStart (const AL_DataSet & *data*) [private]

9.4.7.13 void scheduler_module::WIEventHandler::Notify (const AL_DataSet & *data*) [private]

9.4.7.14 void scheduler_module::WIEventHandler::Register (string *eventPool*) [private]

9.4.7.15 void scheduler_module::WIEventHandler::SetDeltaPower (float *DeltaPower*) [inline]

9.4.7.16 void scheduler_module::WIEventHandler::SetEventName (string *EventName*) [inline]

9.4.7.17 void scheduler_module::WIEventHandler::SetJobId (string *JobId*) [inline]

9.4.7.18 void scheduler_module::WIEventHandler::SetNodeId (string *NodeId*) [inline]

9.4.7.19 void scheduler_module::WIEventHandler::SetPhysicalGraph (string *PhysicalGraph*) [inline]

9.4.7.20 void scheduler_module::WIEventHandler::TriggerEvent (string *poolName*, string *eventName*, string *stringDataName1*, string *stringDataValue1*, string *stringDataName2*, string *stringDataValue2*, string *stringDataName3*, string *stringDataValue3*)

9.4.7.21 void scheduler_module::WIEventHandler::TriggerEvent (string *poolName*, string *eventName*, string *stringDataName*, string *stringDataValue*)

Send an event

9.4.8 Member Data Documentation

9.4.8.1 float scheduler_module::WIEventHandler::_deltaPower [private]

9.4.8.2 string scheduler_module::WIEventHandler::_eventName [private]

9.4.8.3 WIEventHandler* scheduler_module::WIEventHandler::_instance [static, private]

9.4.8.4 string scheduler_module::WIEventHandler::_jobId [private]

9.4.8.5 string scheduler_module::WIEventHandler::_nodeId [private]

9.4.8.6 string scheduler_module::WIEventHandler::_physicalGraph [private]

9.4.8.7 AL_Dispatcher<WIEventHandler> scheduler_module::WIEventHandler::dispatcher [private]

The event Dispatcher.

9.4.8.8 AL_Notifier<WIEventHandler> scheduler_module::WIEventHandler::notifier [private]

The event Notifier.

The documentation for this class was generated from the following file:

E:/tmp/Administrator-doxygen/**wieventhandler.h**

9.5 WIHandler_module::WIHandler Class Reference

`#include <wihandler.h>`

9.5.1 Public Member Functions

WIHandler ()
~WIHandler ()
bool **Init** (string serverURI, string eventEngineURI, ContractRecord contract, int minPort, int maxPort)
bool **Fini** ()
int **HandleDeployment** (WorkItemRecord &WIRec, DeployRecord deployTable[], string installDir)
int **HandleEnactment** (WorkItemRecord &WIRec)
int **HandleCall** (WorkItemRecord &WIRec)
int **Wait** ()
void **ManageQoSEnd** (string graphId)
void **ManageQoSRskd** (string graphId)
void **ManageQoSAbort** (string graphId)

9.5.2 Private Member Functions

int **VerifyDeployment** (string resoureceURI, int processIndex, AbstractNode *absNode, DeployRecord deployTable[])
int **AddImplementationUnit** (ImplementationUnit *impl, string resourceURI, int processIndex, DeployRecord deployTable[])
string **GetGraphId** (string componentURI, DeployRecord deployTable[])
verify if there is a deployed graph for this component

bool **FindFile** (DeployRecord deployTable[], string fileName, DeployRecord &deployRecord)
update the graphId Value

void **AddGraphToTable** (DeployRecord deployTable[], string idGraph)
void **AddFileToTable** (DeployRecord deployTable[], DeployRecord)
bool **NodeExists** (char hostList[HOST_LIST][DIM_ITEM], string resourceURI)
verify if the file is deployed in that host

void **NodeAdd** (char hostList[HOST_LIST][DIM_ITEM], string resourceURI)
void **UpdateEnactCmd** (DeployRecord se, string resourceURI, int processIndex)
string **GetPortNumber** (int *r_port=0)
string **GetCompType** (string componentURI)
string **GetCompName** (string componentURI)
void **GetInitEventData** (vector< string > &dataName, vector< string > &dataValue)

