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

Modellazione UML dello Skeleton di coordinamento di un Componente Parallelo Master-Slave e di patterns per il controllo esterno della sua performance

Alberto Machì, Fabio Collura, Saverio Lombardo

RT-ICAR-PA-05-03 **Marzo 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

Modellazione UML dello Skeleton di coordinamento di un Componente Parallelo Master-Slave e di patterns per il controllo esterno della sua performance

Alberto Machì, Fabio Collura, Saverio Lombardo

Rapporto Tecnico N.: **Marzo 2005**
RT-ICAR-PA-05-03

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 Sommario

L'ottimizzazione delle prestazioni è un aspetto importante della programmazione parallela, ed assume particolare rilevanza in un ambiente di griglia e in architetture software che si conformano al paradigma "Service Oriented Approach" (SOA), in cui l'utente non possiede il diretto controllo sull'esecuzione del proprio codice. Ad oggi non si sono ancora consolidate metodologie standard per controllare ed intervenire sulla performance in tali scenari.
In questo documento presentiamo uno *skeleton* che coordina in maniera adattiva il processamento di "work-items" su una griglia computazionale utilizzando un grafo di processi slave riconfigurabile. Lo *skeleton* mantiene aggiornati ed espone ad un *manage*r esterno indici di performance efficace quali la potenza efficace o l'efficienza parallela supportandolo nel rispetto di un "service level agreement" (SLA). Sono descritti due pattern che definiscono il protocollo d'interazione fra il *manager*, lo *skeleton* ed un *supporto a run-time*, nella gestione di improvvisa caduta di un nodo slave, nel monitoraggio e nel tuning della performance.
Lo skeleton proposto auto-adatta periodicamente le proprie regole di distribuzione del carico di lavoro ai nodi slave secondo indici di performance istantanea di nodi e connessioni del grafo fisico di esecuzione dei processi del modulo parallelo, aggiornati asincronamente dal *manager* esterno. Lo skeleton offre, inoltre, primitive per l'acquisizione ed il rilascio di nodi slave in risposta a richiesta asincrone (eventi) emanati dal *manager*. Nel seguito del documento verrà illustrata l'architettura multi-thread dello skeleton, la logica di distribuzione dei dati, le politiche di check-point e i patterns ideati per la riconfigurazione asincrona.

2 Introduzione

Gli skeletons sono dei patterns per la distribuzione e controllo dei dati comunemente utilizzati negli ambienti di programmazione parallela strutturata per uno sviluppo rapido di applicazioni ad alte prestazioni [1][2]. Essi esonerano i programmatori di algoritmi paralleli dal compito laborioso di implementare le politiche di scheduling, le regole di bilanciamento del carico e la gestione delle comunicazioni inter-processo. Recentemente sono stati condotti diversi studi su possibili estensioni degli skeletons al fine di poter eseguire le applicazioni su ambienti eterogenei e su risorse di griglia. Diversi progetti, infatti, ricorrono all'utilizzo di moduli basati su skeleton per implementare applicazioni grid-aware ad alte prestazioni [3] [4][5][6].
In [3] è proposta una metodologia per sviluppare applicazioni da una libreria di skeleton predefiniti. Patterns di riconfigurazione sono impiegati per implementare lo scheduling adattivo su risorse di griglia eterogenee. Tali patterns sono basati su modelli di costo, inoltre è descritto un modello di costo per lo skeleton pipeline.
In [4] è proposto un approccio orientato agli oggetti per un controllo globale della performance di applicazioni parallele basate sulla composizione di moduli paralleli basati su skeleton. Un package Java, denominato Muskel, è presentato per gestire il controllo sulla performance di farms e pipelines in un ambiente dinamico come le griglie. Lo stesso package fornisce un prototipo di "Application Manager" che si occupa di mantenere un certo livello di performance acquisendo nuove risorse da utilizzare per ridiregendo le computazioni in caso di caduta di nodi o di performance inadeguate. Un supporto run-time è installato su ogni nodo per fornire servizi di trasferimento e avvio dei moduli paralleli.

