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

Virtual Private Grid (VPG 1.2):

Un middleware a supporto del ciclo di vita e del monitoraggio dell'esecuzione grafi di processi su griglia computazionale.

Versione 1.2.1

Saverio Lombardo, Alberto Machì

RT-ICAR-PA-11 -2004 **Ottobre 2004**

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

Virtual Private Grid (VPG 1.2):

Un middleware a supporto del ciclo di vita e del monitoraggio dell'esecuzione grafi di processi su griglia computazionale.

Versione 1.2.1

Saverio Lombardo, Alberto Machì

Deliverable II° anno
Progetto MIUR FIRB Grid.it
Work Package 9 Librerie Scientifiche

Data Aggiornamento **10 Ottobre 2004**

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*

1 Sommario:

VPG è un package software sviluppato nell'ambito del progetto MIUR-FIRB "Piattaforme abilitanti per griglie computazionali a elevate prestazioni orientate a organizzazioni virtuali scalabili" GRID.IT , Workpackage Librerie Scientifiche.

Esso fornisce a sviluppatori di applicazioni distribuite che si avvalgono di reti di elaboratori connettibili in una griglia computazionale attraverso i servizi del Toolkit GLOBUS 2, una interfaccia di programmazione ad alto livello che fa apparire la griglia come fosse un cluster di nodi distribuiti, connessi virtualmente in una griglia privata (Virtual Private Grid o VPG).

Nella versione 1.2 descritta nel presente documento e' possibile installare, attivare e controllare da programma l'esecuzione di gruppi di processi sui nodi della griglia virtuale, ed essere avvertiti della evenienza di cadute di nodi, della disconnessione di canali delle rete o anche di errori software.

Il sistema è sviluppato con approccio centralizzato client-server: Un servizio VPG-master è attivato sul nodo di controllo della applicazione, un demone VPG-slave su ogni nodo. Il servizio master riceve i comandi dal controller della applicazione distribuita e utilizza i servizi dei demoni slave per pilotare il trasferimento di dati e l'attivazione di processi tramite i servizi di basso livello di Globus2 e meccanismi dei sistemi operativi locali ai nodi remoti per monitorarne e controllarne l'esecuzione.

I demoni slave utilizzano un sistema di trasmissione di eventi per informare il servizio master della loro vitalità o di errori relativi ai processi.

I servizi VPG sono accessibili al sistema di controllo anche attraverso API .

L'approccio a Macchina Virtuale permette di enucleare in un ridotto insieme di funzioni e segnali le operazioni di controllo delle esecuzione di gruppi di processi coordinati, base per il supporto al ciclo di vita di componenti paralleli e pattern di monitoraggio della Qualità del Servizio su griglia.

Nella versione 1.2 il servizio è limitato al ciclo di vita di insiemi di processi connessi da un grafo diretto (senza analisi delle dipendenze) ed al monitoraggio della disponibilità delle risorse da essi impiegate.

Primitive per il controllo del ciclo di vita di componenti (grafi di processi coordinati) ed il monitoraggio delle esecuzione degli stessi secondo specifiche non funzionali di Qualità del Servizio saranno disponibili con la versione 1.3 del prodotto .

Nella realizzazione del software si è utilizzato un processo iterativo e incrementale modellando il sistema in linguaggio UML utilizzando tecnologie ad oggetti e codificando il codice in linguaggio C++. Strumenti utilizzati durante la realizzazione sono:

- Poseidon for UML: (della Gentleware Inc.)per la modellazione e per la stesura dei diagrammi UML;
- Kdevelop: (Red Hat Linux) per l'implementazione dei package, degli eseguibili e delle applicazioni di test.

Il documento si compone di tre parti:

- Nella sezione **Analisi dei requisiti**: vengono analizzati i requisiti del prodotto con attenta analisi e formulazione del problema;
- Nella sezione **Architettura del sistema**: viene presentata l'architettura generale del sistema, l'insieme dei package sviluppati e le teconologie utilizzate nell'implementazione del prototipo del prodotto;
- Nella sezione **Progetto esecutivo:** vengono descritte e documentate le principali classi sviluppate in linguaggio C++ e riportati diagrammi che illustrano le relazioni tra di esse;

2 Indice

3 Analisi dei requisiti
(Requirements Analysis Document)

Versione:

Versione 0.1, data creazione: 10 Ottobre 2004;

Prefazione:

Questo documento riporta i requisiti del sistema proposto.

3.1 Introduzione

Le Macchine Virtuali, nate a livello di meta-sistema operativo, semplificano di molto il calcolo distribuito mettendo a disposizione di fornitori di risorse ed utenti funzionalità di attivazione e controllo su rete ad un livello di astrazione che permette di prescindere dalla effettiva complessità e dai particolari della gestione della attivazione dei comandi di sistema operativo sul singolo nodo e delle comunicazioni sulla rete e del loro monitoraggio.

Nel caso del calcolo parallelo esse permettono inoltre ad una collezione eterogenea di computer con sistemi operativi differenti e collegati insieme da un network, di operare come un singolo computer parallelo (es. PVM [1]) o ad un sistema di gestione delle risorse di effettuare una efficace mappatura di un grafo di processi virtuali sulle risorse fisiche disponibili e di effettuare il loro coordinamento. Un semplice esempio di un tale uso è la CLAM [3] (Coordination Language Abstract Machine), utilizzata nell'ambiente ASSIST a supporto dei pattern di coordinamento di programmazione parallela strutturata (skeletons).

La Macchina Virtuale può essere resa disponibile al processo di controllo che gestisce la interazione fra i processi di elaborazione che costituiscono il grafo di una applicazione parallela (program-level management) come supporto a run-time . Alternativamente essa può costituire uno strato di middleware distinto a supporto di un sistema indipendente di work-flow-management che operi come sistema operativo distribuito (system level management). Tali sistemi si curano generalmente anche del monitoraggio del rispetto di specifiche non funzionali della applicazione come la Qualità del Servizio (sicurezza, affidabilità, performance).

Nel contesto delle griglie computazionali recenti studi hanno investigato l'utilizzo di macchine virtuali classiche (a livello di sistema operativo) per la personalizzazione e l'incapsulamento di interi ambienti applicativi (es legacy) su griglia al fine di permettere la clonazione degli stessi su risorse distribuite, ad esempio a supporto delle operazioni di recover da checkpoint [5].

L'uso di macchine virtuali a supporto dell' enactment di jobs complessi e delle operazioni di workflow enforcement successive al rilievo di condizioni anomale su risorse di griglia non ha ricevuta sufficiente attenzione. Ciò probabilmente in quanto il monitoring delle stato delle risorse è stato considerato parte del sistema informativo di griglia (Globus MDS [4], R-GMA [5]) ed a causa del diffuso approccio application-oriented al controllo della performance GRADS [6].

Riteniamo invece che una Macchina Virtuale di griglia in grado di fornire un insieme coerente di primitive essenziali per il deployment,l'attivazione, il monitoraggio della Qualità della

esecuzione e la riorganizzazione del grafo in conseguenza di eventi notevoli sulle risorse di griglia possa risultare estremamente utile allo sviluppo di efficienti pattern di management (sia a livello di applicazione che di sistema) di istanze di work-flow e/o componenti paralleli su griglia .

3.2 Il Sistema Proposto

Il package VPG fornisce a sviluppatori di applicazioni distribuite che si avvalgono di reti di elaboratori connettibili in una griglia computazionale attraverso i servizi del Toolkit GLOBUS 2, una interfaccia di programmazione ad alto livello che fa apparire la griglia come fosse un cluster di nodi distribuiti, connessi virtualmente in una griglia privata. E' possibile montare e smontare nodi, installare, attivare e controllare da programma l'esecuzione di gruppi di processi sui nodi della griglia virtuale, ed monitorare in polling lo stato dei processi e le evenienza di cadute di nodi, della disconnessione di canali delle rete o anche di errori software segnalati a cura del codice utente.

Il sistema è sviluppato con approccio centralizzato client-server: Un servizio VPG-master è attivato sul nodo di controllo della applicazione, un demone VPG-slave su ogni nodo. Il servizio master riceve i comandi dal controller della applicazione distribuita e utilizza i servizi dei demoni slave per pilotare il trasferimento di dati e l'attivazione di processi tramite i servizi di basso livello di Globus2 e meccanismi dei sistemi operativi sui nodi remoti per monitorarne e controllarne l'esecuzione.

Per effettuare il check dei nodi della griglia privata il Master emette, ad intervalli di tempo regolari, un evento di mantenimento del sistema in vita (keep-alive) cui i nodi attivi rispondono con un segnale di acknoledge (heart-beat). La mancata ricezione di tale segnale viene considerato dal master come un evento di caduta del nodo. La mancata ricezione dell'evento di keep-alive provoca l'auto spegnimento del servizio VPG Slave.

Per effettuare il controllo delle connessioni significative i nodi slave, configuarati come appartenenti ad un grafo, ad intervalli regolari effettuano il polling dei nodi primi vicini attivandone la funzionalità di ping. La mancata risposta al comando viene comunicata al master attraverso un evento interpretato dal master di connection fault.

La Remote Engine comunica al Master la terminazione normale o abnorme dei processi gestiti sul proprio nodo attraverso un evento di fine processo. Segnali di avanzamento o stato del processo (checkpoint) possono essere inviati dal codice utente al VPGMaster utilizzando le API dell'interfaccia ad Eventi.

3.3 Lo stato dei processi è disponibile alle applicazioni client tramite l'interfaccia funzionale del VPGMaster.

3.3.1 Requisiti funzionali

Di seguito sono riportate le funzionalità del sistema:

- esposizione di un insieme di primitive utili per il deployment, l'attivazione, il monitoraggio e la riorganizzazione di un grafo di processi (per un maggiore approfondimento si rimanda alla documentazione delle API):
    - deployment di un grafo di processi con indicazione per ognuno di essi della lista dei nodi su cui effettuare lo staging degli eseguibili e delle librerie richieste e la lista dei primi vicini connessi dalla applicazione;
    - rimozione di un grafo di processi precedentemente installati;
    - attivazione di un grafo di processi con l'indicazione per ognuno di essi del nodo fisico di esecuzione, (ad ogni processo del grafo avviato viene attribuito un indice simbolico)
    - interruzione di alcuni processi precedentemente attivati individuati attraverso i propri indici simbolici;
    - rilancio di alcuni processi precedentemente interrotti, con possibilità di attivazione su nodi differenti dalla precedente esecuzione;
    - informazioni sullo stato del ciclo di vita (esecuzione, done, abort) di un grafo di processi precedentemente attivati;
- esposizione di un insieme di primitive utili per l'acquisizione/rilascio di nodi computazionali (vedi di seguito documentazione delle API):
    - aggiunta di un insieme di nodi alla VPG;
    - rimozione di un insieme di nodi dalla VPG;
    - informazioni sullo stato dei nodi;
- esposizione di un servizio per l'emissione/ricezione di eventi. Tale servizio permette di:
    - registrarsi per l'emissione/ricezione di particolari eventi;
    - emettere eventi;

- ricevere eventi di proprio interesse;

- test di connettività tra alcuni nodi montati sulla VPG. Tale test indica la presenza di connettività tra le porte delle Remote Engine e non garantisce la connettività su altre porte;

3.3.1 Requisiti non funzionali

Il sistema software si interfaccia verso l'esterno attraverso meccanismi RPC (Remote Procedure Call). Per interfacciarsi col sistema bisogna dunque svilupparsi degli appositi proxy che permettano la comunicazione con il VPGMaster, che costituisce il front-end di tutta la piattaforma.

3.3.1.1 Documentazione

Durante la progettazione del sistema saranno prodotti e rilasciati i seguenti documenti.

Documento	Utenti del documento
Documento di analisi dei requisiti (RAD) Capitolo 3 del presente documento	Sviluppatori, cliente
Documento dell'architettura software (SDD) Capitolo 4 del presente documento	Sviluppatori, analisti
Documento del progetto esecutivo (ODD) Capitolo 5 del presente documento	Sviluppatori

3.3.1.2 Considerazioni Hardware

Il software non richiede dispositivi di interfacciamento particolari nelle macchine dell'utente che intende usufruire dei servizi offerti dal sistema. Esso necessita solamente di un sistema di connessione intranet/internet in modo da permettere una connessione remota verso il VPGMaster.

Le macchine abilitate ad essere montate sulla VPG invece devono possedere un indirizzo IP statico e un certo numero di porte TCP aperte per permettere la comunicazione tra le remote engine e col master del sistema.

3.3.1.1 Prestazioni Hardware

Non sono considerate necessarie prestazioni particolari per le macchine utilizzate poiché i componenti facenti parte del sistema (VPGMaster e Remote Engine) non presentano caratteristiche particolari. L'unico requisito attualmente è dettato degli strumenti utilizzati per implementare alcune funzionalità, cioè Globus Toolkit 2.

3.3.1.2 Interfaccia del sistema

Il software proposto impone l'utilizzo di XML-RPC come linguaggio di comunicazione col fron-end del sistema, ciò comporta l'utilizzo di tale linguaggio per effettuare delle richieste al servizio.

3.3.2 Vincoli: pseudo-requisiti

I vincoli attuali del sistema sono legati solamente ai nodi che si vogliono montare sulla VPG e sul nodo in cui si vuole installare il front-end del sistema, non sono invece imposti vincoli sulle macchine in cui risiedono le applicazioni client del sistema.

Affinchè un nodo possa essere utilizzato e montato sulla VPG devono essere installati i seguenti servizi di Globus Toolkit 2:

- **GRAM**: per l'avvio della "Remote Engine" sul nodo (mount del nodo);
- **GridFTP** : staging degli eseguibili/dati sugli host della VPG (anche della Remote Engine);

Il nodo su cui si vuole installare il VPGMaster deve necessariamente disporre dell'installazione del pacchetto Globus Toolkit 2.

Inoltre su tutte le macchine deve essere eseguita una corretta configurazione del sistema di sicurezza di globus, ciò significa che:

- nella macchina dove viene installato il VPGMaster deve essere installato un utente globus con un certificato digitale valido;
- le macchine da montare sulla VPG devono abilitare tale utente globus ad eseguire operazioni di griglia.

3.3.3 Upgrade alla versione corrente del software (1.2) in corso di sviluppo

Al fine di rendere più agevole lo sviluppo di applicazioni client è in corso la realizzazione di una libreria per gestire la serializzazione/deserializzazione dei comandi e per costruire dei proxy personalizzati. Si rimanda alla documentazione del progetto esecutivo per una migliore comprensione di come poter sviluppare una applicazione client.