9.5.3 Private Attributes

VPGMasterProxy_Soap * **_vpgProxy**
WIEventHandler * **_eventHandler**
vector< SoftElem > **_deployArg**
vector< Task > **_enactArg**
map< int, int > **_componentPorts**
SoftwareElement * **_softElem**
QoSGraph * **_physicalGraph**
ContractRecord **_contract**
string **_componentURI**
string **_eventEngineURI**
string **_graphId**
string **_jobId**
string **_installDir**
string **_compType**
string **_compName**
string **_str_physicalGraph**
CommandSender * **_cmdSender**
int **_enforceCode**
ACE_Thread_Mutex **_mutex**
attribute for the sincronization between the two threads

ACE_Condition< ACE_Thread_Mutex > * **_guard**
int **_count**
int **_minPort**
int **_maxPort**
int **_currentPort**
map< int, int > **_ports**

9.5.4 Detailed Description

9.5.4.1.1 Author:

lombardo saverio

9.5.5 Constructor & Destructor Documentation

9.5.5.1 WIHandler_module::WIHandler::WIHandler ()

9.5.5.2 WIHandler_module::WIHandler::~WIHandler ()

9.5.6 Member Function Documentation

9.5.6.1 void WIHandler_module::WIHandler::AddFileToTable (DeployRecord *deployTable*[], DeployRecord) [private]

9.5.6.2 void WIHandler_module::WIHandler::AddGraphToTable (DeployRecord *deployTable*[], string *idGraph*) [private]

9.5.6.3 int WIHandler_module::WIHandler::AddImplementationUnit (ImplementationUnit * *impl*, string *resourceURI*, int *processIndex*, DeployRecord *deployTable*[]) [private]

9.5.6.4 bool WIHandler_module::WIHandler::FindFile (DeployRecord *deployTable*[], string *fileName*, DeployRecord & *deployRecord*) [private]

update the graphId Value

9.5.6.5 bool WIHandler_module::WIHandler::Fini ()

9.5.6.6 string WIHandler_module::WIHandler::GetCompName (string *componentURI*) [private]

9.5.6.7 string WIHandler_module::WIHandler::GetCompType (string *componentURI*) [private]

9.5.6.8 string WIHandler_module::WIHandler::GetGraphId (string *componentURI*, DeployRecord *deployTable*[]) [private]

verify if there is a deployed graph for this component

9.5.6.9 void WIHandler_module::WIHandler::GetInitEventData (vector< string > & *dataName*, vector< string > & *dataValue*) [private]

9.5.6.10 string WIHandler_module::WIHandler::GetPortNumber (int * *r_port* = 0) [private]

9.5.6.11 int WIHandler_module::WIHandler::HandleCall (WorkItemRecord & *WIRec*)

Submitt the call to the component

9.5.6.11.1 Returns:

returned value are: 1. all done -2 Front-end port number not found

9.5.6.12 int WIHandler_module::WIHandler::HandleDeployment (WorkItemRecord & *WIRec*, DeployRecord *deployTable*[], string *installDir*)

Verify the deployment statis of the graph and deploy files if necessary

9.5.6.12.1 Parameters:

WIRec description of the WorkItem to execute
deployTable table of the current deployment files on the VPG
QoSOptions options for the Quality of Service to pass to the Inspector

9.5.6.12.2 Returns:

returned value are: 1. all done -1. An error occurs during physical graph description parsing -2. the physical graph description or the **SEDL** description are incoerents or invalid -3. an error occurs during deployment of Component

9.5.6.13 int WIHandler_module::WIHandler::HandleEnactment (WorkItemRecord & *WIRec*)

9.5.6.13.1 Parameters:

WIRec description of the WorkItem to execute

9.5.6.13.2 Returns:

rules rules
returned value are: 1. all done -1. -2. -3. -4. caught execption during VPGMaster communication