Entrambi gli approcci offrono meccanismi per fornire delle estensioni agli skeleton per realizzare delle applicazioni alte prestazioni che sappiano gestire l'etereogeneità e l'instabilità delle risorse di griglia. Ma, in accordo con il paradigma della Organizzazione Virtuale [7], la gestione delle risorse di griglia non riguarda solamente il controllo delle prestazioni e la disponibilità delle risorse. Devono essere considerati altri fattori come i costi d'utilizzo delle risorse e le politiche d'utilizzo stabilite dal proprietario della risorsa. Per questi motivi, riteniamo che la gestione di tali aspetti dovrebbe essere mantenuta a livello di gestione complessiva dell'applicazione [6]. D'altro canto, con l'emergente paradigma Service Oriented Architecture (SOA), un'applicazione non è sviluppata all'interno di un ambiente di programmazione, ma è il risultato di una connessione di servizi (service) ognuno sviluppati indipendentemente per fornire determinate funzionalità. L'accoppiamento largo introdotto da quest'approccio rende la gestione della Qualità del servizio (QoS) più complesso.
Mentre la standardizzazione del "service level agreement" sta ricevendo una grande attenzione nella comunità Grid Services [8] e i patterns per il controllo del flusso di lavoro e dati stanno ricevendo grande attenzione nelle comunità che studiano le problematiche di workflow management [9], scarso interesse è stato invece dedicato alla formalizzazione di patterns che controllano la performance di moduli paralleli basati su skeleton. Tali pattern permetterebbero di poter distribuire e connettere i moduli in applicazioni complesse guidate da un gestore del workflow .
Per importare i paradigmi di programmazione parallela strutturata in frameworks basati su sull'architettura SOA, si richiede un'estensione dei patterns di routing [10], comunemente utilizzati dai linguaggi per la composizione di servizi come BPEL [11], o XPDL [12] per il controllo della performance.
In questo documento è modellato una estensione di skeleton parallelo in grado di coordinare la esecuzione di un set di processi su una griglia computazionale secondo il paradigma master-slave. Nel seguito sono descritti due patterns per aggiungere o togliere nodi slave sotto il controllo di un application Manager esterno.
Nella sezione 2 definiamo una corrispondenza tra i termini utilizzati nella programmazione parallela strutturata e quelli utilizzati negli studi di workflow management e descriviamo l'anatomia dello skeleton proposto attraverso dei diagrammi UML dei componenti [15]. Nella sezione 3 descriviamo la "fisiologia" dello skeleton modellando, attraverso dei diagrammi di sequenza UML, la politica di distribuzione dei dati e le politiche che regolano le azioni di checkpoint. Nella sezione 2 presentiamo, attraverso dei diagrammi di sequenza UML, le interazioni tra lo skeleton, l'application manager e il supporto run-time che danno luogo ai patterns di "aggiungi/togli" nodo slave per poter realizzare degli scenari di performance tuning e recover-from-slave-fault.

3 Workflow activities, skeletons, componenti e patterns

Gli ambienti correnti di Workflow management forniscono delle semantiche specifiche per identificare gli elementi del processo che essi modellano, definendo funzionalità e patterns per controllare l'esecuzione delle funzionalità composte. Per esempio la **Workflow Management Coalition (WFMC)** [9] definisce un "**business process**" come un "insieme di una o più procedure o attività collegate che realizzano un obiettivo" [13]. Esso definisce un'attività come "una descrizione di un'unità di lavoro che rappresenta una unità logica all'interno di un processo. Un'attività è tipicamente l'unità minima di lavoro programmata da un sistema che si occupa

dell'attivazione delle attività (Workflow Engine). Un'attività tipicamente genera uno o più **work-item**. Un work-item è una rappresentazione del lavoro da processare (da un partecipante del processo) in un'attività all'interno di una istanza di processo. La definizione del processo, insieme ai dati rilevanti ai fini della determinazione del flusso di programma (**workflow relevant data**), sono usati per controllare la navigazione attraverso le varie attività del processo, fornendo informazioni per le condizioni d'attivazione e uscita delle singole attività e le scelte d'esecuzione parallela/sequenziale".
Secondo queste definizioni uno skeleton parallelo potrebbe coordinare l'intero processo o, più ragionevolmente, una attività che produce/consuma work-items.
La performance è una proprietà non funzionale di un'attività ma non viene definita nel modello WFMC. Il controllo delle proprietà non-funzionali delle attività è assegnato ad un **Sistema di WorkFlow Management (WFMS)** descritto come un "sistema che definisce, gestisce ed esegue 'workflow' attraverso l'esecuzione di software il cui ordine di esecuzione è guidato da una rappresentazione computerizzata del processo". Il WFMS e i partecipanti al processo scambiano informazioni attraverso variabili di stato (relevant data) che ne determinano l'instradamento (routing).
Al fine di supportare il controllo sulla QoS delle attività attraverso skeletons devono essere soddisfatte quattro condizioni:

1. Devono essere estese le funzionalità degli skeleton con delle primitive di riconfigurazione per reagire alle direttive di un WFMS.
2. Va offerta al sistema di gestione (WFMS) esterno una rappresentazione espressiva dello stato interno dello skeleton per supportare la valutazione dinamica del livello di performance.
3. Vanno definiti patterns che codificano i protocolli di interazione tra il sistema di management, lo skeleton ed il supporto a run-time;
4. Devono essere definiti sia una politica per mantenere la coerenza tra i processi dello skeleton in presenza di direttive asincrone di riconfigurazione dal sistema di management che meccanismi per implementare tale politica.

