Replicazione Master-Slave per Default Places in sistemi SOMA

Alessandro Ghigi - 0000156175
Reti di Calcolatori LS - Prof. A. Corradi - A.A. 2003-2004

Abstract

Il progetto consiste nella realizzazione di un sistema di replicazione Master-Slave per Default Places in sistemi SOMA. In particolare è stata concentrata l'attenzione sulla replicazione dei servizi di base, quali le tabelle DNS e PNS, il NetworkManager (in particolare le connessioni permanenti da/verso il Place) ed infine l'AgentManager, responsabile della gestione degli agenti mobili. In seguito a questa prima fase, sulla nuova architettura sono stati effettuati alcuni test; grazie anche all'ausilio di un semplice agente mobile, è stato possibile sperimentare le potenzialità di un sistema di questo tipo e verificare, simulando la caduta di un nodo, l'affidabilità dell'infrastruttura di replicazione costruita.

1 Agenti Mobili

Il paradigma ad Agenti Mobili rappresenta un'idea innovativa ed un tentativo per porre soluzione ad alcuni gravi problemi delle reti distribuite su larga scala, come ad esempio quello dell'ampiezza di banda.

Un modo per tentare di risolvere il problema consiste nel cercare di ottimizzare la comunicazione per liberare il più possibile il canale, ed è proprio in questa direzione che si spinge l'utilizzo degli agenti mobili. Un agente viene infatti programmato per soddisfare un determinato compito, quindi messo in esecuzione; a questo punto, nel luogo remoto che ha prescelto grazie alla sua autonomia, l'agente è in grado di filtrare e raccogliere tutte le informazioni di cui necessita. Solo dopo aver concluso il compito per cui è stato creato, l'agente impiega una seconda volta il canale di comunicazione per portare il risultato del lavoro svolto presso la località di partenza. Pertanto una computazione anche molto complessa può in questo modo essere portata a termine con due sole trasmissioni.

1.1 Caratteristiche

Un sistema ad agenti mobili deve realizzare una certa astrazione di mondo basata su alcune caratteristiche peculiari. L'infrastruttura deve innanzitutto essere formata da un insieme di luoghi distinti che contengono risorse (entità attive e/o passive), e agenti; gli agenti dislocati nei diversi luoghi devono aver la capacità di spostarsi, interagire con le risorse e dialogare fra loro. Il tutto deve naturalmente avvenire in maniera ordinata e sicura.

Dunque un sistema ad agenti mobili deve essere innanzitutto scalabile, ovvero non deve esistere un limite al numero di luoghi che si possono aggiungere al mondo; è necessario, a supporto di tale necessità, adottare una forma di identificazione efficiente e flessibile, come un sistema di naming. Per aumentare la scalabilità del sistema dovrebbero essere inoltre presenti meccanismi in grado di gestire e coordinare una qualche forma di replicazione delle risorse.

Un sistema ad agenti deve poi essere aperto, deve cioè poter interagire con sistemi di natura diversa; in questo senso si può prendere in considerazione lo sforzo di standardizzazione operato da OMG. Strettamente correlata alla caratteristica di apertura, troviamo la caratteristica di portabilità: il sistema deve poter cioè essere installato su una qualunque macchina in rete, indipendentemente dall'architettura hardware o software.

1.2 SOMA

SOMA (Secure and Open Mobile Agent) è il sistema ad agenti mobili creato e sviluppato dal

DEIS, presso la facoltà di Ingegneria Informatica dell'Università di Bologna. È scritto interamente in Java ed è stato realizzato cercando di rispettare tutti i punti appena visti che caratterizzano un buon sistema ad agenti mobili. L'utilizzo di Java come linguaggio di programmazione permette di avere a disposizione non solo una serie di validi strumenti per la sicurezza, ma anche la completa portabilità del sistema. Tramite opportune (e forti) ipotesi di località si riesce poi a garantire una grande scalabilità del sistema.

Tali ipotesi vanno in sostanza a ricalcare la struttura interna di Internet: il *Place* rappresenta l'ambiente di esecuzione degli agenti e realizza in pratica l'astrazione di nodo, potendo contenere sia agenti che risorse. Un *Dominio* è costituito da un insieme di Place che hanno uguali caratteristiche (sotto l'aspetto fisico o logico); rappresenta cioè l'astrazione del concetto di rete locale.