9.5.6.14 bool WIHandler_module::WIHandler::Init (string *serverURI*, string *eventEngineURI*, ContractRecord *contract*, int *minPort*, int *maxPort*)

9.5.6.15 void WIHandler_module::WIHandler::ManageQoSAbort (string *graphId*)

9.5.6.16 void WIHandler_module::WIHandler::ManageQoSEnd (string *graphId*)

9.5.6.17 void WIHandler_module::WIHandler::ManageQoSRskd (string *graphId*)

9.5.6.18 void WIHandler_module::WIHandler::NodeAdd (char *hostList*[HOST_LIST][DIM_ITEM], string *resourceURI*) `[private]`

9.5.6.19 bool WIHandler_module::WIHandler::NodeExists (char *hostList*[HOST_LIST][DIM_ITEM], string *resourceURI*) `[private]`

verify if the file is deployed in that host

9.5.6.20 void WIHandler_module::WIHandler::UpdateEnactCmd (DeployRecord *se*, string *resourceURI*, int *processIndex*) `[private]`

update the enactment command with a process as described by the deploy Record

9.5.6.21 int WIHandler_module::WIHandler::VerifyDeployment (string *resoureceURI*, int *processIndex*, AbstractNode * *absNode*, DeployRecord *deployTable*[]) `[private]`

9.5.6.22 int WIHandler_module::WIHandler::Wait ()

9.5.7 Member Data Documentation

9.5.7.1 CommandSender* WIHandler_module::WIHandler::_cmdSender [private]

9.5.7.2 string WIHandler_module::WIHandler::_compName [private]

9.5.7.3 map<int, int> WIHandler_module::WIHandler::_componentPorts [private]

9.5.7.4 string WIHandler_module::WIHandler::_componentURI [private]

9.5.7.5 string WIHandler_module::WIHandler::_compType [private]

9.5.7.6 ContractRecord WIHandler_module::WIHandler::_contract [private]

9.5.7.7 int WIHandler_module::WIHandler::_count [private]

9.5.7.8 int WIHandler_module::WIHandler::_currentPort [private]

9.5.7.9 vector<SoftElem> WIHandler_module::WIHandler::_deployArg [private]

9.5.7.10 vector<Task> WIHandler_module::WIHandler::_enactArg [private]

9.5.7.11 int WIHandler_module::WIHandler::_enforceCode [private]

9.5.7.12 string WIHandler_module::WIHandler::_eventEngineURI [private]

9.5.7.13 WIEventHandler* WIHandler_module::WIHandler::_eventHandler [private]

9.5.7.14 string WIHandler_module::WIHandler::_graphId [private]

9.5.7.15 ACE_Condition<ACE_Thread_Mutex>* WIHandler_module::WIHandler::_guard [private]

9.5.7.16 string WIHandler_module::WIHandler::_installDir [private]

9.5.7.17 string WIHandler_module::WIHandler::_jobId [private]

9.5.7.18 int WIHandler_module::WIHandler::_maxPort [private]

9.5.7.19 int WIHandler_module::WIHandler::_minPort [private]

9.5.7.20 ACE_Thread_Mutex WIHandler_module::WIHandler::_mutex [private]

attribute for the sincronization between the two threads

9.5.7.21 QoSGraph* WIHandler_module::WIHandler::_physicalGraph [private]

9.5.7.22 map<int,int> WIHandler_module::WIHandler::_ports [private]

9.5.7.23 SoftwareElement* WIHandler_module::WIHandler::_softElem [private]

9.5.7.24 string WIHandler_module::WIHandler::_str_physicalGraph [private]

9.5.7.25 VPGMasterProxy_Soap* WIHandler_module::WIHandler::_vpgProxy [private]

The documentation for this class was generated from the following file:

E:/tmp/Administrator-doxygen/**wihandler.h**

9.6 scheduler_module::WIScheduler Class Reference

`#include <wischeduler.h>`

9.6.1 Public Member Functions

WIScheduler ()
~WIScheduler ()
int **ScheduleNextWI** (WorkItemRecord WorkItemList[], ContractRecord ContractList[], int dimList, WorkItemRecord &WI)
bool **Init** (string serverURI)
bool **Fini** ()