Per esempio, lo skeleton potrebbe mantenere aggiornato, per conto del WFMS, una variabile di stato indicante il numero di work-items processati o di macro-operazioni o di cicli effettuati, e potrebbe metterli a disposizione tramite meccanismi d'introspezione per permettere la valutazione delle performance. Nel caso in cui il contratto di performance non fosse onorato, il WFMS potrebbe intervenire con direttive d'aggiunta o eliminazione di un nodo per variare il grado di parallelismo dello skeleton.
Il punto chiave è che la valutazione delle performance è un processo complesso, che normalmente avviene asincronamente all'esecuzione dello skeleton, e che anche le direttive di management vanno quindi emesse asincronamente al processamento dei work-items.
Punti di checkpoint devono quindi essere inclusi nell'attività dello skeleton al fine di supportare la ri-sincronizzazione ed appropriati patterns d'interazione sono necessari per assicurare la coerenza tra il processamento dei dati e attività di allocazione/liberazione di risorse.
La tecnologia a componenti offre metodologie per supportare il controllo concorrente di elementi software connettibili in modo standard [15]. Architetture a componenti, come il Corba Component Model (CCM) [16] definiscono dei metodi standard e dei meccanismi per interfacciare moduli software tra di loro, per attivare funzionalità e comunicare messaggi perfino in maniera asincrona. Tre interfacce e due processi logici sono necessari per il modello dello skeleton configurabile.

- ◊ Un'interfaccia funzionale per fornire tramite chiamata remota, sincrona o asincrona, i modi di funzionare (funzionalità) dello skeleton;
- ◊ Un'interfaccia di configurazione/introspezione per la lettura/scrittura dei relevant data;
- ◊ Un'interfaccia ad eventi per la notifica asincrona del cambiamento dei relevant data;
- ◊ Un processo o thread per processare i work-items in ingresso allo skeleton;
- ◊ Un processo o thread di controllo per fornire le primitive di controllo al WMFS in maniera asincrona al processameno del work-item;

In figura 1 è riportato un diagramma a componenti UML che mostra l'organizzazione dello skeleton. E' rappresentato, come un attore, un sistema di supporto a run-time che opera da ponte tra il WFMS e lo skeleton. Se il WFMS non integra alcun meccanismo per comunicazioni asincrone (come nel caso dei Web Services) è compito del supporto-run-time fornire un bus ad eventi allo skeleton e presentare al WFMS gli eventi sotto forma di variabili di stato accessibili attraverso una interfaccia di introspezione sincrona.

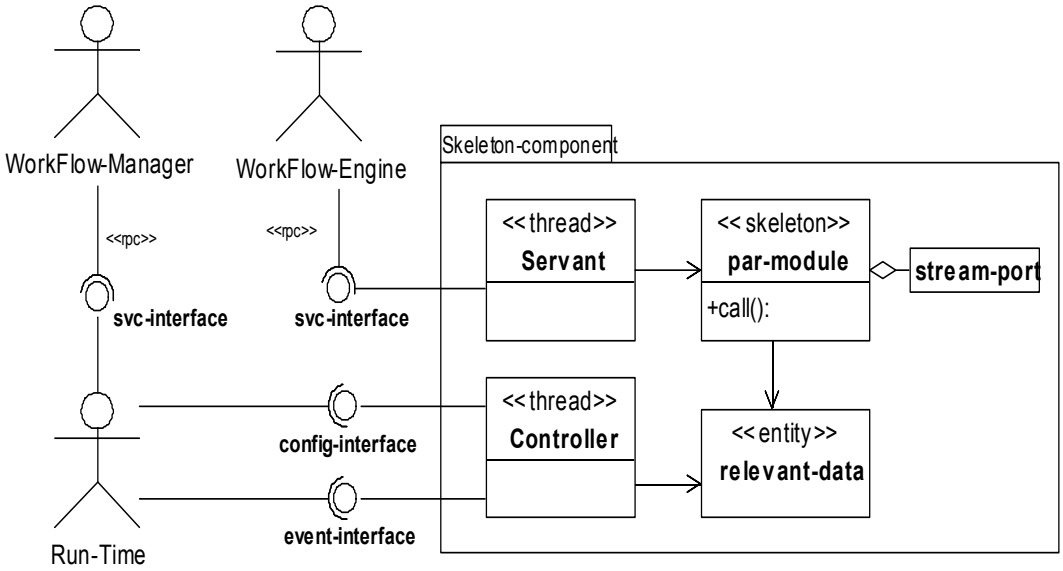

Figura 1. Un Diagramma di Componenti in UML che mostra in un modello logico di organizzazione dello scheletro un thread di controllo ed uno esecutivo interagenti attraverso un oggetto di stato , tre interfacce per invocazione remota delle funzionalità, introspezione e controllo non funzionale ed in fine una porta stream per lo scambio di dati interno allo skeleton.

In [4] il gruppo di ricerca di ASSIST descrive una architettura a componenti e un modello di programmazione per lo sviluppo nativo di componenti ad alte prestazioni, adattivi e basati su skeleton chiamati parmod. Le operazioni di adattività dell'applicazione e di controllo sulle prestazioni sono affidate ad una gerarchia di Managers. I Managers sono sviluppati automaticamente dall'ambiente di programmazione durante la compilazione dei parmod (Module Application Managers or MAM) e durante l'integrazione dell'applicazione (Component Application Managers or CAM). Il deployment dei processi e l'attivazione sulla griglia sono affidati ad un supporto run-time chiamato Grid Abstract Machine. I MAM monitorizzano proattivamente le performance dello skeleton e lo stato delle risorse, servendosi del modello di costo dello skeleton e, se necessario, emettono delle richieste di riconfigurazione ai CAM che analizzano tali richieste in accordo con i criteri d'ottimizzazione globale delle risorse. Nel lavoro i patterns utilizzati per il management non sono formalizzati.