Tra tutti i Place del Dominio ne esiste uno particolare con la funzione di interfaccia tra la rete locale ed il mondo esterno: è il *Place di Default*; ha una conoscenza completa dei Place del suo dominio ed una certa visione, anche solo parziale, della topologia dell'intero sistema.

Tutti i Place appartenenti allo stesso dominio si conoscono l'un l'altro; perchè ciò sia possibile ogni Place ha una tabella chiamata *PNS (Place Name Service)*, nella quale sono memorizzati i nomi dei Place appartenenti al Dominio.

I Domini sono inseriti in un albero gerarchico, proprio per permettere un'elevata scalabilità del sistema; ogni Place di Default dovrà conoscere pertanto il proprio nodo genitore ed i propri nodi figli. Inoltre un Place di Default può essere interessato ad avere visibilità su altri Place di Default; è per questo motivo che ogni Place di Default ha una tabella aggiuntiva chiamata *DNS (Domain Name Service)* in cui sono memorizzati i nomi dei Place di Default di interesse.

2 Replicazione

Per garantire una completa scalabilità del sistema, come già evidenziato in precedenza, è opportuno predisporre meccanismi in grado di garantire un certo grado di replicazione delle risorse, per permettere un corretto funzionamento del sistema anche in caso di malfunzionamenti come, ad esempio, la caduta di un nodo.

Scopo di questo progetto è la predisposizione di un meccanismo di replicazione per i Default Place, funzionalità non ancora presente in SOMA. Il problema viene affrontato sotto forti ipotesi realizzative, le quali verranno a mano a mano messe in evidenza; tuttavia è opportuno sottolineare fin da subito le principali linee guida che sono state seguite:

- Replicazione di grado due (Master-Slave)
- Ipotesi di malfunzionamento limitata alla caduta di un nodo alla volta
- Replicazione dei soli servizi di base per un Default Place

Come è noto, il modello di replicazione Master-Slave è un modello passivo, ovvero è solo il processo Master che esegue le azioni sui dati, mentre la copia passiva (lo Slave) serve solo in caso di guasto. Naturalmente lo Slave dev'essere costantemente aggiornato dal Master: in questo problema si è deciso di seguire la politica in base alla quale l'aggiornamento avviene in maniera *event-driven*, ovvero ogni volta che il Master riceve un comando volto a modificarne lo stato (intendendo con stato l'insieme dei componenti sui quali si è deciso di limitare la replicazione), deve inoltrare lo stesso tipo di modifica anche alla copia passiva. Il Master in particolare esegue l'azione di aggiornamento sullo Slave sempre dopo aver modificato il proprio stato, assicurando tempi di risposta inferiori nei confronti degli altri componenti del sistema ma minori garanzie in caso di malfunzionamento.

È compito dello Slave identificare le situazioni di guasto, tramite una costante osservazione del Master: in particolare in questa trattazione è stato messo in piedi un protocollo di verifica in base al quale lo Slave, a intervalli regolari, controlla che il Master non abbia dei malfunzionamenti, caso in cui provvede a sostituirsi ad esso comunicando poi a tutti i componenti del sistema interessati la nuova identità di quel nodo. Inoltre, mentre è attivo, lo Slave, sempre a intervalli regolari di tempo, tenta di comunicare con il Master: appena vi riesce, e ciò significa che quel Default Place è tornato operativo, gli cede nuovamente tutto il controllo e torna a ricoprire il ruolo passivo di propria competenza.

2.1 Struttura

Il sistema ad agenti SOMA definisce l'ambiente di esecuzione attraverso la classe *Environment*, la quale contiene al suo interno tutti gli oggetti che compongono un singolo Place. Fra di essi sono presenti i servizi di base, ovvero i servizi sui quali è stata concentrata l'attenzione per quanto riguarda la replicazione: DNS, PNS, Network Manager a Agent Manager, ognuno dei quali svolge un preciso compito che sarà evidenziato nel seguito.

Occorre indicare fin da subito il fatto che, nello sviluppo di tutte le parti della trattazione, si è cercato di sfruttare il più possibile la comoda infrastruttura di comunicazione offerta da SOMA basata sull'invio e ricezione di comandi tra i differenti Place del sistema.