Al fine di facilitare l'utilizzo delle API sono in stesura programmi applicativi di esempio e test delle funzionalità.

3.3.4 Modelli del sistema

Per spiegare il modello del sistema vengono riportati dei diagramma dei casi d'uso che mostrano le funzionalità principali del sistema e alcuni diagrammi di sequenza che illustrano alcuni scenari tipici di utilizzo del sistema software.

3.3.4.1 Modelli dei Casi D'uso

Di seguito viene riportato un primo diagramma che fornisce una vista d'insieme delle funzionalità offerte dal sistema proposto e da due diagrammi che mostrano le funzionalità implementate dal front-end del sistema e da ogni demone lanciato sui nodi acquisiti sulla griglia.

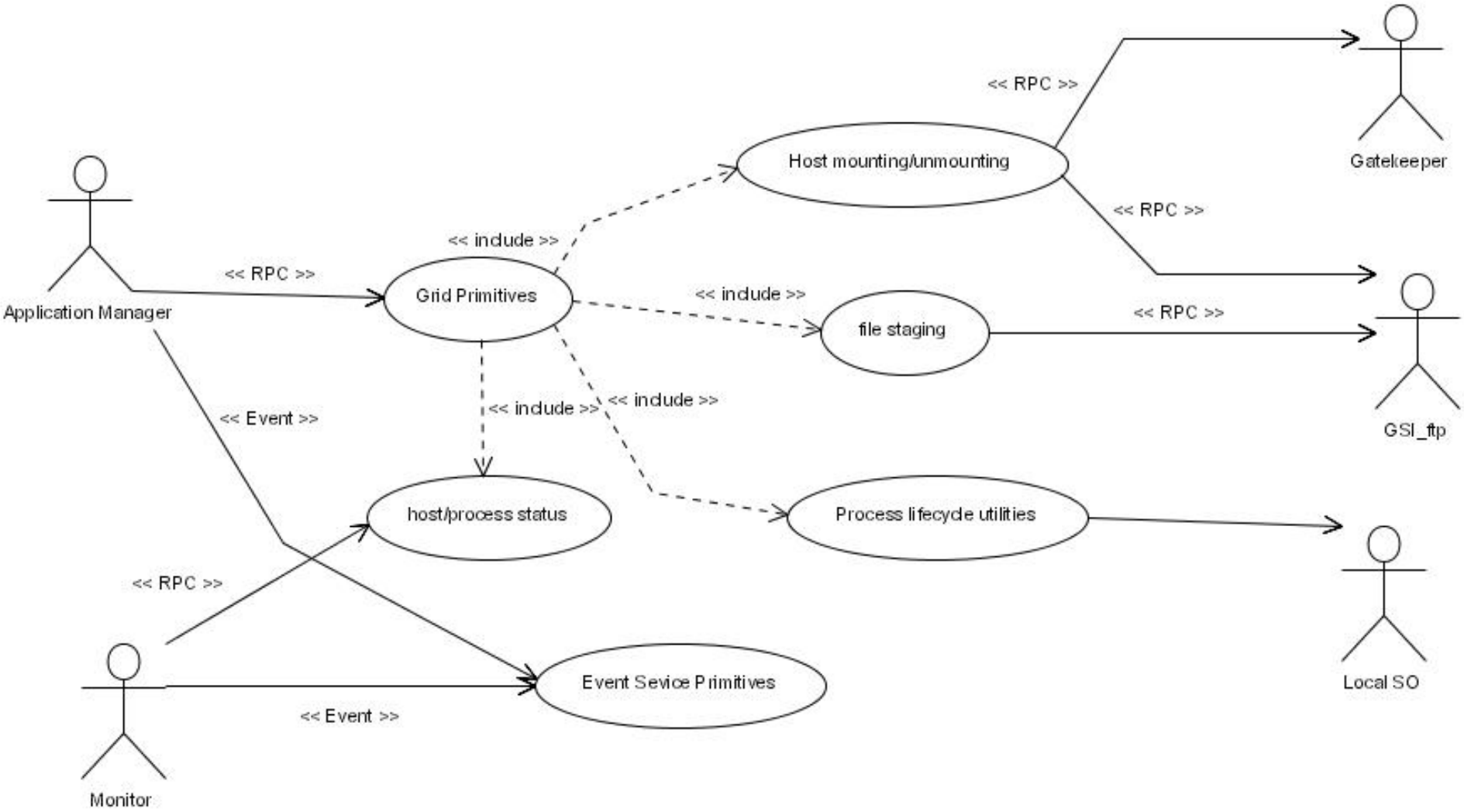
3.3.4.1.1 Attori

Nella tebella seguente sono riassunti tutti gli attori coinvolti nell'utilizzo del sistema software proposto.

Nome	Descrizione
VPG Master	Front-end del sistema software, presenta una porta per fornire le proprie funzionalità e una porta che fornisce un servizio ad eventi (iscrizione, cancellazione, invio/ricezione)
Gatekeeper Server	Servizio di Globus Toolkit 2 utile per avviare l'esecuzione remota di processi. Utilizzato per l'avvio della Remote Engine nei nodi montati sulla VPG.
GSI ftp Server	Servizio Globus Toolkit 2 utile per effettuare lo staging dei file nei vari nodi della griglia.
Remote Engine	Demone lanciato su ogni nodo acquisito sulla VPG. Presenta una interfaccia per ricevere tutti i comandi per gestire il nodo locale da parte del VPG Master e una interfaccia per l'invio e la ricezione di eventi.
Application Manager	Elemento Software capace di gestire grafi di processi in base a delle regole stabilite dalla propria logica di management.

Monitor	Elemento Software il cui compito è quello di monitorare sullo stato di avanzamento delle esecuzioni al fine di verificare il rispetto della qualità del servizio dei processi lanciati sulla VPG. Possiede delle regole/politiche interne per stabilire le operazioni da fare in caso di mancato rispetto della suddetta Qualità del Servizio.
Local SO	Sistema operativo locale ad ogni nodo montato. Fornisce tutti i meccanismi per lanciare/uccidere un processo

3.3.4.1.2 Diagramma generale



3.3.4.1.2.1 Casi d'uso del diagramma

Nome del caso d'uso	Grid Primitives
Attori coinvolti	Invocato dall'Application Manager per usufruire di un insieme di primitive.
Entry condition	L'Application Manager necessita di svolgere delle operazioni di staging, attivazione di graphi di processi o modificare l'insieme dei nodi montati sulla VPG.
Descrizione	Tramite l'interfaccia del front-end della VPG è possibile effettuare delle operazioni per: • il deployment, l'attivazione, il monitoraggio e la riorganizzazione di un grafo di processi • aggiungere/togliere nodi alla VPG
Exit condition	Comando eseguito

Nome del caso d'uso	**Host mounting/Unmounting**
Attori coinvolti	Comunica con: • GSI_ftp: per effettuare lo staging dei file della Remote Engine sul nodo da montare; • Gatekeeper: per avviare la Remote Engine;
Entry condition	Necessita di modificare l'insieme dei nodi della VPG
Descrizione	Il sistema offre un insieme di primitive per aggiungere e togliere nodi dalla VPG. , l'aggiunta di un nodo comporta l'installazione e l'esecuzione presso il nodo della Remote Engine. L'eliminazione del nodo comporta la "pulitura" di tutti i processi installati su di esso.
Exit condition	Nodo aggiunto/rimosso con successo o codice di errore

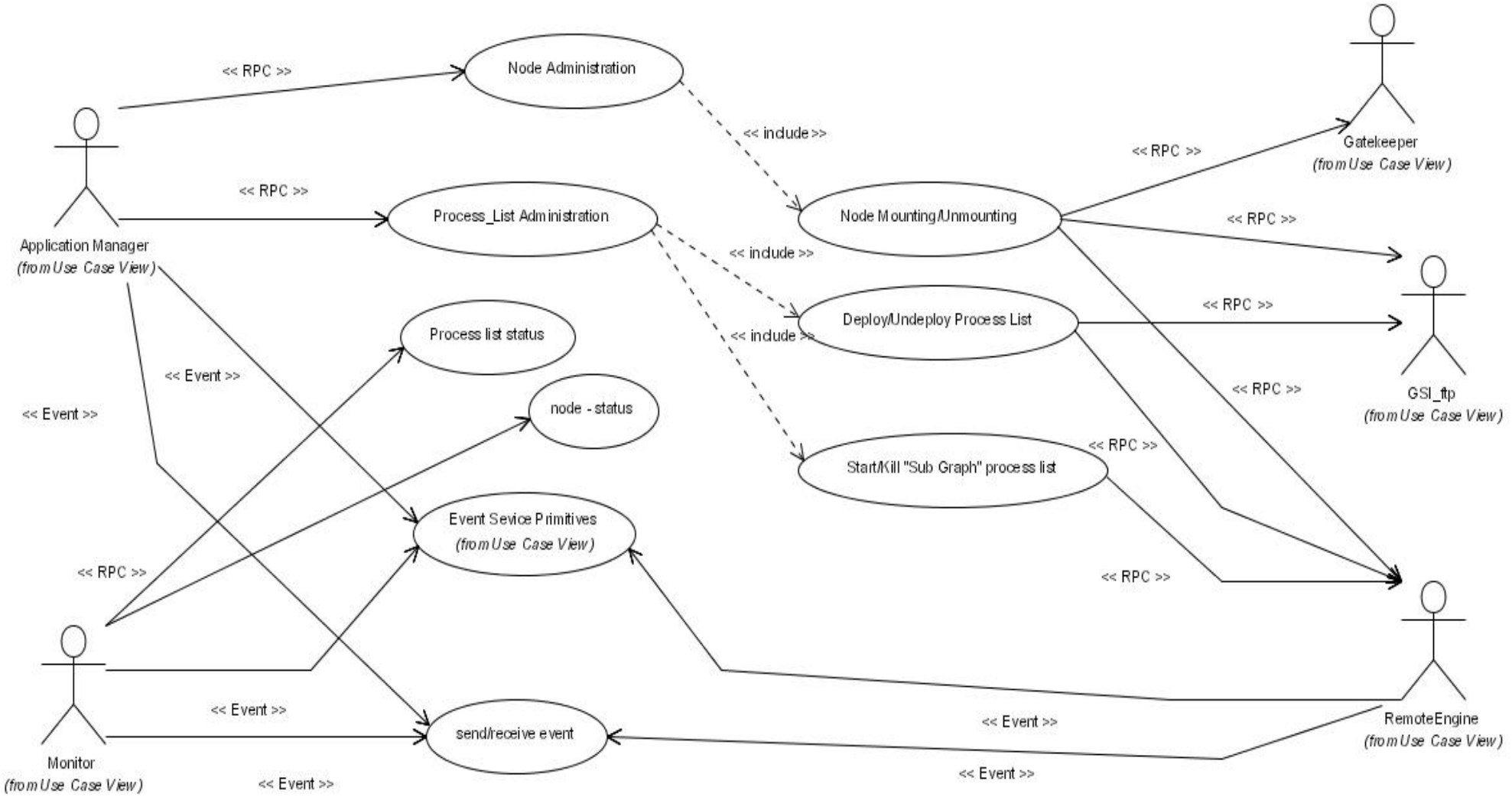
Nome del caso d'uso	**Host/Process Status**
Attori coinvolti	Invocato da un monitor o da un application manager al fine di poter avere informazine sullo stato delle risorse
Entry condition	Polling sullo stato delle risorse (nodi e/o grafi di processi in esecuzione)
Descrizione	Il front-end del sistema possiede un registro che contiene lo stato di esecuzione di tutti i processi lanciati e lo stato dei nodi montati sulla VPG.
Exit condition	Informazione di stato sul grafo di processi o sui nodi della VPG.

Nome del caso d'uso	File staging
Attori coinvolti	Comunica con: GSI ftp Server: per lo staging dei file
Entry condition	Si è richiesto un comando di montaggio di un grafo di processi sulla VPG.
Descrizione	Funzionalità utile per effettuare il deployment del grafo dei processi sulla VPG, ogni grafo di processi puo' essere montato in maniera ridondante, cioè su tutti i nodi della VPG oppure è possibile specificare per ogni processo del grafo l'insieme di nodi su cui fare il deployment
Exit condition	Processi installati con successo o codice di errore

Nome del caso d'uso	Process lifecycle utilities
Attori coinvolti	Comunica con il Sistema operativo locale per poter attivare/uccidere i processi del grafo
Entry condition	Si è richiesto un comando per lanciare /uccidere un grafo di processi o un processo singolo
Descrizione	Permette di lanciare un grafo di processi sulla VPG specificando il mapping fisico, cioè quale processo lanciare in quale nodo. Una volta lanciati un grafo di processi è possibile ucciderne alcuni per rilanciarli sullo stesso nodo di partenza o su nodi differenti.
Exit condition	Processi lanciati/uccisi con successo o codice di errore ritornato

Nome del caso d'uso	Process event primitive
Attori coinvolti	Invocato dall'Application Managero e/o dal Monitor per iscriversi al servizio ad eventi e per emettere/ricevere degli eventi;
Entry condition	Un elemento Software (interno al sistema o attore esterno) è interessato a ricevere/emettere eventi
Descrizione	Servizio di eventi per poter ricevere/emettere eventi. Ogni elemento software che è interessato ad emettere/ricevere eventi deve dapprima registrarsi, in seguito alla registrazione tale elemento è abilitato ad emettere eventi e a ricevere eventi di proprio interesse.
Exit condition	Codice di ritorno

3.3.4.1.3 Front-End del sistema (VPGMaster)



3.3.4.1.4 Casi D'uso del diagramma

Nome del caso d'uso	Node Administration
Attori coinvolti	Invocato dall'Application Manager: per poter modificare l'insieme di nodi appartenenti alla VPG;
Entry condition	Necessità di modificare l'insieme di nodi della VPG;
Descrizione	L'insieme dei nodi acquisiti e facenti parte della VPG può essere cambiato dinamicamente attraverso delle operazioni di aggiunta ed eliminazione.
Exit condition	Operazione eseguita con successo o codice di errore ritornato

Nome del caso d'uso	Node Status
Attori coinvolti	Invocato dal monitor per monitorare tramite polling lo stato dei nodi della VPG;
Entry condition	Monitoraggio sullo stato dei nodi acquisiti;
Descrizione	Il VPG Master contiene un registro interno in cui è memorizzato lo stato di ogni nodo montato sulla griglia. Tale registro è aggiornato costantemente attraverso gli eventi lanciati dalle Remote Engine (VPG_SHB). E' permesso dunque l'interrogazione di tale registro per avere informazioni sullo stato globale della VPG.
Exit condition	Stato della VPG o codice di errore ritornato.