Il sistema proposto nel presente lavoro aderisce al modello di programmazione ed all'architettura a componenti del progetto Grid.it, nello scenario specifico dell'integrazione di componenti (legacy) ad alte prestazioni all'interno di processi gestiti da sistemi di Workflow Management.

Nel Paragrafo successivo sarà riportata l'implementazione a componenti reattivi di uno skeleton master-slave, sarà dettagliato la sua interazione con un Application Manager non gerarchico, utilizzando il supporto offerto dalla macchina astratta Virtual Private Grid (VPG) [6][17].

4 Lo skeleton parallelo riconfigurabile Master-slave

Lo skeleton considerato implementa una distribuzione del carico di lavoro di tipo stream/data-parallel e map-and-reduce [2]. Un processo master gioca sia il ruolo di emitter che di collector. Tale processo ottiene i work-items da processare da uno stream esterno, li partiziona in differenti data-items (non intersecanti), li distribuisce ad un insieme di processi slave (workers) per il processamento concorrente e, finalmente, raccoglie e ordina i risultati. La specializzazione dello skeleton in un farm-ordinante o in un map recursivo è immediata.
Lo skeleton modifica la politica di partizionamento del workload secondo alcuni dati di configurazione (relevant data) forniti dal sistema di management e aggiornati periodicamente dal supporto a run-time. Tali dati di configurazione includono la lista di stimatori dell'effettiva potenza di ogni nodo appartenente al grafo di mappatura dei processi dello skeleton e una lista di stimatori dell'effettiva larghezza di banda dei collegamenti più significativi tra tali nodi.
Lo skeleton emette periodicamente degli eventi di checkpoint per notificare il completamento di alcuni passi di processamento del work-item. Gli eventi trasmettono messaggi contenenti nella loro sezione dati valori di stima del numero di data-item, di iterazioni o di macro-operazioni già completate al checkpoint.
La Figura 2 mostra, tramite un diagramma delle attività UML, le operazioni effettuate dai thread del processo master (a sinistra) e del processo slave (a destra).
Il processo master itera sugli elementi dello stream di ingresso (ruolo di emitter). Nel ciclo più esterno (work-item-loop), esegue le operazioni di pre-processamento divisione e distribuzione del work-item ai workers, quindi di raccolta orinamento e post-processamento del risultato. Al completamento d'ogni iterazione del ciclo di processamento del work-item è emesso un evento.