Lo Slave è stato modellato come un nuovo Environment, ovvero un nuovo Place; esso va creato dopo il Default Place al quale fa riferimento, cioè il Master, con il quale condivide il nome del dominio e con il quale stabilisce immediatamente una connessione permanente indicando, al momento della definizione, l'indirizzo e la porta del nodo fisico col quale intende comunicare. A questo punto entra

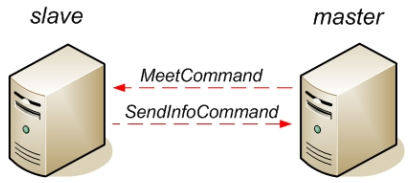

Fig. 1: *Protocollo di presentazione Master-Slave.*

in gioco un piccolo protocollo di presentazione: lo Slave invia al Master il comando *MeetCommand*, con il quale comunica la propria presenza e fa in modo che il Master salvi anch'esso una connessione per tutte le comunicazioni future.

Il Master a questo punto risponde con un *SendInfoCommand*, il quale contiene in pratica tutte le informazioni di stato del nodo sorgente: PlaceInfo (oggetto SOMA che contiene le informazioni di base di un Place quale nome, indirizzo e porta), DNS, PNS, riferimenti gerarchici (agli eventuali padre e/o nodi figli) e connessioni permanenti (saranno descritte più avanti). Tale comando, non appena giunge a destinazione presso lo Slave, salva le informazioni di stato appena viste ed attiva il thread che ha il compito di controllare, a intervalli regolari, la presenza o meno di malfunzionamenti, ovvero che il Master sia effettivamente operativo. Lo stesso thread viene poi impiegato anche quando, in seguito alla caduta del Master, lo Slave prende il controllo della situazione e tenta di riconnettersi al Default Place presso il quale è avvenuto il malfunzionamento, trasferendo ad esso il controllo non appena si accorge che la situazione è tornata alla normalità. Lo scambio ampio di informazioni di stato tra Master e Slave permette di creare una entità passiva di un Default Place in qualsiasi momento; infatti in questa maniera viene creata una copia esatta di tale nodo, per essere poi nel seguito aggiornata continuamente in maniera *event-driven*.

Vengono ora messe in evidenza, per ciascun servizio sul quale si è deciso di concentrare l'attenzione, le caratteristiche operative e le problematiche affrontate nel tentativo di garantire una efficiente e corretta replicazione.

2.2 DNS e riferimenti gerarchici

Il DNS, come menzionato in precedenza, è la tabella, presente solo presso i Default Place, che contiene i nomi dei domini del sistema ai quali quel nodo può essere interessato; esprime in pratica la visibilità che un certo dominio ha dell'intero sistema. Inoltre, come già evidenziato prima, SOMA dá la possibilità di costruire una gerarchia di domini.

2.2.1 Registrazione

Quando un Default Place intende registrarsi presso un altro Default Place, significa che desidera costituire un nuovo nodo figlio per quel Default Place, il quale di conseguenza assume il ruolo di padre; sono molteplici pertanto le operazioni a contorno di questo evento.

In primo luogo, tramite il *DomainRegisterCommand*, il Default Place che intende registrarsi invia al nodo destinazione il proprio PlaceInfo; il Place che riceve tale comando aggiorna il proprio DNS con la nuova entry (SOMA provvede anche ad aggiornare i DNS di tutti i nodi già appartenenti alla gerarchia già presente fino a quel momento) e aggiunge il nome del dominio al vettore contenente i nodi figli.