Nome del caso d'uso	Node Mounting/Unmonting
Attori coinvolti	Comunica con: • GSI_ftp: per effettuare lo staging dei file della Remote Engine; • Gatekeeper: per avviare la Remote Engine sul nodo montato;
Entry condition	• Eliminazione di nodi non più necessari; • Aggiunta di nuovi nodi;
Descrizione	L'insieme dei nodi acquisiti e facenti parte della VPG può essere cambiato dinamicamente attraverso delle operazioni di aggiunta ed eliminazione. Ad ogni nodo aggiunto alla VPG viene installato il componente Remote Engine per permettere l'esecuzione di primitive remote sul nodo montato.
Exit condition	Codice di ritorno

Nome del caso d'uso	Process_List Administration
Attori coinvolti	Invocato dall'Application Manager per il deployment, l'attivazione, il monitoraggio e la riorganizzazione di un grafo di processi
Entry condition	Enacment di un grafo di processi sulla VPG
Descrizione	Il sistema fornisce una serie di primitive che permette l'enactment e operazioni basilari di enforcement per l'esecuzione di un grafo di processi sulla VPG.
Exit condition	Codice di ritorno

Nome del caso d'uso	Deploy/Undeploy Process List
Attori coinvolti	Invocato dall'Application Manager per il deployment di un grafo di processi sulla VPG
Entry condition	deployment di un grafo di processi sulla VPG
Descrizione	Il sistema fornisce una serie di primitive che permette il deployment

	di un grafo di processi sulla VPG, ogni grafo di processi puo' essere montato in maniera ridondante, cioè su tutti i nodi della VPG oppure è possibile specificare per ogni processo del grafo l'insieme di nodi.
Exit condition	Codice di ritorno

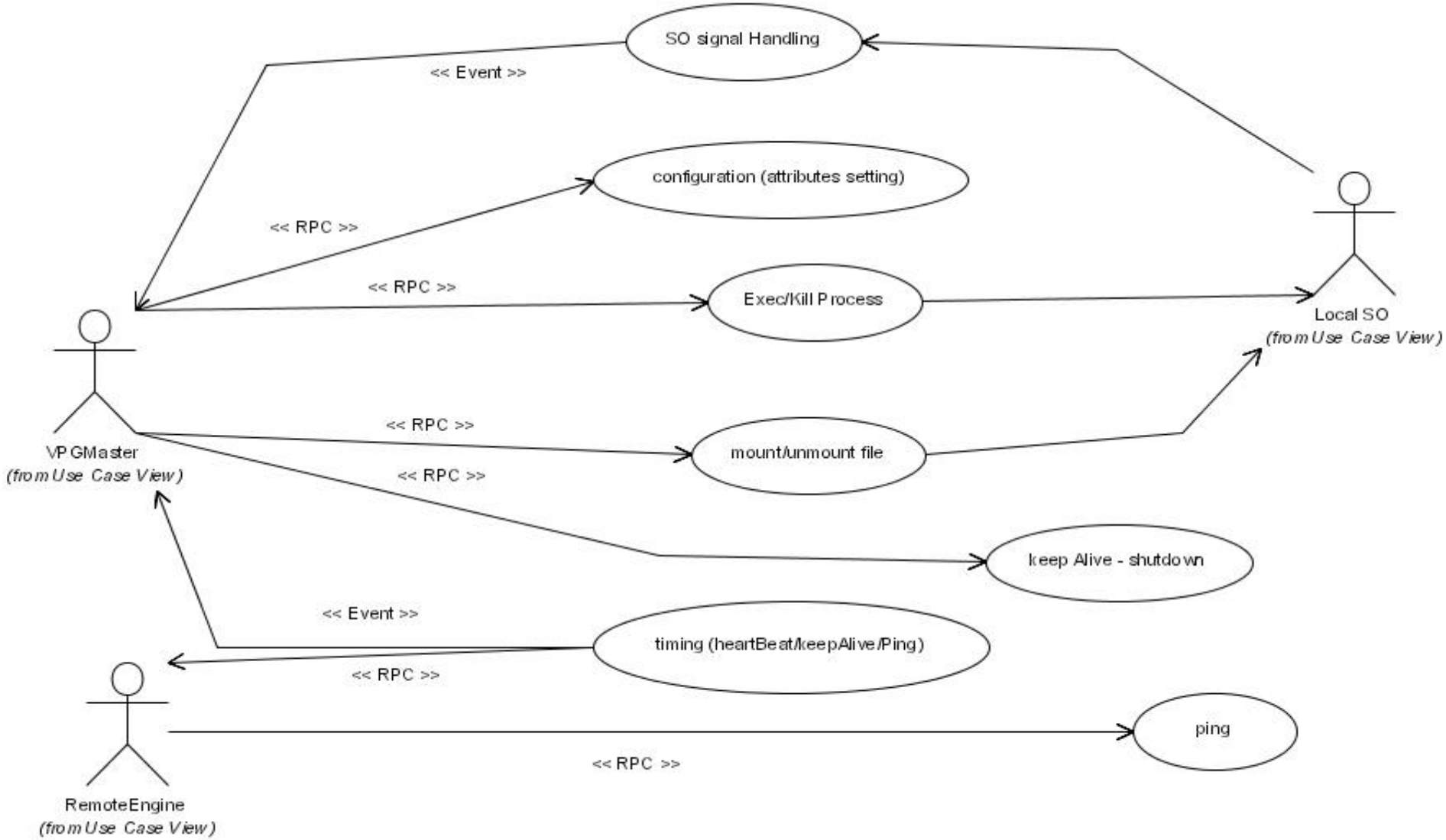
Nome del caso d'uso	Start/Kill Process List
Attori coinvolti	Comunica le Remtote Engine del nodo per attivare/uccidere i processi.
Entry condition	Enactment di un grafo (o sottografo) di processi sulla VPG
Descrizione	Il sistema fornisce una serie di primitive che permette l'attivazione di un grafo di processi sulla VPG con l'indicazione per ognuno di essi del nodo fisico di esecuzione. Ad ogni processo del grafo avviato viene attribuito un indice simbolico in modo da poter uccidere e riavviare un processo indicandone il solo indice simbolico.
Exit condition	Codice di ritorno

Nome del caso d'uso	Process Status
Attori coinvolti	Invocato dal monitor per monitorare tramite polling sullo stato di esecuzione dei processi lanciati;
Entry condition	Monitoraggio sullo stato di un grafo di processi.
Descrizione	Il VPG Master contiene un registro interno in cui è memorizzato lo stato di esecuzione di ogni processo attivato. Tale registro è aggiornato costantemente attraverso gli eventi lanciati dalle Remote Engine (VPG_PEND, VPG_PFLT). E' permesso dunque l'interrogazione di tale registro per avere informazioni sullo stato dei processi di un grafo
Exit condition	Stato di esecuzione su ogni processo del grafo

Nome del caso d'uso	**Event Service Registration**
Attori coinvolti	Invocato dall'Application Managero e/o dal Monitor per iscriversi al servizio ad eventi e per emettere/ricevere degli eventi;
Entry condition	Un elemento Software (interno al sistema o attore esterno) è interessato a ricevere/emettere eventi
Descrizione	Servizio di eventi per poter ricevere/emettere eventi. Ogni elemento software che è interessato ad emettere/ricevere eventi deve dapprima registrarsi, in seguito alla registrazione tale elemento è abilitato ad emettere eventi e a ricevere eventi di proprio interesse.
Exit condition	Codice di ritorno

Nome del caso d'uso	Send/Receive Event
Attori coinvolti	Invocato dall'Application Manager, Monitor, Remote Engine per per emettere/ricevere degli eventi;
Entry condition	Un elemento Software (interno al sistema o attore esterno) emette un evento
Descrizione	Il VPGMaster posside una porta provide che permette di poter ricevere/emettere eventi. Ogni elemento software che è interessato ad emettere/ricevere eventi deve dapprima registrarsi, in seguito alla registrazione tale elemento è abilitato ad emettere eventi e a ricevere eventi di proprio interesse.
Exit condition	Evento emesso

3.3.4.1.5 Remote Engine



3.3.4.1.6 Casi D'uso del diagramma

Nome del caso d'uso	Exe/Kill Process
Attori coinvolti	Attivato dal VPG Master per attivare un processo o per cambiarne lo stato di esecuzione
Entry condition	Attivazione di un processo
Descrizione	La Remote Engine pemette di attivare/uccidere dei processi in precedenza montati permettendo di specificare: • directory di lavoro; • parametri di avvio (argv); • variabili di ambiente;
Exit condition	Processo avviato/ucciso o codice di errore di ritorno

Nome del caso d'uso	Configuration
Attori coinvolti	Attivato dal VPG Master per effettuare una (ri)configurazione della Remote Engine
Entry condition	Riconfigurazione della remote engine
Descrizione	La Remote Engine presenta i seguenti attributi il cui valore è configurato all'avvio del componente e può essere modificato durante l'esecuzione dello stesso attraverso dei metodi "set" presenti nell'interfaccia della Remote Engine: • numero di porta provide; • URI della porta ad eventi del master; • time-out per il keep-alive, hartbeat, ping dei nodi connessi
Exit condition	Setting del parametro effettuato

Nome del caso d'uso	Mount/Unmonut file
Attori coinvolti	Attivato dal VPG Master per comunicare lo staging di file o per eliminare file sul nodo
Entry condition	Staging di un file, eliminazione i un file
Descrizione	Ogni operazione di staging di un file su un nodo deve essere comunicato alla Remote Engine in modo da poter settare i corretti permessi di esecuzione (nel caso di processi) o per tenere traccia dei file montati sul nodo (informazione utile per le operazioni di Garbage collection).
Exit condition	Processo montato/eliminato con successo o codice di errore di ritorno

Nome del caso d'uso	Keep-Alive/Shutdown
Attori coinvolti	Attivato dal VPG Master mantenere in vita la Remote Engine o per effettuare lo shutdown pr smontare il nodo dalla VPG
Entry condition	Mantenere in vita il nodo
Descrizione	La Remote Engine rimane in vita solamente se soggetta ad un meccanismo di keep-alive,cioè se riceve periodicamente un comando da parte del master che indica di rimanere in vita. Un time-out presso la tiene traccia di tale comando e in caso di mancata ricezione si attiva un processo di auto-shutdown.
Exit condition	

Nome del caso d'uso	**Timing**
Attori coinvolti	Comunica con • VPGMaster per inviare evento di heartbeat • RemoteEngine: per effettuare un ping su una connessione tra Remote Engine
Entry condition	Time-out scaduto
Descrizione	La Remote Engine possiede un triplice timer necessario per i meccanismi di: • Heartbeat: emette un evento verso il Master per indicare che il nodo è ancora vivo; • Keep-Alive: necessario per verificare se il comando di keep-Alive è arrivato o meno nell'ultimo quanto di tempo • Neighbors: quanto di tempo per temporizzate la verifica delle connessioni con i nodi "vicini". L'insieme di tali nodi vengono stabiliti dal master
Exit condition	Timing scaduto

Nome del caso d'uso	Ping
Attori coinvolti	Attivato dalla Remote Engine di un nodo vicino per testare la connessione
Entry condition	Time-out di ping di connesione

Descrizione	La Remote Engine possiede un meccanismo per stabilire la connettività tra due nodi adiacenti. Se tale connettività viene meno allora viene emesso un evento verso il master comunicando l'accaduto
Exit condition	Ping effettuato

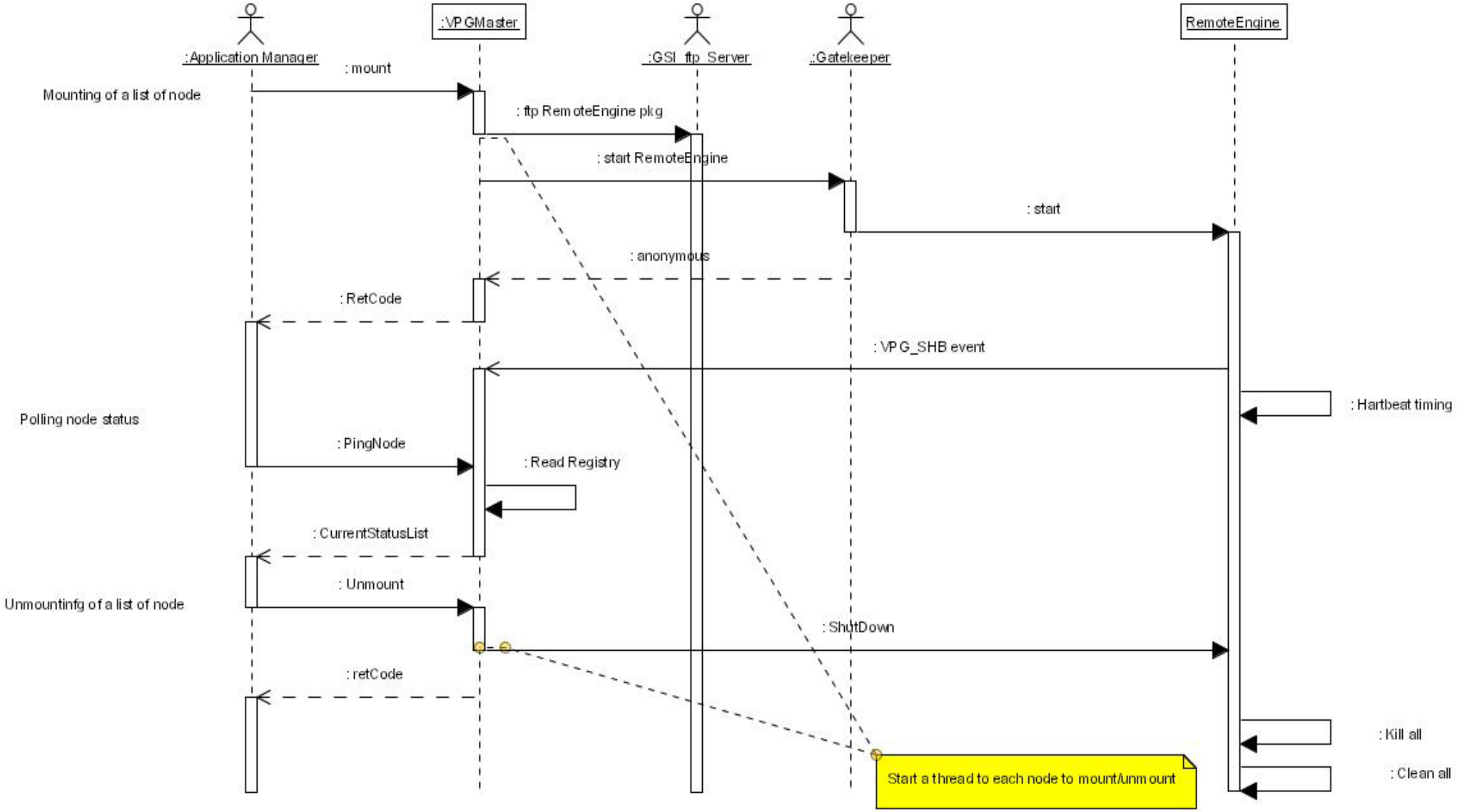
Nome del caso d'uso	SO Signal Handlig
Attori coinvolti	Attivato dal Sistema Operativo locale per indicare il cambiamento dello stato di esecuzione di un processo; Comunica col VPGMaster tramite eventi indicando il cambiamento di stato.
Entry condition	Cambiamento di stato di un processo
Descrizione	• La Remote Engine permette di reagire agli eventi di sistema operativo relativi ad un processo. In casi di cambiamento di stato dell'esecuzione di un processo viene emesso un evento verso il master per comunicare l'accaduto
Exit condition	Segnale processato (evento messo)