9.6.2 Private Member Functions

int **GetNextIndex** (WorkItemRecord WorkItemList[], int dimList)
int **SetMapping** (QoSGraph *resourceReservation, QoSGraph *powerConstrain, vector< CurrentStatus > vpgStatus, WorkItemRecord &WI, ContractRecord contract)
int **AnalizePowerConstrain** (QoSGraph *powerConstrain, vector< CurrentStatus > vpgStatus, WorkItemRecord &WI, ContractRecord contract, int &pIndex)
int **AnalizeResourceReservation** (QoSGraph *powerConstrain, vector< CurrentStatus > vpgStatus, WorkItemRecord &WI, ContractRecord contract, int &pIndex)
int **AddFreeNodes** (vector< CurrentStatus > vpgStatus, int &pIndex)
int **AddStandByNodes** (vector< CurrentStatus > vpgStatus, int &pIndex)
int **GetNodesPowerConstrained** (QoSNode *node, vector< CurrentStatus > vpgStatus, AbstractNode *absNode, int &pIndex, bool isResourceReservation=false)
int **GetVirtualNodes** (vector< CurrentStatus > vpgStatus, AbstractNode *absNode, int n, int &pIndex)
int **GetStandByNodes** (vector< CurrentStatus > vpgStatus, AbstractNode *absNode, int n, int &pIndex)
bool **MultiplicityMaxCheck** (AbstractNode *absNode, int multipl=1)
void **CombNodes** (int i, int n, vector< CurrentStatus > &oc)
bool **IsNodeInGraph** (string resourceURI, QoSGraph *graph)

void **RemoveSharedNodes** (vector< CurrentStatus > nodeStatus, WorkItemRecord workItemList[], ContractRecord contractList[], int dimList, int wiScheduled)
void **RemoveReservedNodes** (vector< CurrentStatus > nodeStatus, WorkItemRecord workItemList[], ContractRecord contractList[], int dimList, int wiScheduled)
int **GetContract** (WorkItemRecord wi, ContractRecord contractList[], int dimList)
void **RemoveNode** (vector< CurrentStatus > nodeStatus, string resourceURI)
vector< CurrentStatus > **PingNode** ()

9.6.3 Private Attributes

float **_confMin**
float **_confMax**
vector< CurrentStatus > **_confNodes**
map< float, vector< CurrentStatus > > **_confs**
map< float, vector< CurrentStatus > >::iterator **_confs_iter**
map< int, int > **_currentlyMulteplicity**
SoftwareElement * **_softElem**
QoSGraph * **_physicalGraph**
VPGMasterProxy_Soap * **_vpgProxy**
string **_componentURI**

9.6.4 Constructor & Destructor Documentation

9.6.4.1 scheduler_module::WIScheduler::WIScheduler ()

9.6.4.2 scheduler_module::WIScheduler::~WIScheduler ()

9.6.5 Member Function Documentation

9.6.5.1 int scheduler_module::WIScheduler::AddFreeNodes (vector< CurrentStatus > *vpgStatus*, int & *pIndex*) [private]

Complete the physical graph with physical nodes derived by abstract nodes free from comntract options

9.6.5.2 int scheduler_module::WIScheduler::AddStandByNodes (vector< CurrentStatus > *vpgStatus*, int & *pIndex*) [private]

Add nodes to physical Graph that are execute in a stamb-by mode: it is necessary for the CCMAP/FARM skeleton

9.6.5.3 int scheduler_module::WIScheduler::AnalizePowerConstrain (QoSGraph * *powerConstrain*, vector< CurrentStatus > *vpgStatus*, WorkItemRecord & *WI*, ContractRecord *contract*, int & *pIndex*) [private]

Analizes the "Power constraint" option and update the phisical Graph of the component to satisfy this option