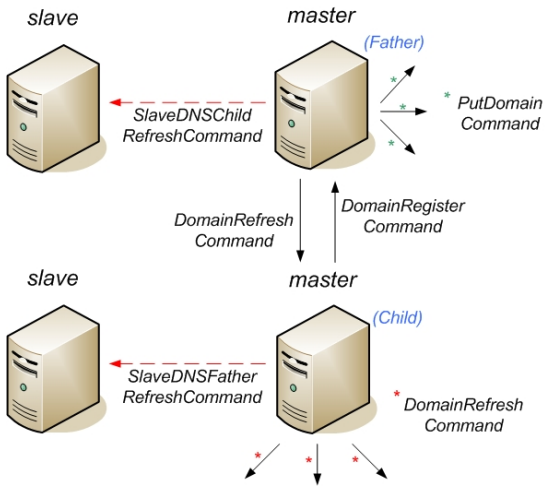

Fig. 2: *Aggiornamento DNS in seguito a registrazione.*

A questo punto viene inviato al Default Place che si è registrato il comando *DomainRefreshCommand* (verrà fra poco analizzato) e si procede all'aggiornamento dell'eventuale Slave, verso il quale viene inviato il comando *SlaveDNSChildRefreshCommand*: quest'ultimo provvede a compiere le stesse operazioni eseguite sul Master, ovvero update della tabella dei nomi (solo quella dello Slave in questo caso) e aggiornamento del vettore contenente i riferimenti ai nodi figli.

È stato appena detto che il sistema aggiorna le informazioni presso i Default Place già appartenenti alla gerarchia: ciò avviene tramite un molteplice invio di *PutDomainCommand*, il cui unico scopo è quello di inserire la nuova entry nella tabella e di propagarsi il più possibile verso nodi non ancora aggiornati. Naturalmente, nel caso in cui tali nodi siano affiancati a loro volta da una copia passiva, devono inoltrare l'aggiornamento ad essa: viene inviato in tal caso il comando *SlaveDNSTableRefreshCommand*, che ha il compito di eseguire la medesima operazione svolta sul Master presso gli Slave.

Il comando *DomainRefreshCommand*, nominato in precedenza, non è altro che una conferma dell'avvenuto successo dell'operazione, e permette di fare in modo che anche il Default Place mittente aggiorni il proprio stato, ovvero imposti il riferimento al Default Place padre ed aggiorni il proprio

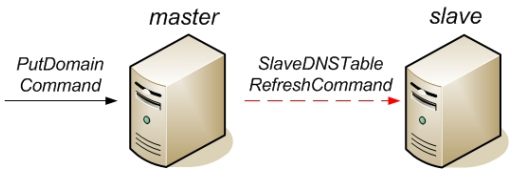

Fig. 3: *Aggiornamento DNS in seguito a PutDomainCommand.*

DNS; altra operazione che svolge è quella di inviare le medesime operazioni agli eventuali Default Place figli.

Anche in questo caso il nodo (quello da cui è partita la registrazione) deve aggiornare l'eventuale Slave: invia ad esso il comando *SlaveDNSFatherRefreshCommand*, tramite il quale, secondo la logica consueta, fa in modo che le stesse operazioni (aggiornamento DNS ed impostazione del riferimento al nodo padre) siano eseguite anche sulla copia passiva.

2.2.2 Inserimento

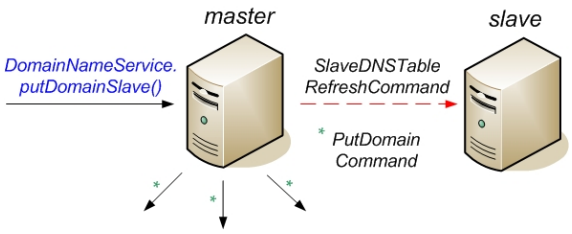

Fig. 4: *Aggiornamento DNS in seguito ad inserimento.*

Oltre all'operazione di registrazione, SOMA prevede che presso il DNS di un certo Default Place possano essere liberamente inserite delle entry; in tal caso è stato aggiunto all'oggetto *DomainNameService* il metodo *PutDomainSlave()*, il quale provvede ad inviare a tutti i domini già presenti nella gerarchia la nuova entry (tramite il comando *PutDomainCommand*), e ad aggiornare il proprio Slave tramite l'invio del comando *SlaveDNSTableRefreshCommand*, lo stesso comando che, per quanto visto prima, viene inviato anche dai Default Place ai quali viene inoltrata la nuova modifica verso la propria copia passiva, se presente.