3.3.4.2 Modelli dinamici

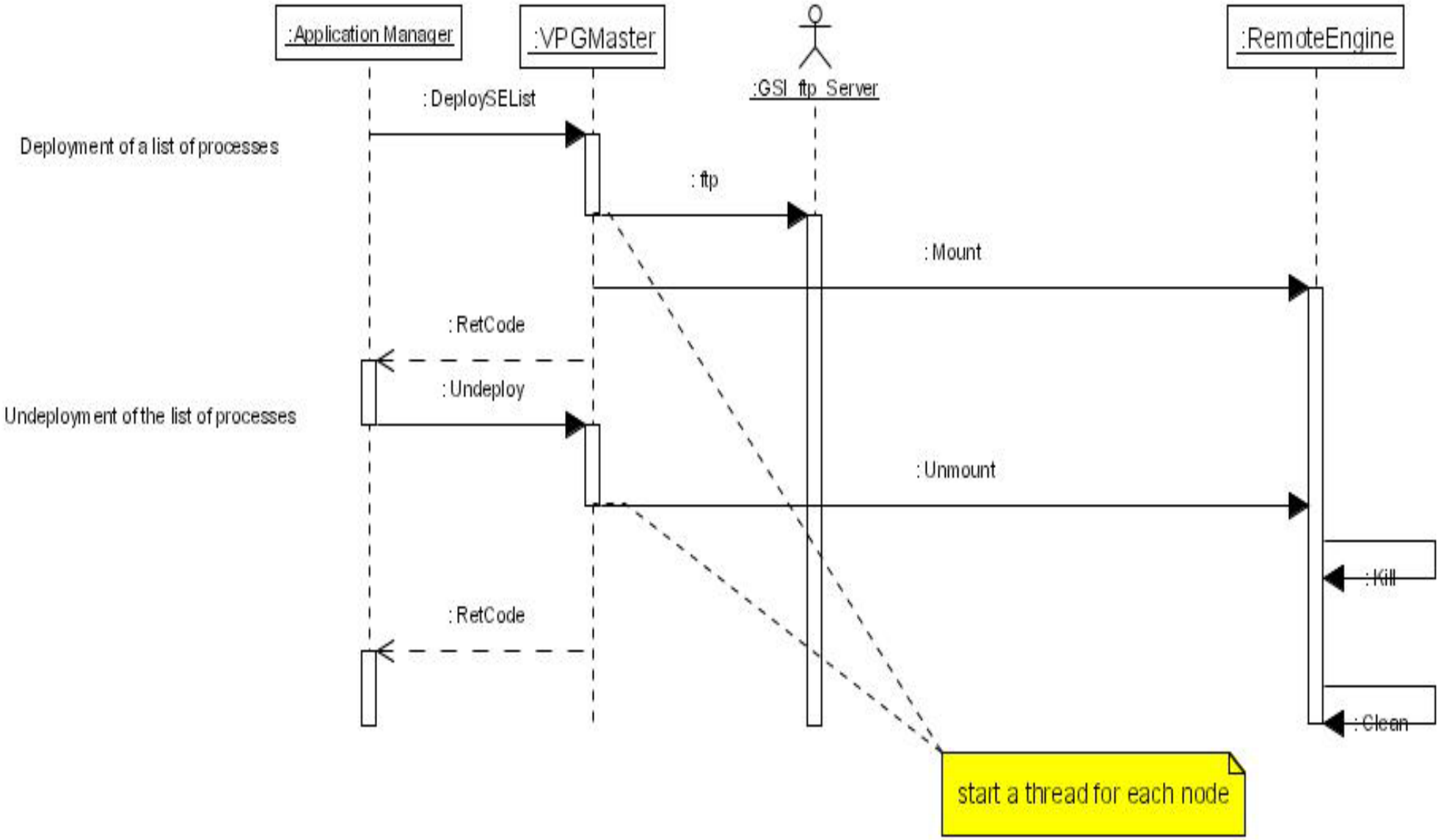
3.3.4.2.1 Diagrammi di sequenza

Di seguito sono riportati alcuni diagrammi che mostrano alcuni scenari tipici di funzionamento del sistema.

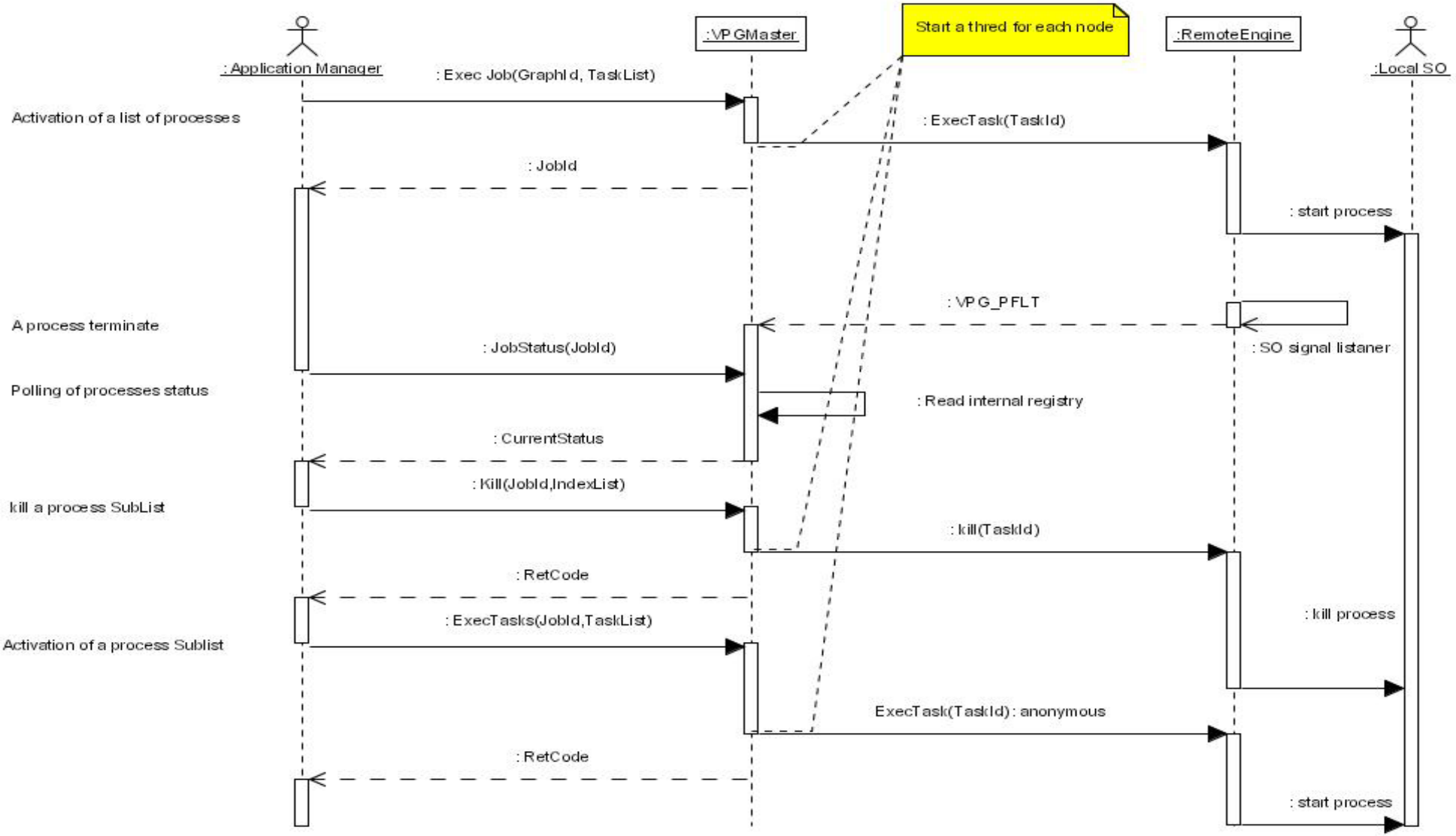
3.3.4.2.1.1 Mounting/unmountig di nodi sulla VPG



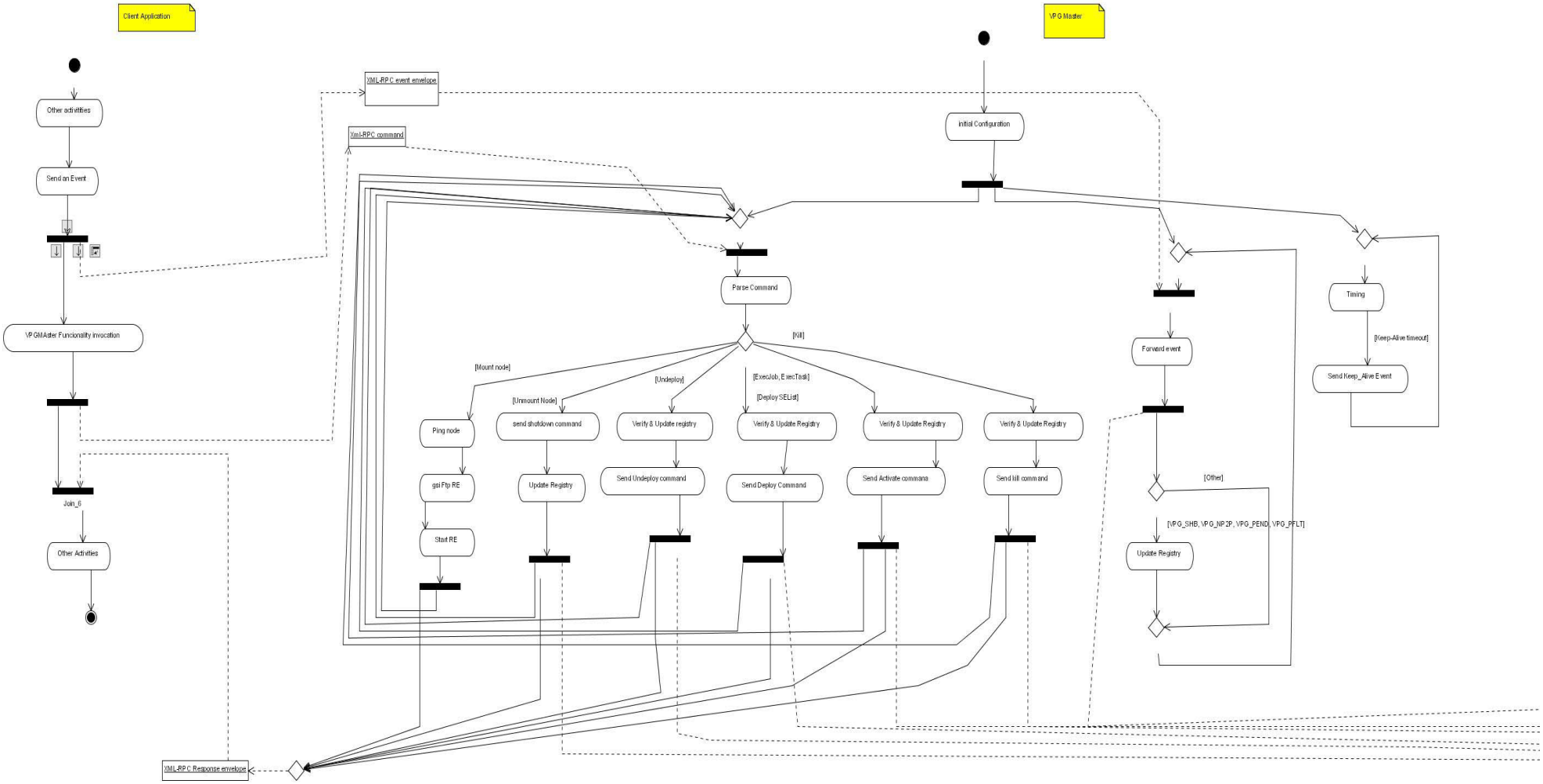
3.3.4.2.1.2 Deployment di un grafo di processi