9.6.5.4 int scheduler_module::WIScheduler::AnalizeResourceReservation (QoSGraph * *powerConstrain*, vector< CurrentStatus > *vpgStatus*, WorkItemRecord & *WI*, ContractRecord *contract*, int & *pIndex*) [private]

Analizes the "Power constraint" option and update the phisical Graph of the component to satisfy this option

9.6.5.5 void scheduler_module::WIScheduler::CombNodes (int *i*, int *n*, vector< CurrentStatus > & *oc*) [private]

Create combinations of node list that satisfy a constraint on the power of nodes

9.6.5.6 bool scheduler_module::WIScheduler::Fini ()

Fini the class. It Releases resources

9.6.5.7 int scheduler_module::WIScheduler::GetContract (WorkItemRecord *wi*, ContractRecord *contractList*[], int *dimList*) [private]

9.6.5.8 int scheduler_module::WIScheduler::GetNextIndex (WorkItemRecord *WorkItemList*[], int *dimList*) [private]

Get the index of next WorkItem to dispatch

9.6.5.9 int scheduler_module::WIScheduler::GetNodesPowerConstrained (QoSNode * *node*, vector< CurrentStatus > *vpgStatus*, AbstractNode * *absNode*, int & *pIndex*, bool *isResourceReservation* = false) [private]

Determines a set of physical nodes that satisfy the option

9.6.5.10 int scheduler_module::WIScheduler::GetStandByNodes (vector< CurrentStatus > *vpgStatus*, AbstractNode * *absNode*, int *n*, int & *pIndex*) [private]

Determines a set of physical nodes candidate for stand-by node of the physical graph

9.6.5.11 int scheduler_module::WIScheduler::GetVirtualNodes (vector< CurrentStatus > *vpgStatus*, AbstractNode * *absNode*, int *n*, int & *pIndex*) [private]

9.6.5.12 bool scheduler_module::WIScheduler::Init (string *serverURI*)

9.6.5.13 bool scheduler_module::WIScheduler::IsNodeInGraph (string *resourceURI*, QoSGraph * *graph*) [private]

Verify if a node already belong to a graph

9.6.5.14 bool scheduler_module::WIScheduler::MultiplicityMaxCheck (AbstractNode * *absNode*, int *multipl* = 1) [private]

Check if the multiplicity of an abstract node is satisfied

9.6.5.15 vector<CurrentStatus> scheduler_module::WIScheduler::PingNode () [private]

9.6.5.16 void scheduler_module::WIScheduler::RemoveNode (vector< CurrentStatus > *nodeStatus*, string *resourceURI*) [private]

9.6.5.17 void scheduler_module::WIScheduler::RemoveReservedNodes (vector< CurrentStatus > *nodeStatus*, WorkItemRecord *workItemList*[], ContractRecord *contractList*[], int *dimList*, int *wiScheduled*) [private]

9.6.5.18 void scheduler_module::WIScheduler::RemoveSharedNodes (vector< CurrentStatus > *nodeStatus*, WorkItemRecord *workItemList*[], ContractRecord *contractList*[], int *dimList*, int *wiScheduled*) [private]

9.6.5.19 int scheduler_module::WIScheduler::ScheduleNextWI (WorkItemRecord *WorkItemList*[], ContractRecord *ContractList*[], int *dimList*, WorkItemRecord & *WI*)

Schedule the next Work Item from the WotkItem list. It modify the WI description in the list

9.6.5.19.1 Parameters:

WorkItemList the list of WorkItem
WorkItem description of dispatched WorkItem
dimList lencht of the list
dimWI lenght of a WorkItem

9.6.5.19.2 Returns:

the position of the WorkItem Dispatched, otherwise: -1. No items to dispatch -2. It's impossible to find the contract of scheduled WI -3. Error during contract option handling -4. Error during VPG communication -5. Error during component description handling -6. not enough resources are available -7. contract not valid