2.2.3 Rimozione

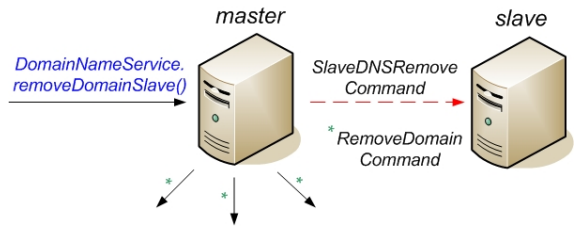

Fig. 5: *Aggiornamento DNS in seguito a rimozione.*

È anche possibile l'operazione di rimozione di una entry dalla tabella, che avviene in maniera analoga all'inserimento: all'oggetto DNS è stato aggiunto in questo caso il metodo *RemoveDomainSlave()*, il quale elimina la entry dalla tabella, elimina gli eventuali riferimenti al nodo padre o ad uno dei nodi figli (nel caso in cui il Default Place da rimuovere sia in qualche modo gerarchicamente legato al nodo sul quale ci si trova) e provvede ad informare di tale modifica tutti i nodi della gerarchia (comando *RemoveDomainCommand*). L'eventuale Slave viene in questo

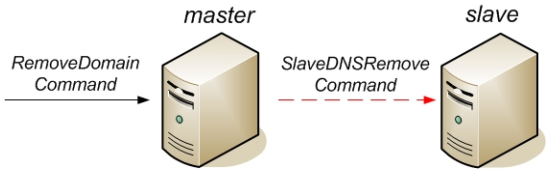

Fig. 6: *Aggiornamento DNS in seguito a RemoveDomainCommand.*

caso aggiornato dal corrispondente Default Place tramite il comando *SlaveDNSRemoveCommand*, che provvede ad eliminare la corrispondente riga della tabella e, secondo la logica consueta, anche gli eventuali riferimenti al nodo padre o ad uno dei nodi figli. Lo stesso comando viene inviato all'eventuale copia dai Default Place raggiunti dall'informazione di eliminazione.

2.3 PNS

Il PNS è invece la tabella che permette ai Place di un dominio di conoscersi a vicenda; sia il Default Place che gli altri nodi hanno la stessa identica tabella, contenente una entry per ogni entità appartenente al dominio.

2.3.1 Registrazione

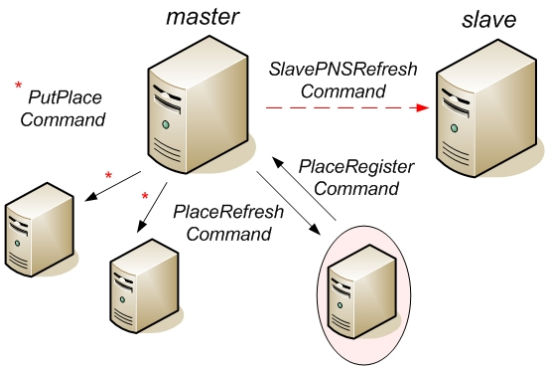

Fig. 7: *Aggiornamento PNS in seguito a registrazione.*

Quando un nuovo Place desidera registrarsi presso un dominio, invia al corrispondente Default Place il comando *PlaceRegisterCommand*, che provvede ad inserire la corrispondente entry nella tabella e a salvare la connessione con tale Place. Va ricordato a questo punto che in SOMA le connessioni tra Default Place e Place del dominio sono permanenti e sempre presenti, mentre se un Default Place desidera comunicare con una altro dominio viene creata una connessione temporanea, subito distrutta; i Default Place hanno tuttavia la possibilità, tramite un comando esplicito, di definire connessioni permanenti anche con i domini.

Il Default Place provvede ora ad inviare agli altri Place del suo dominio (già registrati) la nuova tabella; questa operazione è effettuata tramite il comando *PutPlaceCommand*. A questo punto il nodo invia, similmente a come succedeva con la gestione del DNS, il comando *PlaceRefreshCommand* al Place che aveva chiesto la registrazione, in modo tale che esso possa salvare la nuova tabella aggiornata.

In questa situazione lo Slave viene mantenuto aggiornato tramite il comando *SlavePNSRefreshCommand*, che ha il semplice scopo di aggiornare la tabella con la nuova entry; nel caso in cui il Master sia attivo non ha infatti senso mantenere con-

nessioni permanenti anche tra lo Slave ed i Place del dominio. Tale esigenza, come sarà descritto più avanti, si manifesta solamente quando lo Slave deve diventare la copia attiva in seguito alla caduta del nodo Master. Anche nel caso del PNS sono possibili ulteriori operazioni, in particolare inserimento e rimozione.