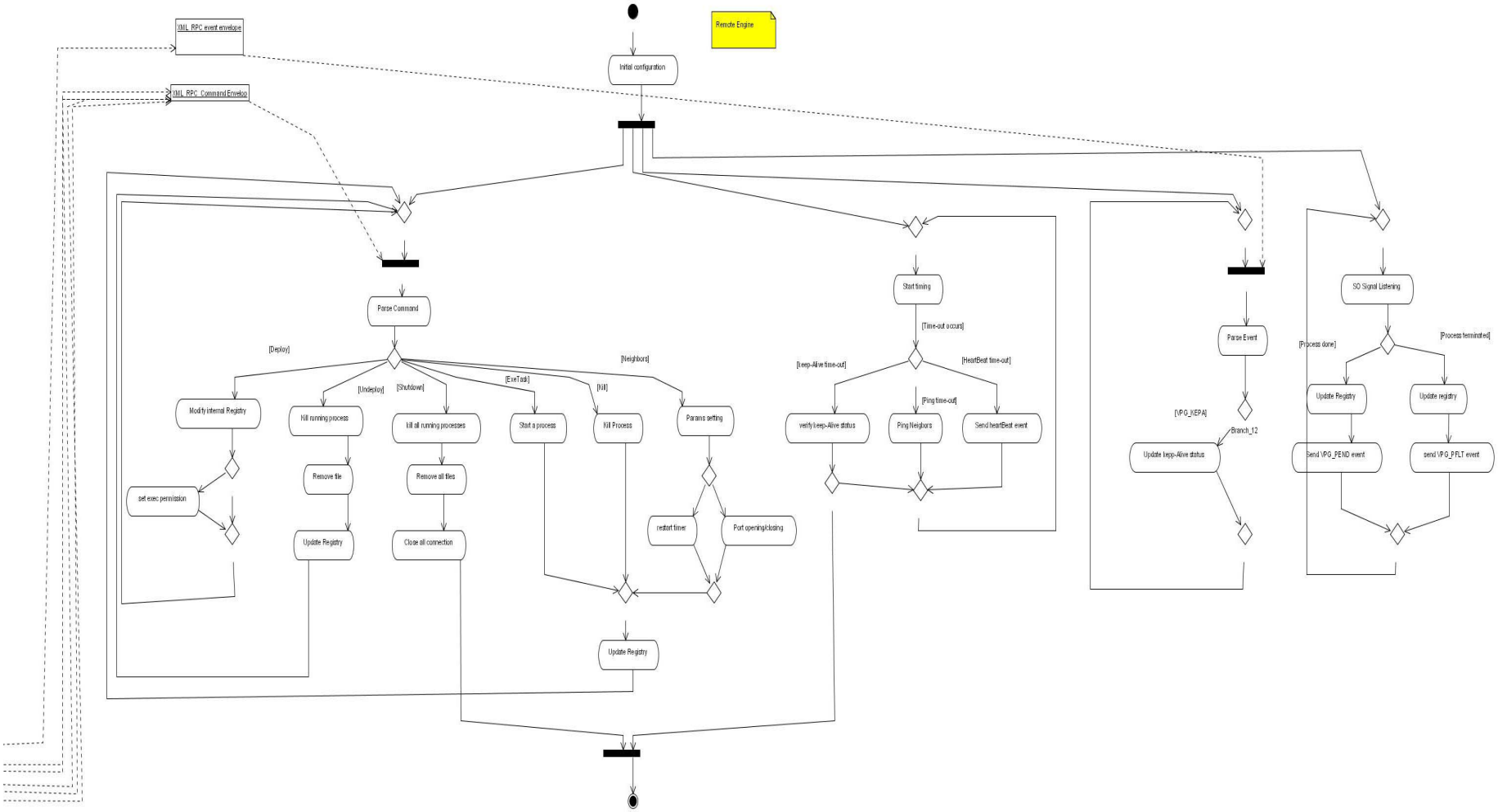
3.3.4.2.1.3 Attivazione (e riattivazione) di grafi di processi



3.3.4.2.2 Diagrammi delle attività



(segue)


XML_RPC event envelope
XML_RPC Command Envelop
Remote Engine
Initial configuration
Parse Command
[Deploy]
[Undeploy]
[Shutdown]
[ExeTask]
[Kill]
[Neighbors]
Modify internal Registry
set exec permission
Kill running process
Remove file
Update Registry
kill all running processes
Remove all files
Close all connection
Start a process
Kill Process
Update Registry
Params setting
restart timer
Port opening/closing
Start timing
[Time-out occurs]
[keep-Alive time-out]
[HeartBeat time-out]
[Ping time-out]
verify keep-Alive status
Ping Neighors
Send heartBeat event
Parse Event
[\PG_KEPA]
Branch_12
Update keep-Alive status
SO Signal Listening
[Process done]
[Process terminated]
Update Registry
Update registry
Send \PG_PEND event
send \PG_PLT event

4 Architettura del sistema (System Design Document)

Versione:
Versione 0.1, data creazione: 10 Ottobre 2004

4.1 Librerie

Il sistema si compone sostanzialmente di quattro package, oltre ad alcuni package third-party disponibili con licenza open-source e utilizzati nell'implementazione del software.

Di seguito è riportato un diagramma che illustra i package sviluppati (colore grigio) e i package utilizzati e viene data una breve descrizione per ognuno di tali package.

4.1.1 VPGMaster

Libreria contenente le classi sviluppate per la realizzazione del front-end della VPG (VPGMaster).

4.1.2 RemoteEngine

Libreria contenente le classi sviluppate per la realizzazione della Remote Engine della VPG, cioè il componente lanciato su ogni nodo acquisito.

4.1.3 Utilities

Libreria contenente alcune classi utili per la creazione/manipolazione dei comandi xml-rpc trasmessi tra i vari componenti e per la gestione delle comunicazioni.

4.1.4 Event Support Library

Libreria contenente l'implementazione dei pattern di una piattaforma ad eventi (Subscribe, Notify) [13] [14].

4.1.5 Xerces-C++

Libreria Open-Source distribuita dall'organizzazione no-profit "Apache Software Foundation" [12] utile per il trattamento di documenti xml. Tale libreria, interamente implementata in C++ ma disponibile anche in Java, permette la creazione, validazione e la manipolazione di documenti e schemi XML.

4.1.6 ACE

Libreria Open-Source [9] che permette lo sviluppo di codice portabile per implementare tutti i meccanismi di

- comunicazione remota tra i diversi host.
- gestione dei segnali di sistema operativo;

4.1.7 VPG Stub ()

Libreria utile allo sviluppo di applicazioni client verso il sistema.

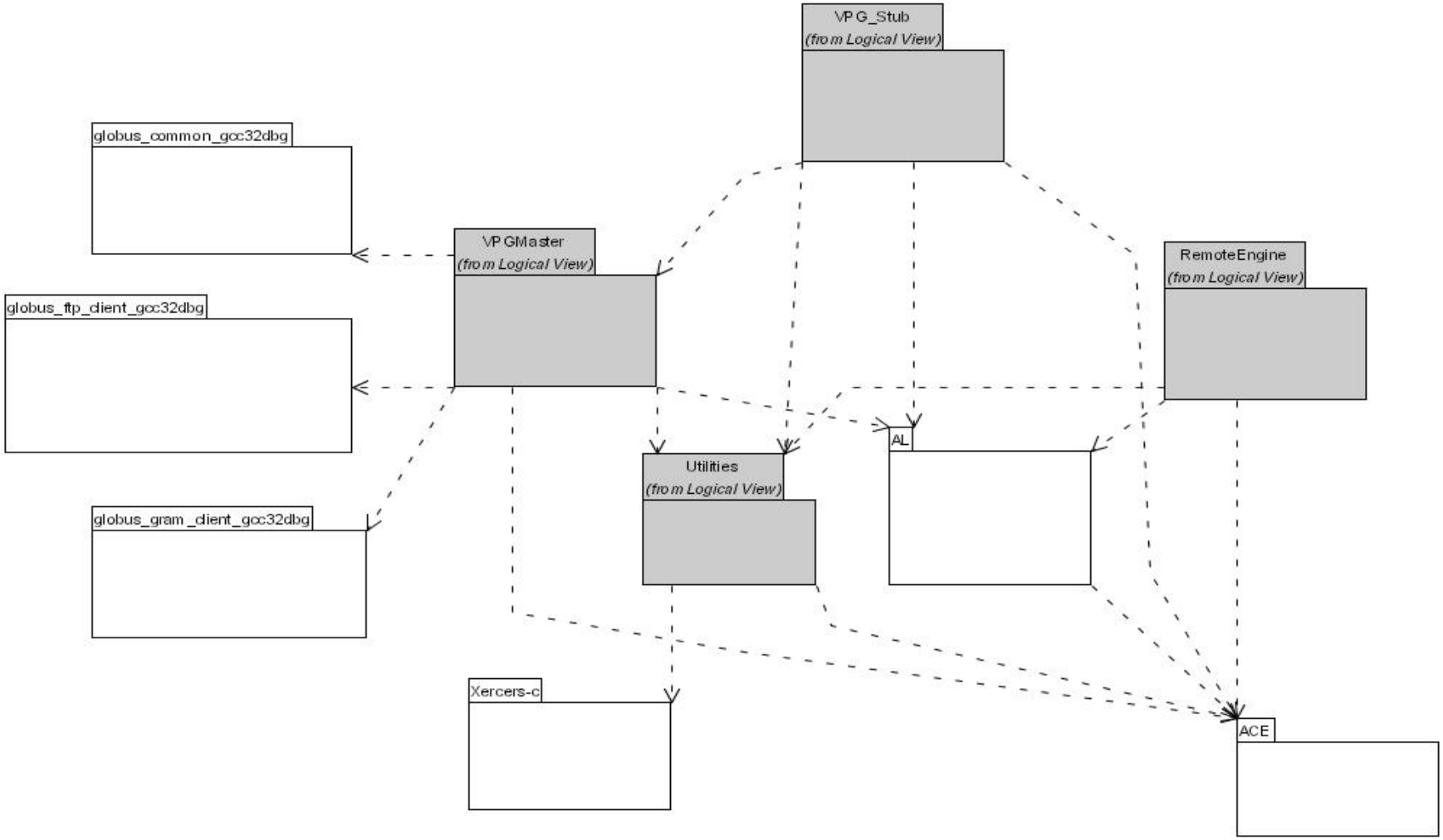
Fornisce un'insieme di classi per gestire le comunicazioni ed effettuare le serializzazioni/deserializzazioni dei comandi.

4.1.8 Moduli Globus Toolkit

Libreria Open-Source [4] che permette la realizzazione di moduli Proxy verso i principali servizi offerti dal Toolkit Globus. I moduli utilizzati sono:

- globus_gram_client_gcc32dbg: codice per lo sviluppo delle classi che permettono la comunicazione con il Gatekeeper di Globus 2
- globus_ftp_client_gcc32dbg: codice per lo sviluppo delle classi che permettono la comunicazione con il server gsi ftp.di Globus 2
- globus_common_gcc32dbg: codice per lo sviluppo di alcune primitive essenziali, come ad esempio il ping di un nodo Globus.

4.2 Diagramma dei package del sistema



4.3 Architettura software

Il sistema è composto da due componenti:

- VPGMaster: costituisce il front-end della griglia, servizio per l'accesso alle funzionalità del sistema;
- RemoteEngine: demone lanciato sul nodo della VPG per fornire tutti i servizi necessari per supportare l'intero ciclo di vita di un processo all'interno di un nodo (esecuzione, kill, clean, etc,…);

L'architettura software utilizzata è sostanzialmente di tipo client-server. Il VPGMaster si presenta come un servizio (server) verso le applicazioni esterne e nel frattempo agisce da client (master) verso le remote engine montate sui vari nodi. Inoltre è stato realizzato un meccanismo di ping non centralizzato tra le varie Remote Engine inspirato dagli approcci peer-to-peer.

4.3.1 Concorrenza

Per una migliore efficienza del sistema si è dovuto ricorrere ad un forte grado di concorrenza nello sviluppo dei componenti. Di seguito viene fornito un elenco dei principali motivi per cui si è reso necessario l'utilizzo di diversi thread:

- esecuzione concorrente dello staging di file su host differenti;
- esecuzione concorrente del mounting/unmountig di nodi;
- monitoraggio sui segnali lanciati dal sistema operativo (necessario per tenere traccia del cambiamento di stato dei processi lanciati);
- gestione dei timing;
- gestione del sistema ad aventi;
- gestione dei canali di entrata dei comandi;

4.3.2 Gestione dei dati

Ognuno dei due componenti del sistema presenta un registro interno con delle tabelle comprendenti tutti i dati necessari alla loro esecuzione (si rimanda alla documentazione delle classi entità per un maggiore approfondimento)

Le tabelle presenti nel registro del componente Remote Engine sono:

- **File**: insieme dei file montati sul nodo locale, utile per il cleaning del nodo;
- **Process**: insieme dei processi lanciati, per ogni processo lanciato viene creato un handler;
- **Neighbors**: insieme dei nodi considerati "adiacenti", cioè l'insieme dei nodi sottoposti a ping periodico;

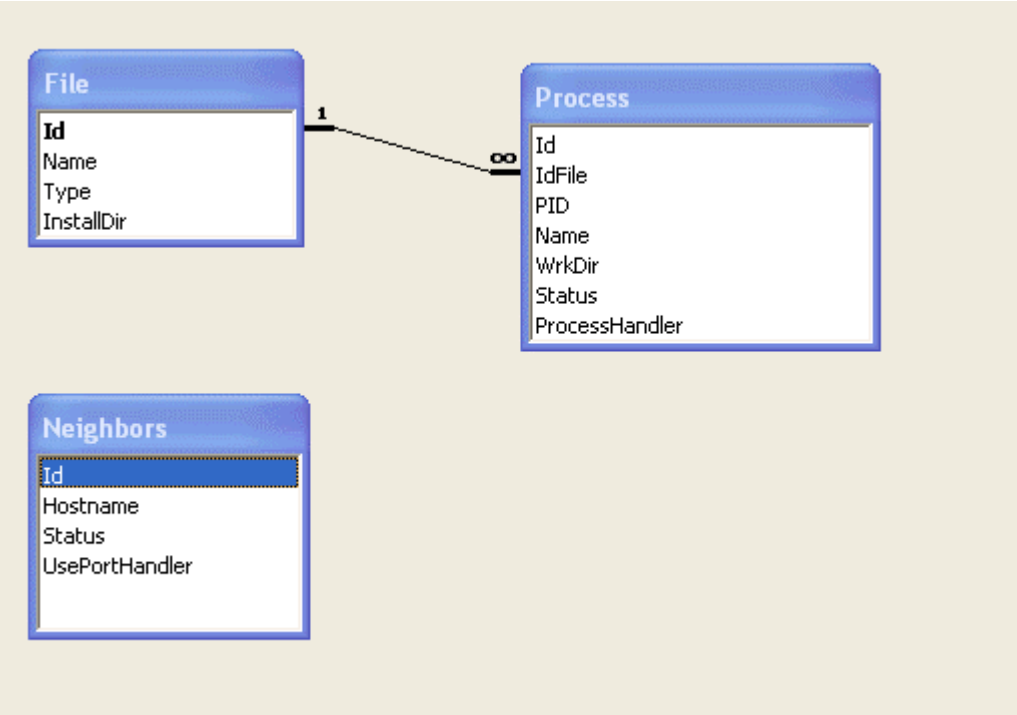


Figura 1. Dati memorizzati nel registro locale della remote engine

Le tabelle presenti nel registro del Master sono:

- SoftwareElement: insieme degli elementi software montate sulla griglia, per ognuno di essi viene descritto a quale grafo appartiene e la lista dei nodi su cui attualmente è installato.
- Job: lista di processi lanciati, ad ogni processo è associato un indice logico;
- RemoteEngine: lista dei nodi montati, per ognuno di essi viene creato un gestore del canale di comunicazione (UsePortHandler)

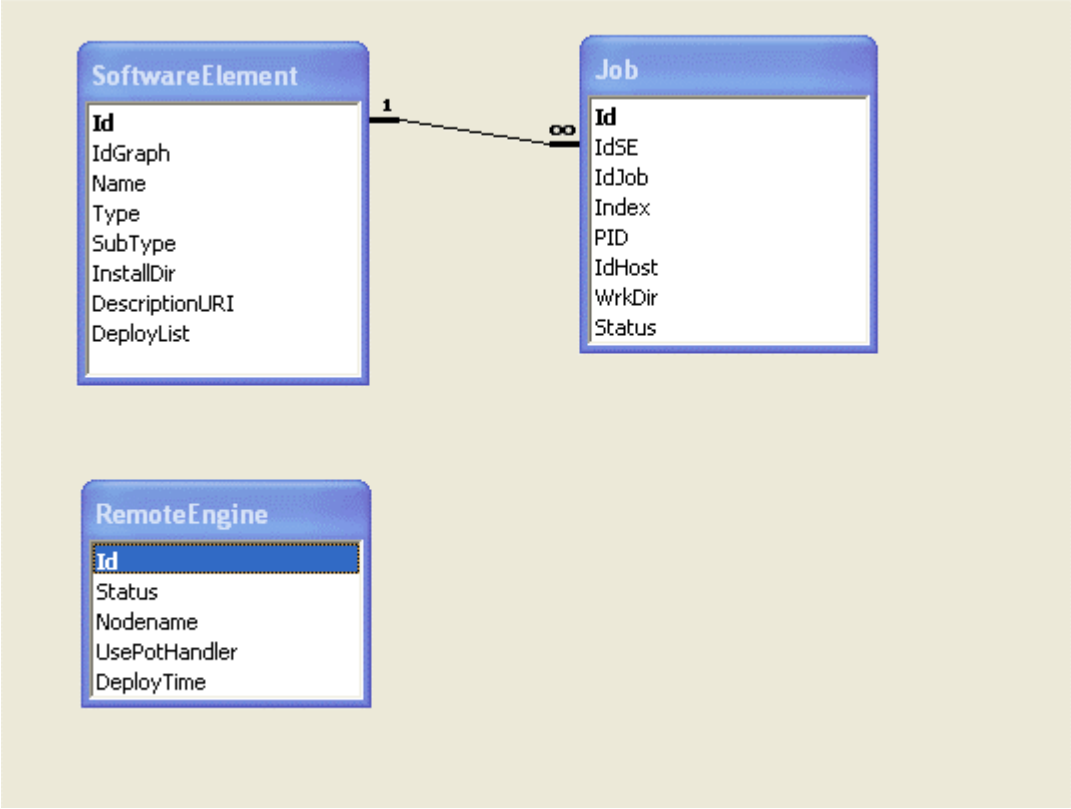
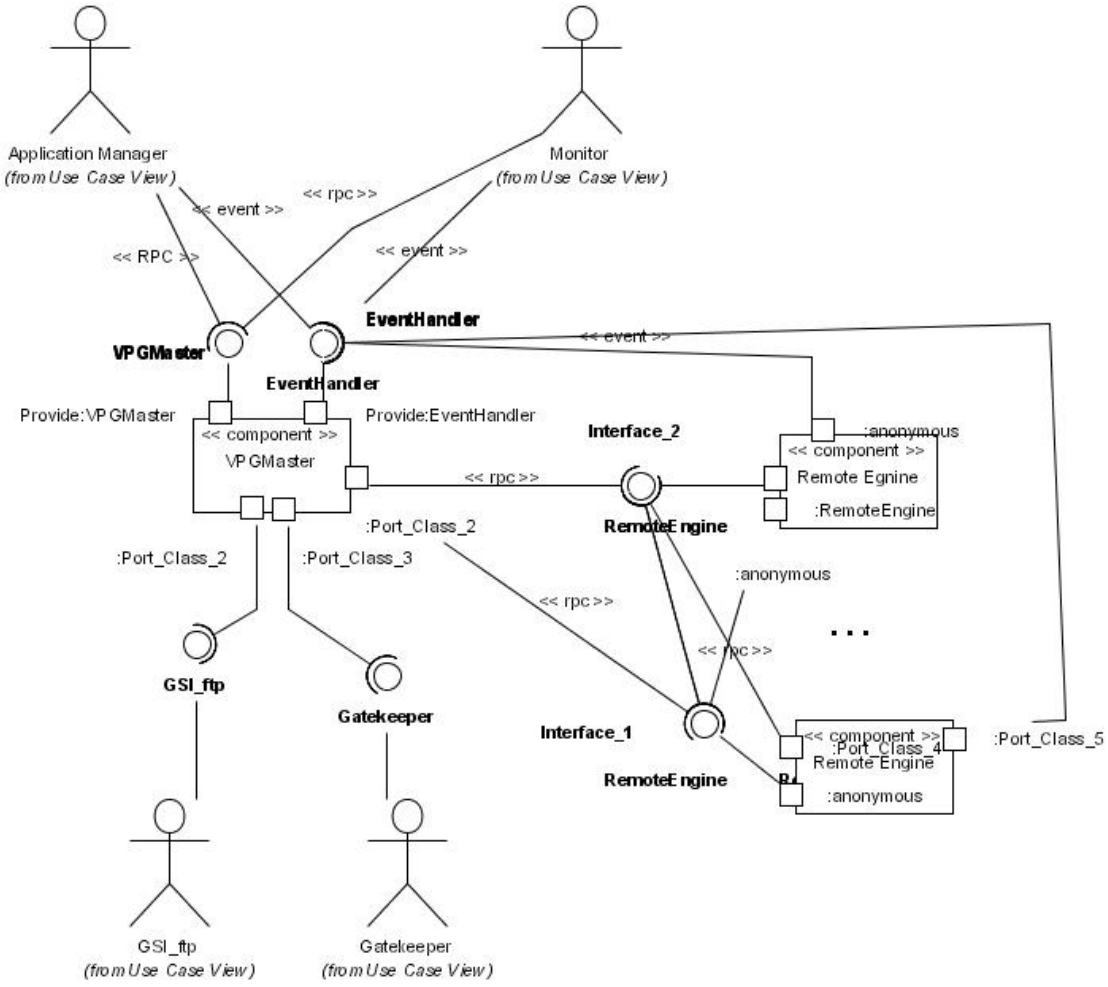


Figura 2 Dati memorizzati nel registro locale del VPGMaster

4.4 Deployment

Come già menzionato il sistema è formato da due componenti, un Master che va installato presso il nodo che svolgerà funzioni da front-end di griglia e un componente slave (Remote Engine) che viene installato su ogni nodo che si intende montare. Il deployment della Remote Engine è a completo carico del componente Master (tramite i servizi di globus 2)

4.4.1 Diagramma di Deployment del sistema



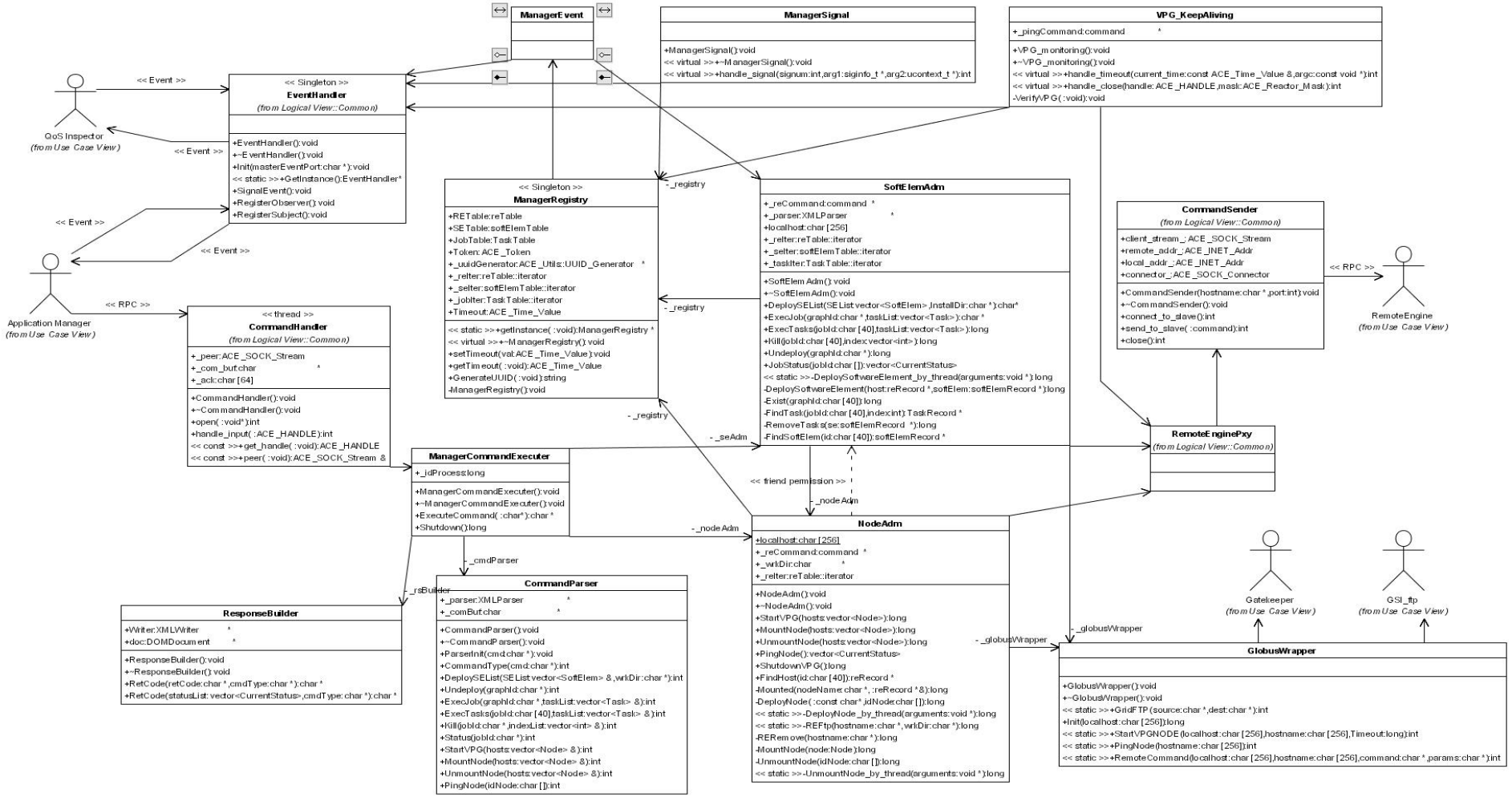
5 Progetto esecutivo
(Object Design Document)

Versione:
Versione 0.1, data creazione: 10 Ottobre 2004;

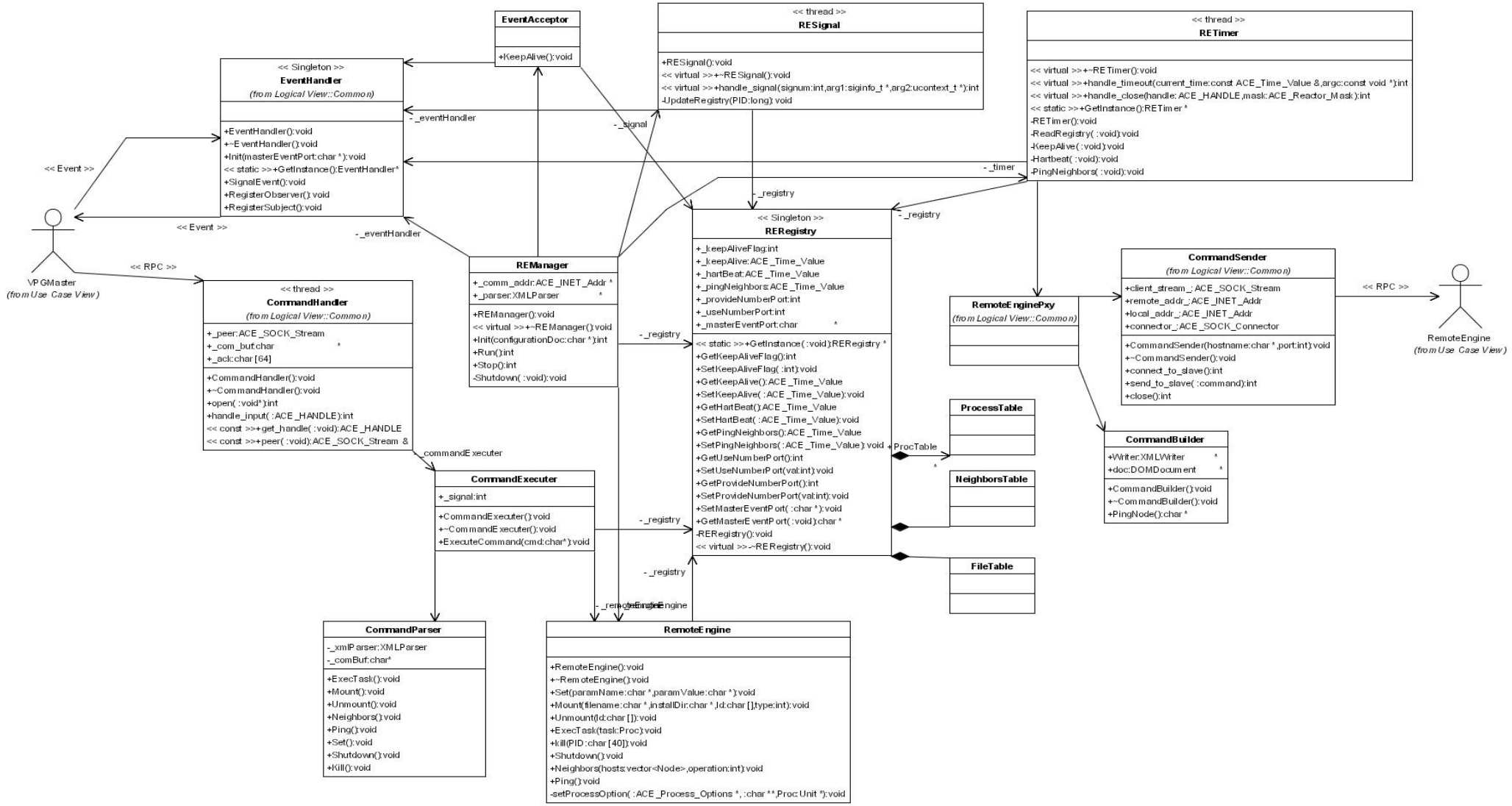
5.1 Diagrammi delle classi

Di seguito sono riportati due diagrammi delle classi che illustrano la struttura dei componenti principali del sistema.