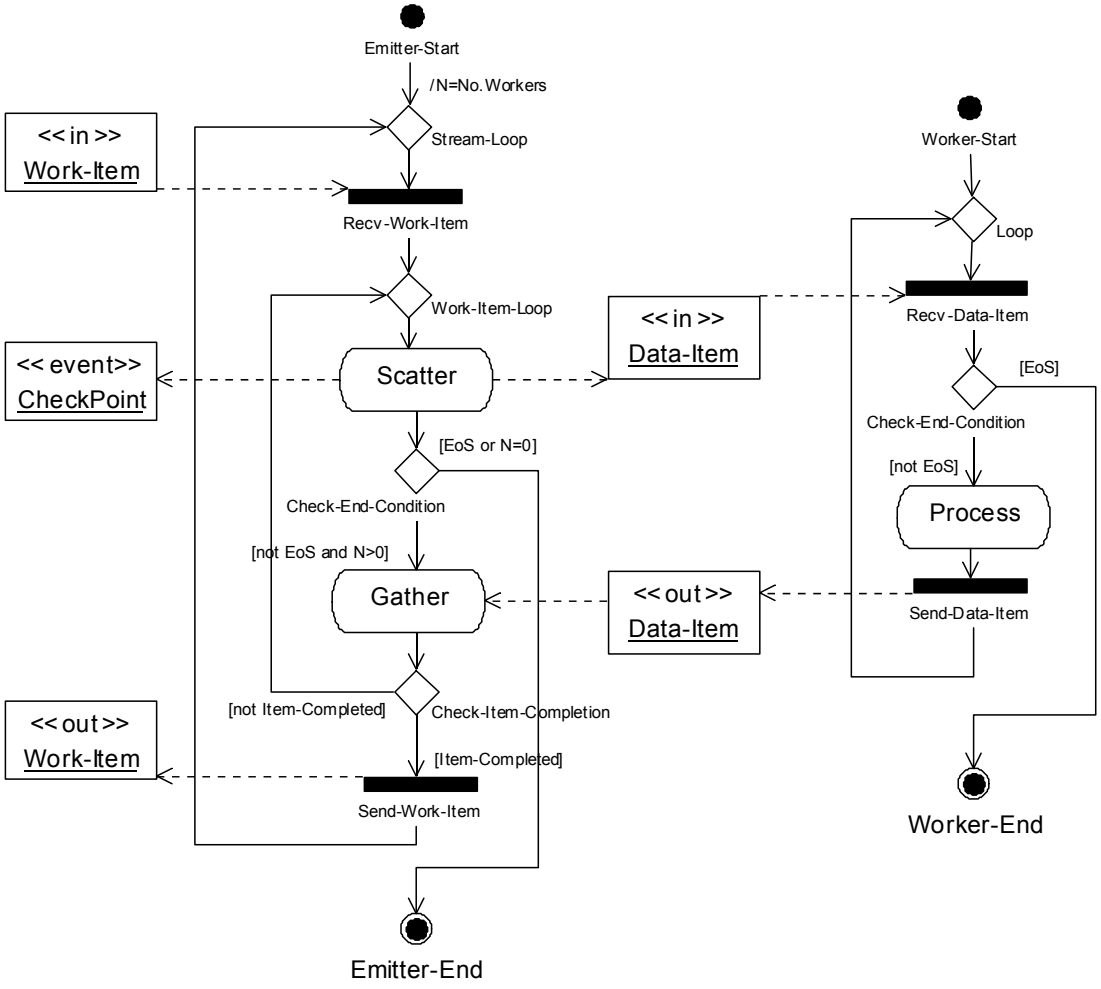

Figura 2. Diagramma delle attività UML che mostra la dinamica delle operazioni del thread del processo master (sinistra) e del nodo slave (destra)

Lo skeleton reagisce a richieste di riconfigurazione asincrone emesse dal Manager. Le primitive di enforcement offerte sono:

- ◊ **Slave-Join**, per aggiungere un nodo all'insieme dei processi workers.
- ◊ **Slave-Disjoin**, per escludere un nodo dall'insieme dei processi worker.

Una sequenza di operazioni di **Slave-Disjoin** e **Slave-Join** realizza una operazione di **Slave-Replacement**. Lo skeleton si sincronizza con le richieste asincrone del Manager e aggiorna la distribuzione del carico controllando i dati di configurazione all'inizio di ogni attività di divisione (scatter). In figura 3 sono mostrati dettagli delle attività di divisione del carico di lavoro (scatter) e di collezione dei risultati (gather). Per evitare situazioni di starvation del collector in caso di caduta di un nodo slave, si utilizza un pattern di controllo di tipo discriminant [10]. Se entro un intervallo di tempo prestabilito il collector non riceve il data-item, effettua un check sui relevant data per verificare lo stato di attività del nodo worker .
In [17] gli autori descrivono un pattern per mantenere una valida conoscenza sullo stato degli nodi slave e notificare eventuali slave-faults. Nel caso di caduta di un nodo slave, il relativo data-item è considerato perso.

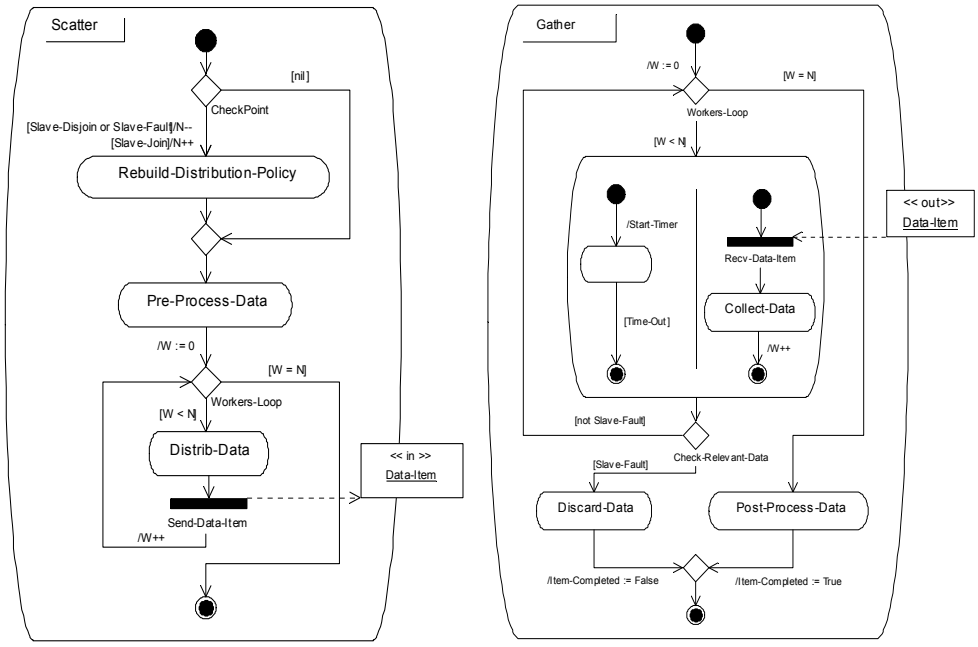

Figura 3. Diagramma delle attività UML che mostra le attività di scatter e gather del processo emitter

La figura 4 illustra il meccanismo di checkpoint utilizzato. Il thread di controllo del processo emitter riceve degli eventi e aggiorna lo stato interno dei relevant-data, condivisi con il thread d' esecuzione. Durante la fase di checkpoint, il thread d'esecuzione legge i relevant-data ed emette un evento di checkpoint.