9.6.5.20 int scheduler_module::WIScheduler::SetMapping (QoSGraph * *resourceReservation*, QoSGraph * *powerConstrain*, vector< CurrentStatus > *vpgStatus*, WorkItemRecord & *WI*, ContractRecord *contract*) [private]

Set the mapping of the abstract graph of the component and create the physical graph on the basis of availlables resources on the VPG

9.6.6 Member Data Documentation

9.6.6.1 string scheduler_module::WIScheduler::_componentURI [private]

9.6.6.2 float scheduler_module::WIScheduler::_confMax [private]

9.6.6.3 float scheduler_module::WIScheduler::_confMin [private]

9.6.6.4 vector<CurrentStatus > scheduler_module::WIScheduler::_confNodes [private]

9.6.6.5 map<float, vector<CurrentStatus > > scheduler_module::WIScheduler::_confs [private]

9.6.6.6 map<float, vector<CurrentStatus > >::iterator scheduler_module::WIScheduler::_confs_iter [private]

9.6.6.7 map<int, int> scheduler_module::WIScheduler::_currentlyMulteplicity [private]

store for each abstract node the number of phisiscal node currently generated

9.6.6.8 QoSGraph* scheduler_module::WIScheduler::_physicalGraph [private]

9.6.6.9 SoftwareElement* scheduler_module::WIScheduler::_softElem [private]

9.6.6.10 VPGMasterProxy_Soap* scheduler_module::WIScheduler::_vpgProxy [private]

The documentation for this class was generated from the following file:

E:/tmp/Administrator-doxygen/**wischeduler.**

10 Appendice E: Web Service Resource Framework

Le specifiche WSRF si dividono in cinque parti e sono frutto di una operazione di refactoring delle precedenti specifiche OGSI. La tabella seguente riassume l'insieme delle specifiche, in pratica esse forniscono alcuni concetti/definizioni e alcune interfacce (Web services port type) per effettuare delle operazioni di gestione e controllo sulle Grid services.

Name	Describes
[3] WS-ResourceLifetime	Mechanisms for WS-Resource destruction, including message exchanges that allow a requestor to destroy a WS-Resource, either immediately or by using a time-based scheduled resource termination mechanism.
[4] WS-ResourceProperties	Definition of a WS-Resource, and mechanisms for retrieving, changing, and deleting WSResource properties.
[5] WS-RenewableReferences	A conventional decoration of a WS-Addressing endpoint reference with policy information needed to retrieve an updated version of an endpoint reference when it becomes invalid.
[6] WS-ServiceGroup	An interface to heterogeneous by-reference collections of Web services
[7] WS-BaseFaults	A base fault XML type for use when returning faults in a Web services message exchange.

Un punto importante è che, a differenza di OGSI, le specifiche WSRF sono applicabili a Web services generiche poiché non vengono proposte modifiche agli standard esistenti (SOAP, WSDL, UDDI). Vorrei sottolineare che questo sforzo di aderenza agli standard è stato facilitato anche dalle nuove specifiche Web Services concepite in accordo col mondo e-business (IBM, HP, Microsoft, etc). In figura 3 è riportato uno schema che indica le relazioni tra le specifiche WSRF (colore arancione) e i recenti standard Web services (colore azzurro).

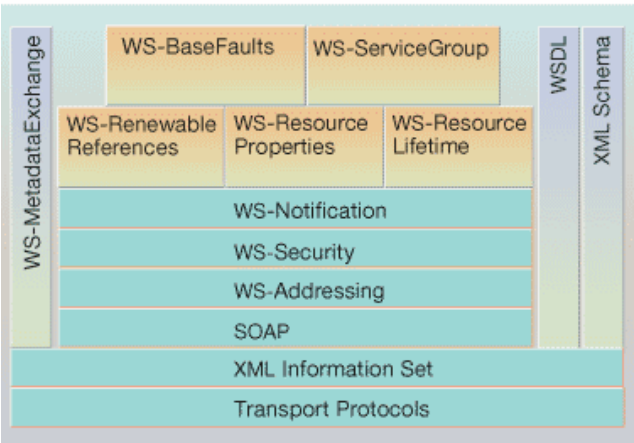

Figura 7. Specifiche WSRF e WS e loro relazione

Per riassumere le specifiche WSRF si riferiscono a particolari Web-Services che aderiscono all'implied-resource pattern, chiamate WS-Resources. In questo paragrafo riassumerò come le specifiche WSRF permettono di controllare l'esecuzione delle WS-Resources.