5.1.1 Front-End del sistema (VPGMaster)



5.1.2 Remote Engine



6 Descrizione classi

Segue la documentazione utente delle classi secondo il formalismo DOXYGEN [15]

6.1 VPGMaster Class Documentation

6.1.1 VPGMaster Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

6.1.2 CommandHandler Class Reference

CommandHandler`#include <commandhandler.h>`

6.1.2.1 Public Member Functions

- ACE_HANDLE **get_handle** (void) const
- ACE_SOCK_Stream & **peer** (void) const

6.1.2.2 Detailed Description

handles the incoming command. It implement the "acceptor" and "reader" patterns. A thread (acceptor) wait the incoming connection and when a connection is "open" it wait the incoming stream command (reader).

6.1.2.3 Member Function Documentation

6.1.2.3.1 ACE_HANDLE CommandHandler::get_handle (void) const

new method which returns the handle to the reactor when it asks for it.

6.1.2.3.2 ACE_SOCK_Stream & CommandHandler::peer (void) const

get a reference to the peer stream

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/commandhandler.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/commandhandler.cpp

6.1.3 CommandParser Class Reference

CommandParser`#include <commandparser.h>`

6.1.3.1 Detailed Description

XML-RPC Parser. It parse the incoming command. It has a method for each Remote Engine method which parse the relative command.

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/commandparser.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/commandparser.cpp

6.1.4 CommandSender Class Reference

CommandSender`#include <CommandSender.h>`

6.1.4.1 Public Member Functions

- int **connect_to_slave** ()
- int **send_to_slave** (command)

Uses a stream component to send data to the remote host.

- int **close** ()

Close down the connection properly.

6.1.4.2 Detailed Description

Send a command (via socket) to the other Remote Engine

6.1.4.3 Member Function Documentation

6.1.4.4 *int* CommandSender::close ()

Close down the connection properly.

Close the connection

6.1.4.5 int CommandSender::connect_to_slave ()

Open the socket and connect it to the server

6.1.4.6 int CommandSender::send_to_slave (command)

Uses a stream component to send data to the remote host.

Send a command

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/CommandSender.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/CommandSender.cpp

6.1.5 DeployArg Class Reference

DeployArg`#include <softelemadm.h>`

6.1.5.1 Detailed Description

Argument to pass to the threads to deploy a software element

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/softelemadm.h

6.1.6 EventAcceptor Class Reference

EventAcceptor`#include <eventacceptor.h>`

6.1.6.1 Public Member Functions

- void **EventOccour** (const string &event, const AL_DataSet &data)

6.1.6.2 Detailed Description

It handles the incoming event. It parse the event and execute the correct behavoir

---

6.1.6.3 Member Function Documentation

6.1.6.3.1 void EventAcceptor::EventOccour (const string & event, const AL_DataSet & data)

Handles the incoming event

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/eventacceptor.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/eventacceptor.cpp

6.1.7 EventHandler Class Reference

EventHandler`#include <eventhandler.h>`

6.1.7.1 Public Member Functions

- void **Init** (char *masterEventPort)
- template<class eventType, class dataType> void **registerSubject** (AL_Subject< eventType, dataType > *subject, const string &pool)
- template<class eventType, class dataType> void **registerObserver** (AL_Observer< eventType, dataType > *observer, const string &pool)

6.1.7.2 Detailed Description

This class wrap the event system engine and provide a set of functionalities to send/receive event

---

6.1.7.3 Member Function Documentation

*6.1.7.3.1 void EventHandler::Init (char * masterEventPort)*

Initialize the event system

*6.1.7.3.2 template<class eventType, class dataType> void EventHandler::registerObserver (AL_Observer< eventType, dataType > * observer, const string & pool)*

register observer of given event pool

*6.1.7.3.3 template<class eventType, class dataType> void EventHandler::registerSubject (AL_Subject< eventType, dataType > * subject, const string & pool)*

register subject of given event pool

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/eventhandler.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/eventhandler.cpp

6.1.8 GlobusWrapper Class Reference

GlobusWrapper`#include <globuswrapper.h>`

6.1.8.1 Public Member Functions

- long **Init** (char localhost[256])

6.1.8.2 Static Public Member Functions

- int **GridFTP** (char *source, char *dest)
- int **StartVPGNODE** (char localhost[256], char hostname[256], long Timeout)
- int **PingNode** (char hostname[256])
- int **RemoteCommand** (char localhost[256], char hostname[256], char *command, char *params)

6.1.8.3 Detailed Description

This class wraps the Globus 2 functionalities

6.1.8.4 Member Function Documentation

*6.1.8.4.1 int GlobusWrapper::GridFTP (char * source, char * dest) [static]*

This function transfer a file from source to dest URL, the transfer runs using all defaults, which implies standard FTP stream mode.

6.1.8.4.2 Parameters:

cource path of the file to transfer
dest destination host

6.1.8.4.3 Returns:

returned value are: 1. OK, ftp completed -1. Error opening local file -2. Generic Error error code

6.1.8.4.4 long GlobusWrapper::Init (char localhost[256])

Verify the globus proxy in the current node

6.1.8.4.4.1 Parameters:

localhost it is returned by the method

6.1.8.4.4.2 Returns:

returned value are: 1. OK, ftp completed -1. GRAM authentication failure -2. It is impossible to retrieve localhost

6.1.8.5 int GlobusWrapper::PingNode (char hostname[256]) [static]

Verify if a node is reachble by Globus 2 toolkit

6.1.8.5.1.1 Parameters:

hostname the name (IP address) of the node

6.1.8.5.1.2 Returns:

returned value are: 1. OK, host reachble -1. host is unreachble -2. GRAM authentication failure -3. Remote job doesn't start -4. Job status: failure

6.1.8.5.2 int GlobusWrapper::RemoteCommand (char localhost[256], char hostname[256], char * command, char * params) [static]

Execute a remote command in a specified globus host

6.1.8.5.2.1 Parameters:

localhost the name (IP address) of the curret node
hostname the name (IP address) of the node where execute the command

6.1.8.5.2.2 Returns:

returned value are: 1. OK, command executed -1. host is unreachble -2. GRAM authentication failure -3. Remote job doesn't start -4. Job status: failure

6.1.8.5.3 int GlobusWrapper::StartVPGNODE (char localhost[256], char hostname[256], long Timeout) [static]

Starts the Remote Engine in the specified host

6.1.8.5.3.1 Parameters:

localhost the name (IP address) of the curret node
hostname the name (IP address) of the remote node
Timeout the remote engine time-out param

6.1.8.5.3.1 Returns:

returned value are: 1. OK, remote engine started -1. host is unreachble -2. GRAM authentication failure

---

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/globus_wrapper/globuswrapper.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/globus_wrapper/globuswrapper_old.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/globus_wrapper/globuswrapper.cpp
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/globus_wrapper/globuswrapper_old.cpp

6.1.9 management_director Class Reference

management_director`#include <management_director.h>`

6.1.9.1 Public Member Functions

- **~management_director** ()
- int **init** (ACE_Time_Value)
- int **run** (void)
- int **stop** (void)

6.1.9.2 Detailed Description

Main clas of the VPGMaster. It inits local environment and starts the initial thread.

6.1.9.3 Constructor & Destructor Documentation

6.1.9.3.1 management_director::~management_director ()

free memory

6.1.9.4 Member Function Documentation

6.1.9.4.1 int management_director::init (ACE_Time_Value)

Init the environment

6.1.9.4.2 int management_director::run (void)

Start three thread for the:

- handling the incoming command;
- handling the incoming event;
- handling the time-out timing;

6.1.9.4.3 int management_director::stop (void)

Stop the environment

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/management_director.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/management_director.cpp

6.1.10 ManagerRegistry Class Reference

ManagerRegistry`#include <managerregistry.h>`

6.1.10.1 Public Member Functions

- void **setTimeout** (ACE_Time_Value val)
- ACE_Time_Value **getTimeout** (void)
- string **GenerateUUID** (void)

6.1.10.2 Static Public Member Functions

- **ManagerRegistry** * **getInstance** (void)

6.1.10.3 Public Attributes

- reTable **RETable**
- softElemTable **SETable**
- TaskTable **JobTable**
- ACE_Token **Token**

6.1.10.4 Detailed Description

This is an entity class. It store all information about the node of the VPG and the software elements mounted on it.

6.1.10.5 Member Function Documentation

6.1.10.5.1 string ManagerRegistry::GenerateUUID (void)

Create a unique global identifier used for primary key in the tables

*6.1.10.5.2 ManagerRegistry * ManagerRegistry::getInstance (void) `[static]`*

Get the unique instance of the class (pattern singleton

6.1.10.5.2.1 Returns:

the object class

6.1.10.5.3 ACE_Time_Value ManagerRegistry::getTimeout (void)

Get the time-out timer for the VPG polling

6.1.10.5.4 void ManagerRegistry::setTimeout (ACE_Time_Value val)

Set the time-out timer for the VPG polling

6.1.11 Member Data Documentation

6.1.11.1.1 TaskTable ManagerRegistry::JobTable

Table containing all replica element associated to a Software Element. For example all instances of a programm Lauched on each node of the VPG.

6.1.11.1.2 reTable ManagerRegistry::RETable

Table containing information about node mounted on the VPG

6.1.11.1.3 softElemTable ManagerRegistry::SETable

Table containing information about software elemnts mounted on the VPG. A software element can be a process, data, library, etc...

6.1.11.1.4 ACE_Token ManagerRegistry::Token

Token to acquire in multiple thread execution environment

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/managerregistry.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/managerregistry.cpp

6.1.12 RemoteEnginePxy Class Reference

RemoteEnginePxy`#include <remoteenginepxy.h>`

6.1.12.1 Public Member Functions

- void **Mount** (char *filename, char *installDir, char Id[], int type)
- void **ExecTask** (Proc task)
- void **kill** (char PID[40])
- void **Shutdown** ()
- void **Neighbors** (vector< Node > hosts, int operation)
- void **Ping** ()

6.1.12.2 Detailed Description

Remote Engine Proxy. It implements the Remote Procedure Call Mechanizims

---

6.1.12.3 Member Function Documentation

6.1.12.3.1 void RemoteEnginePxy::ExecTask (Proc task)

Start a process on the local node

6.1.12.3.2 void RemoteEnginePxy::kill (char PID[40])

Kill a process specified by his PID after verifing if it terminated

*6.1.12.3.3 void RemoteEnginePxy::Mount (char * filename, char * installDir, char Id[], int type)*

Mount a file on the local host. This is necessary for node cleaning and for setting right permission on the executable files

6.1.12.3.4 void RemoteEnginePxy::Neighbors (vector< Node > hosts, int operation)

Add/Remove host wich this host should ping to verify connections

6.1.12.3.4.1 Parameters:

hosts the list of the node to add/remove
operation flag with folowing value: 1: add; 2: remove;

6.1.12.3.5 void RemoteEnginePxy::Ping ()

Ping command: it is used among Remote Engine to verify ta connection

6.1.12.3.6 void RemoteEnginePxy::Shutdown ()

Shutdown the remote engine daemon. Kill all processes and clean the node

---

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/remoteenginepxy.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/remoteenginepxy.cpp

6.1.13 ResponseBuilder Class Reference

ResponseBuilder`#include <responsebuilder.h>`

6.1.13.1 Detailed Description

Built the XML-RPC command response

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/responsebuilder.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/responsebuilder.cpp

6.1.14 SoftElemAdm Class Reference

SoftElemAdm`#include <softelemadm.h>`

6.1.14.1 Public Member Functions

- char * **DeploySEList** (vector< SoftElem > SEList, char *InstallDir)
- char * **ExecJob** (char *graphId, vector< Task > taskList)
- long **ExecTasks** (char jobId[40], vector< Task > taskList)
- long **Kill** (char jobId[40], vector< int > index)
- long **Undeploy** (char *graphId)
- vector< CurrentStatus > **JobStatus** (char jobId[])

6.1.14.2 Detailed Description

Implements some "Provide" method of the VPGMaster Service

6.1.14.3 Member Function Documentation

*6.1.14.3.1 char * SoftElemAdm::DeploySEList (vector< SoftElem > SEList, char * InstallDir)*

Deploy a Software-Element list on the VPG. Each Software Element can be a dll, exe, agent, etc...