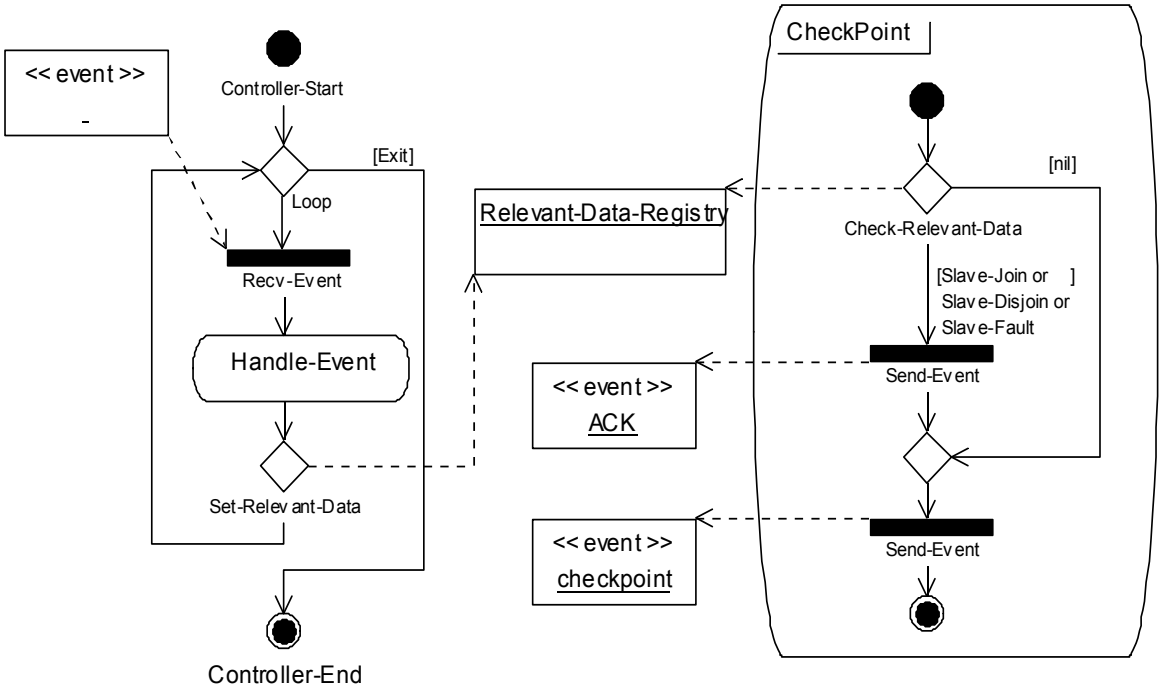

Figura 4. Diagramma delle attività UML che dettaglia la fase di checkpoint

5 Patterns per il controllo della performance

Due patterns comportamentali [18] abilitano un (workflow) Manager al controllo del livello di performance dello skeleton configurabile. Essi utilizzano i servizi offerti al Manager dal supporto run-time VPG [17] che gestisce il ciclo di vita del grafo di processi dello skeleton.
La VPG, infatti, fornisce servizi per:

- ◊ Sottomissione di grafi di processi tramite primitive di deployment, enactment e configuration del grafo di processamento.
- ◊ Enforcement del workflow dello skeleton tramite primitive di riconfigurazione del grafo di processamento trasmesse allo skeleton tramite eventi.
- ◊ Recupero dei Relevant-Data: attraverso primitive per la lettura di informazioni ricevute tramite eventi emessi dallo skeleton ai checkpoint.

Tramite il pattern di recover-from-slave-fault, il processo master dello skeleton continua ad operare anche nel caso in cui cada un nodo slave. Il diagramma di sequenza UML di figura 5 illustra un'istanza del pattern in cui la VPG rivela la caduta di un nodo slave e notifica l'accaduto al processo master dello skeleton tramite un evento. Il processo master, che stava attendendo la terminazione del processamento del data-item inviato al processo slave caduto, dopo il decorrere di un intervallo di time-out, legge l'evento, rimuove il data-item dalla lista dei processati e aggiorna la propria policy di distribuzione dei data-items.