Le specifiche WS-ResouceLifetime di riferiscono a tre aspetti del ciclo di vita delle WS:

- creazione: è possibile creare una istanza di una WS-Resource (pattern WS-Resource Factory);
- identità: un'istanza viene identificata grazie alla proprietà Resource Properties contenuta nell'header della busta SOAP (in accordo con la specifica WS-addressing);
- distruzione: esistono due metodi di distruzione:
    - immediata: metodo Destroy();
    - scheduled: viene stabilito un tempo entro il quale se il servizio non ha finito la sua esecuzione viene distrutto.

Di seguito sono riportate alcune definizioni (prese dai vari documenti ufficiali WSRF [13][14][15][16][17]) molto utili per una migliore comprensione del documento:

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

"A stateless service implements message exchanges with no access or use of information not contained in the input message. A simple example is a service that compresses and decompresses documents, where the documents are provided in the message exchanges with the service."

"A conversational service implements a series of operations such that the result of one operation depends on a prior operation and/or prepares for a subsequent operation. The service uses each message in a logical stream of messages to determine the processing behavior of the service. The behaviour of a given operation is based on processing preceding messages in the logical sequence. Many interactive Web sites implement this pattern through use of HTTP sessions and cookies".

“A service that acts upon **stateful resources** provides access to, or manipulates a set of logical stateful resources (documents) based on messages it sends and receives. A Service that acts upon stateful resources may be described “stateless” if it delegates responsibility for the management of the state to another component such as a database or file system. Statelessness in the implementation of the service itself tends to enhance reliability and scalability: a stateless Web service can be restarted following failure without concern for its history of prior interactions, and new copies of a stateless Web service can be created (and subsequently destroyed) in response to changing load. Thus, statelessness is generally viewed as good engineering practice for Web service implementations”.

“A stateful resource is defined to:

- have a specific set of state data expressible as an XML document:
- have a well-defined lifecycle;
- and be known to, and acted upon, by one or more Web services.

Examples of system components that may be modeled as stateful resources are files in a file system, rows in a relational database, and encapsulated objects such as Entity Enterprise Java beans. Stateful resource is defined by a single XML Global Element Declaration (GED) in a given namespace. This GED defines the type of the root element of the resource’s XML document and hence the type of the stateful resource itself. When a stateful resource instance is created, it may be assigned an identity by the entity that created it. Applications using the resource may assign the resource additional identities (aliases)”.

“A **WS-Resource** is a Web service having an association with a stateful resource, where the stateful resource is defined by a resource properties document type and the association is expressed by annotating a WSDL portType with the type definition of the resource properties document”.

“We define the term **implied resource pattern** to describe a specific kind of relationship between a Web service and one or more stateful resources. A **WS-Addressing endpoint** reference is an XML serialization of a network-wide pointer to a Web service. This pointer may be returned as a result of a Web service message request to a factory to create a new resource. WS-Addressing standardizes the endpoint reference construct used to represent the address of a Web service deployed at a given network endpoint. An endpoint reference may contain, in addition to the endpoint address of the Web service, other metadata associated with the Web service such as service description information and reference properties, which help to define a contextual use of

the endpoint reference. An endpoint reference that is described as following the implied resource pattern may include a ReferenceProperties child element that identifies the stateful resource to be used in the execution of all message exchanges performed using this EndpointReference. This type of endpoint reference is referred to as a WS-Resource qualified endpoint reference.A request message directed to a Web service designated by a WS-Resource-qualified endpoint reference must include the ReferenceProperties information from the endpoint reference, as specified by WS-Addressing. Thus, the WS-Resource framework uses a WS-Resource-qualified endpoint reference to represent a "network-wide pointer" to a WS-Resource".