6.1.14.3.1.1 Parameters:

SEList the list of the Software elements belong to the graph
wrkDir working dir of the proceses in the remote host

6.1.14.3.1.2 Returns:

the graph identifier

*6.1.14.3.2 char * SoftElemAdm::ExecJob (char * graphId, vector< Task > taskList)*

Executes a graph of process (task) on specified host

6.1.14.3.2.1 Parameters:

graphId
taskList the list of the tasks (processes) to execute

6.1.14.3.2.2 Returns:

returned value are. the Job Identifier -1. An host isn't currently available for the execution -2. the processes-list doesn't exist in the registry -3. A softElem (virtual process) doesn't exist in the registry

6.1.14.3.3 long SoftElemAdm::ExecTasks (char jobId[40], vector< Task > taskList)

Executes a set of process (task) on specified host belonging a job

6.1.14.3.3.1 Parameters:

taskList the list of the tasks (processes) to execute
jobId identifier of the job

6.1.14.3.3.2 Returns:

returned value are. 1. It's all done -1. The host isn't currently available for the execution -2. the instance doesn't exist in the registry -3. the process-list doesn't exists in the registry

6.1.14.3.4 vector< CurrentStatus > SoftElemAdm::JobStatus (char jobId[])

TODO

6.1.14.3.5 long SoftElemAdm::Kill (char jobId[40], vector< int > index)

kill a list of process as specified in the command

6.1.14.3.5.1 Parameters:

jobId the identifier of the Job
indexList the list of the tasks to kill

6.1.14.3.5.2 Returns:

returned value are. 1. It's all done -3. the process-list doesn't exists in the registry

*6.1.14.3.6 long SoftElemAdm::Undeploy (char * graphId)*

Clean the VPG from the specified process-list identifier

6.1.14.3.6.1 Parameters:

graphId the identifier of the list

6.1.14.3.6.2 Returns:

returned value are. 1. It's all done -3. the process-list doesn't exists in the registry

---

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/softelemadm.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/softelemadm.cpp

6.1.15 thr_arg Class Reference

thr_arg`#include <nodeadm.h>`

6.1.15.1 Detailed Description

6.1.15.1.1 Author:

assist user

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/nodeadm.h

6.1.16 Timer Class Reference

Timer`#include <timer.h>`

6.1.16.1 Detailed Description

Time for the time-interval calculation

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/timer.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/timer.cpp

6.1.17 VPG_KeepAliving Class Reference

VPG_KeepAliving`#include <vpg_monitoring.h>`

6.1.17.1 Public Member Functions

- virtual int **handle_timeout** (const ACE_Time_Value &current_time, const void *argc)

6.1.17.2 Detailed Description

Ping all the Remote Engine and sent a "keep-Alive" command

6.1.17.3 Member Function Documentation

*6.1.17.3.1 int VPG_KeepAliving::handle_timeout (const ACE_Time_Value & current_time, const void * argc) [virtual]*

This metod is invoked by the Reactor when the timeout occurs

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/vpg_monitoring.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/vpg_monitoring.cpp

6.1.18 VPGNODE_Loader Class Reference

VPGNODE_Loader`#include <vpgnode_loader.h>`

6.1.18.1 Detailed Description

6.1.18.1.1 Author:

Saverio Lombardo

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

- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/vpgnode_loader.h
- D:/ICAR/VPG/vpgrts_04_10_16/proc_set_manager/proc_set_manager/vpgnode_loader.cpp

6.2 RemoteEngine Class Documention

6.2.1 RemoteEngine Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

6.2.2 CommandBuilder Class Reference

`#include <commandbuilder.h>`

6.2.2.1 Public Member Functions

- char * **PingNode** ()

6.2.2.2 Detailed Description

Create an XML-RPC command for the Remote Engine

6.2.2.3 Member Function Documentation

*6.2.2.3.1 char * CommandBuilder::PingNode ()*

Create a Ping command

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandbuilder.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandbuilder.cpp

6.2.3 CommandExecuter Class Reference

`#include <commandexecuter.h>`

6.2.3.1 Public Member Functions

- void **ExecuteCommand** (char *cmd)

6.2.3.2 Detailed Description

The main scope is to parse the incoming command (deseruialize) and to execute the procedure call

---

6.2.3.3 Member Function Documentation

6.2.3.3.1 void CommandExecuter::ExecuteCommand (char * cmd)

Parse the incoming command and call the invoked procedure

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandexecuter.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandexecuter.cpp

6.2.4 CommandHandler Class Reference

`#include <commandhandler.h>`

6.2.4.1 Public Member Functions

- int **handle_input** (ACE_HANDLE)
- ACE_HANDLE **get_handle** (void) const
- ACE_SOCK_Stream & **peer** (void) const

6.2.4.2 Detailed Description

handles the incoming command. It implement the "acceptor" and "reader" patterns. A thread (acceptor) wait the incoming connection and when a connection is "open" it wait the incoming stream command (reader).

6.2.4.3 Member Function Documentation

6.2.4.3.1 ACE_HANDLE CommandHandler::get_handle (void) const

new method which returns the handle to the reactor when it asks for it.

6.2.4.3.2 int CommandHandler::handle_input (ACE_HANDLE)

Callback method invoked when a request arrives

6.2.4.3.3 ACE_SOCK_Stream & CommandHandler::peer (void) const

get a reference to the peer stream

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandhandler.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandhandler.cpp

6.2.5 CommandParser Class Reference

`#include <commandparser.h>`

6.2.5.1 Detailed Description

XML-RPC Parser. It parse the incoming command. It has a method for each Remote Engine method which parse the relative command.

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandparser.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandparser.cpp

6.2.6 CommandSender Class Reference

`#include <commandsender.h>`

6.2.6.1 Public Member Functions

- int **connect_to_slave** ()
- int **send_to_slave** (command)

Uses a stream component to send data to the remote host.

- int **close** ()

Close down the connection properly.

6.2.6.2 Detailed Description

Send a command (via socket) to the other Remote Engine

6.2.6.3 Member Function Documentation

6.2.6.3.1 int CommandSender::close ()

Close down the connection properly.

Close the connection

6.2.6.3.2 int CommandSender::connect_to_slave ()

Open the socket and connect it to the server

6.2.6.3.3 int CommandSender::send_to_slave (command)

Uses a stream component to send data to the remote host.

Send a command

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandsender.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/commandsender.cpp

6.2.7 EventAcceptor Class Reference

`#include <eventacceptor.h>`

6.2.7.1 Public Member Functions

- void **EventOccour** (const string &event, const AL_DataSet &data)

6.2.7.2 Detailed Description

It handles the incoming event. It parse the event and execute the correct behavoir

---

6.2.7.3 Member Function Documentation

6.2.7.3.1 void EventAcceptor::EventOccour (const string & event, const AL_DataSet & data)

Handles the incoming event

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/eventacceptor.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/eventacceptor.cpp

6.2.8 EventHandler Class Reference

`#include <eventhandler.h>`

6.2.8.1 Public Member Functions

- void **Init** (char *masterEventPort)
- template<class eventType, class dataType> void **registerSubject** (AL_Subject< eventType, dataType > *subject, const string &pool)
- template<class eventType, class dataType> void **registerObserver** (AL_Observer< eventType, dataType > *observer, const string &pool)

6.2.8.2 Detailed Description

This class wrap the event system engine and provide a set of functionalities to send/receive event

---

6.2.8.3 Member Function Documentation

*6.2.8.3.1 void EventHandler::Init (char * masterEventPort)*

Initialize the event system

*6.2.8.3.2 template<class eventType, class dataType> void EventHandler::registerObserver (AL_Observer< eventType, dataType > * observer, const string & pool)*

register observer of given event pool

*6.2.8.3.3 template<class eventType, class dataType> void EventHandler::registerSubject (AL_Subject< eventType, dataType > * subject, const string & pool)*

register subject of given event pool

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/eventhandler.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/eventhandler.cpp

6.2.9 REManager Class Reference

`#include <remanager.h>`

6.2.9.1 Public Member Functions

- virtual **~REManager** ()
- int **Init** (char *configurationDoc)
- int **Run** ()
- int **Stop** ()

6.2.9.2 Detailed Description

Main clas of the Remote Engine. It inits local environment and starts the initial thread.

6.2.9.3 Constructor & Destructor Documentation

6.2.9.4 REManager::~REManager () `[virtual]`

do nothing

6.2.9.5 Member Function Documentation

*6.2.9.5.1 int REManager::Init (char * configurationDoc)*

Init the environment

6.2.9.5.2 int REManager::Run ()

Start four thread for :

- handling the incoming command;
- handling the incoming event;
- handling the SO signal;
- handling the time-out timing;

6.2.9.5.3 int REManager::Stop ()

Stop the environment

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/remanager.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/remanager.cpp

6.2.10 RemoteEngine Class Reference

`#include <remoteengine.h>`

6.2.10.1 Public Member Functions

- void **Mount** (char *filename, char *installDir, char Id[], int type)
- void **ExecTask** (Proc task)
- void **kill** (char PID[40])
- void **Shutdown** ()
- void **Neighbors** (vector< Node > hosts, int operation)
- void **Ping** ()

6.2.10.2 Detailed Description

Implements all the "Provide" method of the RemoteEngine

6.2.10.3 Member Function Documentation

6.2.10.3.1 void RemoteEngine::ExecTask (Proc task)

Start a process on the local node

6.2.10.3.2 void RemoteEngine::kill (char PID[40])

Kill a process specified by his PID after verifing if it terminated

*6.2.10.3.3 void RemoteEngine::Mount (char * filename, char * installDir, char Id[], int type)*

Mount a file on the local host. This is necessary for node cleaning and for setting right permission on the executable files

6.2.10.3.4 void RemoteEngine::Neighbors (vector< Node > hosts, int operation)

Add/Remove host wich this host should ping to verify connections

6.2.10.3.4.1 Parameters:

hosts the list of the node to add/remove
operation flag with folowing value: 1: add; 2: remove;

6.2.10.3.5 void RemoteEngine::Ping ()

Ping command: it is used among Remote Engine to verify ta connection

6.2.10.3.6 void RemoteEngine::Shutdown ()

Shutdown the remote engine daemon. Kill all processes and clean the node

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/remoteengine.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/remoteengine.cpp

6.2.11 RERegistry Class Reference

`#include <reregistry.h>`

6.2.11.1 Public Member Functions

- int **GetKeepAliveFlag** ()
- void **SetKeepAliveFlag** (int)
- ACE_Time_Value **GetKeepAlive** ()
- void **SetKeepAlive** (ACE_Time_Value)
- ACE_Time_Value **GetHeartBeat** ()
- void **SetHartBeat** (ACE_Time_Value)
- ACE_Time_Value **GetPingNeighbors** ()
- void **SetPingNeighbors** (ACE_Time_Value)
- int **GetUseNumberPort** ()
- void **SetUseNumberPort** (int val)
- int **GetProvideNumberPort** ()
- void **SetProvideNumberPort** (int val)
- void **SetMasterEventPort** (char *)
- char * **GetMasterEventPort** (void)

6.2.11.2 Static Public Member Functions

- **RERegistry** * **GetInstance** (void)

6.2.11.3 Public Attributes

- Process_Reg **ProcTable**

6.2.11.4 Detailed Description

This class is a Singletone class that contains all the application data used in the Remote Engine

---

6.2.11.5 Member Function Documentation

6.2.11.5.1 ACE_Time_Value RERegistry::GetHeartBeat ()

Get the GetHeartBeat time-out

*6.2.11.5.2 RERegistry * RERegistry::GetInstance (void) `[static]`*

Get the unique instance of the class (pattern singleton

6.2.11.5.2.1 Returns:

the object class

6.2.11.5.3 ACE_Time_Value RERegistry::GetKeepAlive ()

Get the kepp-ALive time-out

6.2.11.5.4 int RERegistry::GetKeepAliveFlag ()

Get the keep-alive flag

*6.2.11.5.5 char * RERegistry::GetMasterEventPort (void)*

Get the MasterEventPort socket number

6.2.11.5.6 ACE_Time_Value RERegistry::GetPingNeighbors ()

Get the Ping time-out

6.2.11.5.7 int RERegistry::GetProvideNumberPort ()

Get the ProvidePort socket number

6.2.11.5.8 int RERegistry::GetUseNumberPort ()

Get the UsePort socket number

6.2.11.5.9 void RERegistry::SetHartBeat (ACE_Time_Value)

Set the GetHeartBeat time-out

6.2.11.5.10 void RERegistry::SetKeepAlive (ACE_Time_Value)

Set the kepp-ALive time-out

6.2.11.5.11 void RERegistry::SetKeepAliveFlag (int)

Set the keep-alive flag

*6.2.11.5.12 void RERegistry::SetMasterEventPort (char *)*

Set the MasterEventPort socket number

6.2.11.5.13 void RERegistry::SetPingNeighbors (ACE_Time_Value)

Set the Ping time-out

6.2.11.5.14 void RERegistry::SetProvideNumberPort (int val)

Set the ProvidePort socket number

6.2.11.5.15 void RERegistry::SetUseNumberPort (int val)

Set the UsePort socket number

6.2.11.6 Member Data Documentation

6.2.11.6.1 Process_Reg RERegistry::ProcTable

Table containing running process

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/reregistry.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/reregistry.cpp

6.2.12 RESignal Class Reference

`#include <resignal.h>`

6.2.12.1 Public Member Functions

- virtual int **handle_signal** (int signum, siginfo_t *arg1, ucontext_t *arg2)

6.2.12.2 Detailed Description

this class monitoring signal receiving

6.2.12.3 Member Function Documentation

*6.2.12.3.1 int RESignal::handle_signal (int signum, siginfo_t * arg1, ucontext_t * arg2) [virtual]*

Callback method invoked when a SO signal arrives

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/resignal.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/resignal.cpp

6.2.13 Unit Class Reference

`#include <Unit.h>`

6.2.13.1 Public Member Functions

- virtual void **setid** (char _newVal[40])

6.2.13.2 Detailed Description

This class represents an handle to an application process

6.2.13.1 Member Function Documentation

6.2.13.2 void Unit::setid (char _newVal[40]) [virtual]

Write property Id

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

- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/Unit.h
- D:/ICAR/VPG/vpgrts_04_10_16/remoteengine/remoteengine/grts/Unit.cp

7 Riferimenti Bibliografici

[1] http://wwwbode.cs.tum.edu/~wismuell/publications/sup95/article.html

[2] A. Sundararaj, P. A. Dinda : “Towards Virtual Networks for Virtual Machine Grid Computing”. Proceedings USENIX VM 04

[3] Pesciullesi et als “The implementation of ASSIST”, in Proc. of the EuroPar 2003 International Conference on Parallel and Distributed Computing, Klagenfurt, Austria (2003).

[4] Globus Toolkit , www.globus.org

[5] Z. Balaton and G. Gombas, Resource and Job Monitoring in the Grid, in Proc. of the EuroPar 2003 International Conference on Parallel and Distributed Computing, Klagenfurt, Austria (2003).

[6] M. Mazina : “A Program-level GrADS Approach to using the Grid”, http://www.hipersoft.rice.edu/grads/publications_reports.htm

[7] Unified Modelling Language, OMG Specification, www.omg.org

[8] Workflow Management Coalition, http://www.wfmc.org

[9] The ADAPTIVE Communication Environment (ACE) http://www.cs.wustl.edu/~schmidt/ACE.html

[10] Poseidon for UML Modelling Tools, http://www.gentleware.com/

[11] XML RPC Specification, www.xmlrpc.com

[12] Xerces XML C++ Parser, http://xml.apache.org/xerces-c/

[13] F. Collura , A. Machì, F. Nicotra “An Interactive Distributed Environment for Digital Film Restoration” A. Laganà et als. (Eds.) Computational Science and its Applications ICCSA 2004 Perugia, Italy LNCS 3044 pp. 536-543, 2004 Springer-Verlag 2004

[14] Fabio Collura, Tesi di Laurea in Ingegneria Informatica, Università degli Studi di Palermo, AA 2003-2004: “Una piattaforma distribuita per il supporto al processamento digitale di filmati video”.

[15] DOXYGEN http://www.stack.nl/~dimitri/doxygen/