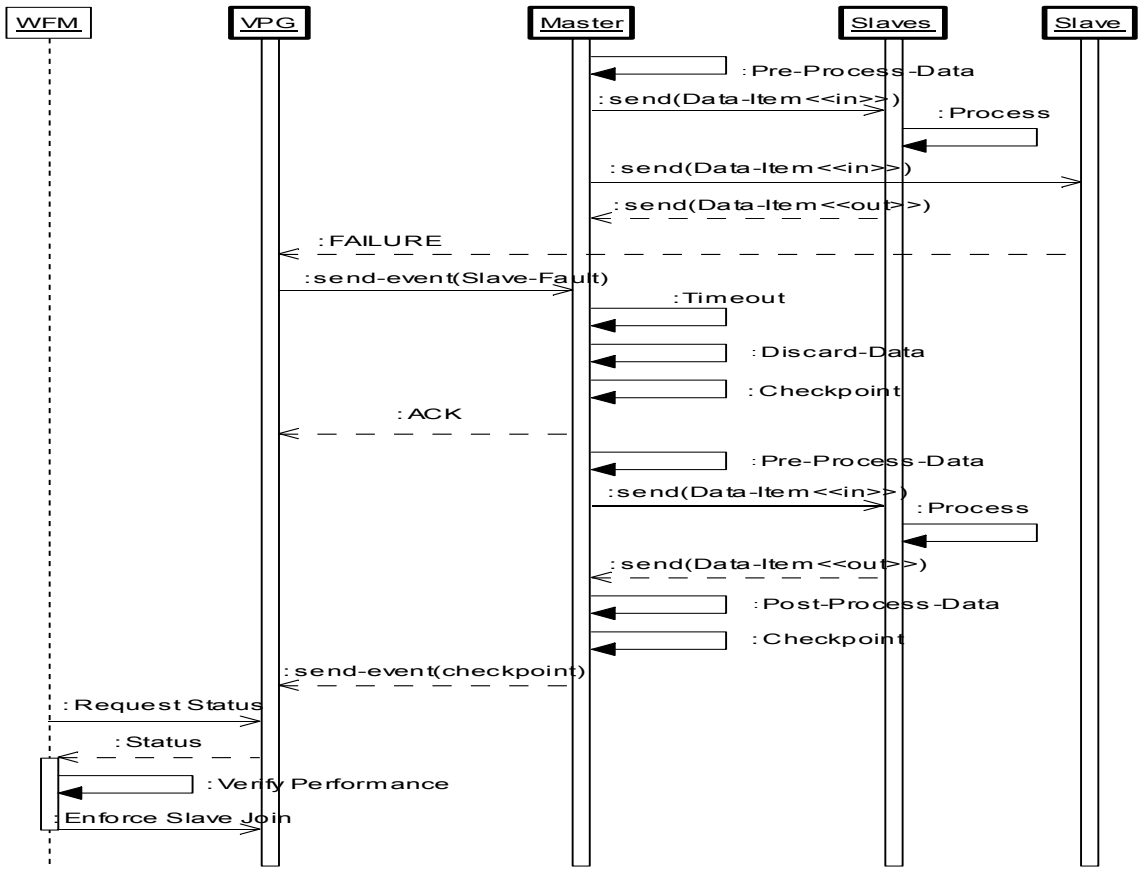

Figura 5. Diagramma di Sequenza UML che illustra uno scenario del pattern "recover-from-slave -fault"

Tramite il pattern **performance-tuning**, il Manager può modificare il livello di performance dello skeleton cambiando la cardinalità dei suoi processi slave. In figura 6 è riportato un diagramma di sequenza UML che illustra il pattern nel caso in cui il Manager legge i relevant-data sullo stato d'avanzamento dell'attività dello skeleton, rileva una decadenza del livello di performance ed emana alla VPG un comando di enforcement "slave-node-join" per aggiungere un nodo slave al grafo dei processi dello skeleton. Dopo il deployment dei files relativi al nuovo nodo da aggiungere, la VPG notifica al processo master dello skeleton un evento di Slave-Join. La modifica del grafo diviene operativa non appena il processo master dello skeleton esce dalla attività di Checkpoint.

6 Conclusioni

In questo documento abbiamo proposto un modello e l'architettura di uno skeleton data-parallel master-slave configurabile. Abbiamo mostrato dei meccanismi e dei patterns comportamentali per il controllo del livello di performance all'interno applicazioni grid-enable guidate da Sistemi di Workflow Management (WFMS). E' stato descritto anche il supporto run-time di un middleware di griglia. Un SDK per l'implementazione dello skeleton è in corso di sviluppo all'interno del progetto **Grid.it**, con l'intento di supportare l'integrazione di codice legacy all'interno di componenti gestibili dai Managers di ASSIT: MAMs/CAMs. Attualmente è in corso lo studio e sviluppo di meccanismi per un pattern di self-updating degli indici di performance dei nodi (attualmente forniti dal Manager esterno) e per il late-binding e configurazione di sensori e attuatori dello skeleton configurabile.