"A **Resource Property** is a piece of information defined as part of the state model of a WS-Resource. A resource property may reflect a part of the resource's state, meta-data, manageability information, etc.".

"A **Resource Properties Document** is the XML document representing a logical composition of resource property elements. The resource properties document defines a particular view or projection of the state data implemented by the WS-Resource. The type (e.g. the XML Schema definition of the root element) of a resource properties document is associated with the WSDL portType defining the Web service interface. This association is the basis of the WS-Resource definition. All instances of a particular WS-Resource type MUST implement a logical resource properties document of the type declared in the WSDL portType".

"A **Resource Property Element** is the XML representation of a resource property. A resource property element must appear as the immediate child of the root element of a resource properties document. A resource property element must be an XML global element definition (GED), and is uniquely identified by QName".

"A **WS-Resource factory** is any Web service capable of creating a new stateful resource, assigning the new stateful resource an identity, and creating the association between the new stateful resource and its associated Web service. The response message of a WS-Resource factory operation contains a WS-Resource-qualified endpoint reference containing a stateful resource identifier that refers to the new stateful resource".

11 Riferimenti Bibliografici

[1] S. Lombardo, A. Machì. “A model for a component based grid-aware scientific library service”. Euro-Par 2004 Parallel Processing: 10th International Euro-Par Conference, Pisa, Italy, August 31-September 3, 2004, pp. 423-428

[2] M. Aldinucci, A. Petrocelli, E. Pistoletti, M. Torquati, M. Vanneschi, L. Veraldi, and C. Zoccolo, Dynamic reconfiguration of grid-aware applications in ASSIST, in 11th Intl Euro-Par 2005: Parallel and Distributed Computing, LNCS, Lisboa, Portugal, August 2005.

[3] Workflow Management Coalition, The Workflow Reference Model (WFMC-TC-1003, 19-Jan-95, 1.1), http://www.wfmc.org

[4] GT4 WS-GRAM Documentation http://www-unix.globus.org/toolkit/docs/4.0/execution/wsgram/

[5] the CONDOR Project, http://www.cs.wisc.edu/condor/

[6] Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, Andreas Savva, Job Submission Description Language (JSDL) Specification Ali Anjomshoaa, EPCC http://forge.gridforum.org/projects/jsdl-wg Fred Brisard, CA

[7] WSRF Specification http://www-128.ibm.com/developerworks/library/ws-resource/index.html

[8] S. Lombardo, V. Graziano, A. Machì. Euristiche per il Controllo della QoS di componenti grid-enabled (modelli e patterns). Technical Report ICAR-CNR Dept. Palermo RT-ICAR-PA-05-XX Novembre 2005

[9] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, J. Dongarra. "New Grid Scheduling and Rescheduling Methods in the GrADS Project," IPDPS, p. 199a, 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 10, 2004

[10] A. Machì, F. Collura, S. Lombardo: “Dependable Execution of Workflow Activities on a Virtual Private Grid Middleware. V. S. Sunderam et als. (Eds.) Computational Science ICCS 2005 LNCS 3516 pp.267-274, 2005 Springer-Verlag 2005

[11] A. Machì, F. Collura, , S. Lombardo. “Modellazione UML dello Skeleton di coordinamento di un Componente Parallelo Master-Slave e di patterns per il controllo esterno della sua performance “ Technical Report ICAR-CNR Dept. Palermo RT-ICAR-PA-05-03 Marzo 2005

[12] RT-ICAR-WP8-INTEGRAZIONE

[13] WS-ResourceProperties http://www-128.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf

[14] WS-ResourceLifetime http://www-128.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf

[15] WS-ServiceGroup http://www-128.ibm.com/developerworks/library/ws-resource/ws-servicegroup.pdf

[16] WS-BaseFaults http://www-128.ibm.com/developerworks/library/ws-resource/ws-basefault.pdf

[17] WS-Addressing http://www.w3.org/Submission/ws-addressing/