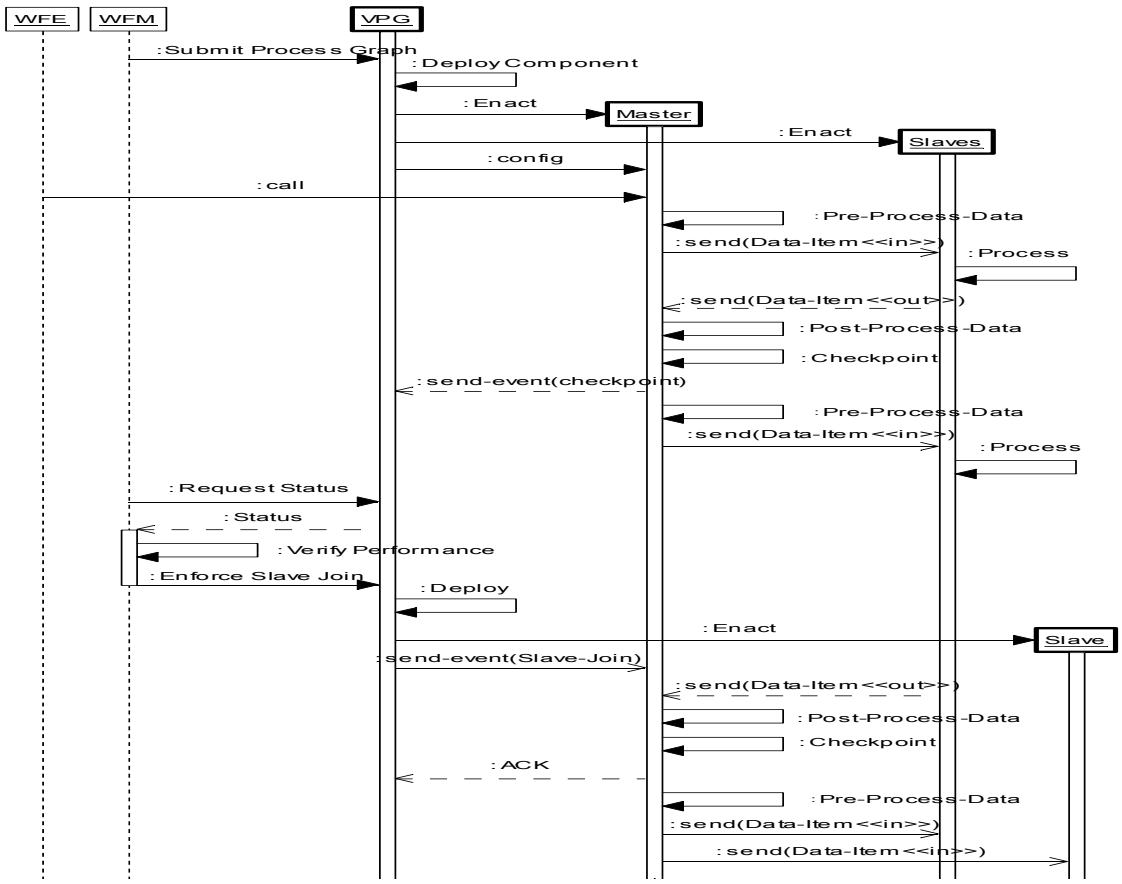

Figura 6. Diagramma di Sequenza UML che illustra uno scenario del pattern “performance-tuining”

7 References

[1] M. Cole. Algorithmic Skeletons: Structured Management of Parallel Computations. Research Monographs in Parallel and Distributed Computing. Pitman, 1989

[2] B. Bacci, M. Danelutto, S. Orlando, S. Pelagatti, M. Vanneschi, P3L: A structured high level programming language and its structured support, Concurrency: Practice and Experience, 7(3):225-255, May 1995

[3] A. Benoit, M. Cole, S. Gilmore and J. Hillstonm. “Evaluating the Performance of Skeleton-Based High Level Parallel Programs”, ICCS 2004, Kraków, Poland, Springer-Verlag LNCS Vol. 3038, pp. 289-296, June 6-9, 2004

[4] M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, C. Zoccolo "Components for high performance Grid programming in the Grid.it Project". Intl. Workshop on Component Models and Systems for Grid Applications.

[5] M. Danelutto, "QoS in parallel programming through application managers" Proceedings of the Euromicro Conference on Parallel, Distributed and Network-based Processing, Lugano, Feb. 2005

[6] 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

[7] F.Berman, G.C. Fox, A.J.G.Hey: Grid Computing. Making the Global Infrastructure a Reality. Wiley 2003

[8] Web Service Level Agreements (WSLA) Project - SLA Compliance Monitoring for e-Business on demand http://www.research.ibm.com/wsla/

[9] The Workflow Management Coalition, www.wfmc.org

[10] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros.
Advanced Workflow Patterns. 7th International Conference on Cooperative Information Systems (CoopIS 2000), LNCS volume 1901, pages 18-29. Springer-Verlag, Berlin, 2000.

[11] Business Process Execution Language for Web Services, the specification. http://www-106.ibm.com/developerworks/library/ws-bpel/

[12] Workflow Process Definition Interface -- XML Process Definition Language (XPDL) Document Number WFMC-TC-1025

[13] The Workflow Reference Model (WFMC-TC-1003, 19-Jan-95, 1.1)

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

[15]Clemens Szyperski, “Component Software: Beyond Object-Oriented Programming”, Addison-Wesley, 1998.

[16] R. Armstrom, D. Gannon, K. Keahey, S.Kohn, L.McInnes, S. Parker, B. Smolinsk. “Toward a common component architecture for high-performance scientific computing”. In Conference on High Performance Distributed Computing, 1999

[17] A. Machì., F.Collura, S. Lombardo,” Dependable Execution of Workflow Activities on a Virtual Private Grid Middleware”, submitted to “2nd International Workshop on Active and Programmable Grids Architectures and Components APGAC'05-ICCS-2005 - Atlanta, USA; 22-25 May 2005

[18] Douglas C. Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann “Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects” Wiley & Sons in 2000, ISBN 0-471-60695-2.