2.3.2 Inserimento

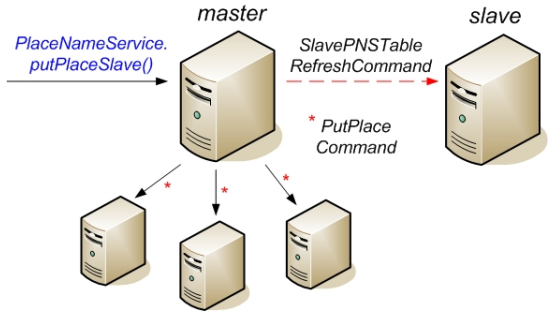

Fig. 8: *Aggiornamento PNS in seguito ad inserimento.*

Per l'inserimento è stato aggiunto all'oggetto *PlaceNameService* il metodo *PutPlaceSlave()*, che provvede ad inserire la nuova entry nella tabella e, se ci si trova su un Default Place, anche ad inviare tale modifica (tramite *PutPlaceCommand*) ai Place del dominio già presenti. Dopo aver effettuato tali operazioni aggiorna il proprio Slave (se presente) inviando ad esso il comando *SlavePNSTableRefreshCommand*.

2.3.3 Rimozione

La rimozione avviene in maniera assolutamente simmetrica: è stato aggiunto il metodo *RemovePlaceSlave()* all'oggetto PNS, responsabile di rimuovere la entry dalla tabella e di inoltrare tale modifica agli altri Place del dominio (tramite *RemovePlaceCommand*), naturalmente solo se ci si trova su un Default Place. Lo Slave viene aggiornato tramite un comando appropriato, *SlavePNSTableRemoveCommand*.

2.4 Network Manager

Il Network Manager è l'oggetto SOMA che funge da gestore unico delle comunicazioni tra due Place

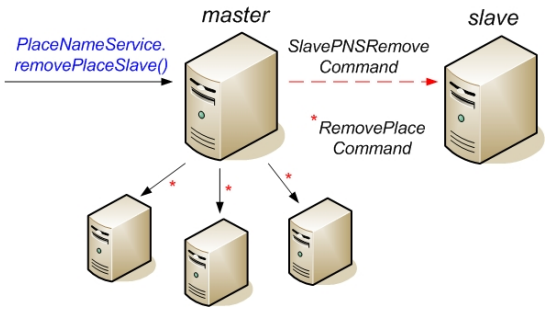

Fig. 9: *Aggiornamento PNS in seguito a rimozione.*

qualsiasi. Contiene al suo interno tre oggetti molto importanti: *ConnectionStore*, che memorizza le connessioni permanenti del Place, *ConnectionServer*, ovvero il demone che attende la richiesta di connessioni per comunicare e scambiare comandi con gli altri domini, ed infine una *Hashtable* chiamata *PermanentConnections*, all'interno della quale sono memorizzate le informazioni per gestire le connessioni permanenti inter-dominio (che un Default Place può effettuare su esplicita richiesta). In particolare SOMA crea la connessione stabile fra due Default Place, inserendola nel *ConnectionStore*, non appena il dominio che l'ha richiesta invia il primo comando verso l'altro dominio.

2.4.1 Connessioni permanenti

Per quanto già detto in merito alla gestione delle connessioni in SOMA nella sezione precedente relativa al PNS, appare abbastanza chiaro lo Slave deve mantenere aggiornate fondamentalmente solo le informazioni relative alle connessioni permanenti: le connessioni inter-dominio, come appena visto, vengono infatti stabilite *by need*, mentre quelle (permanenti) fra tutti i Place dello stesso dominio vengono automaticamente stabilite, in caso di malfunzionamento, non appena ai singoli Place viene inviata l'identità del nuovo nodo attivo, lo Slave, ovvero il nuovo Default Place al quale devono fare riferimento. Il sistema infatti fa in modo che, non appena un Default Place tenta di connettersi ad un Place del suo dominio e la connessione non è presente all'interno del *ConnectionStore*, essa venga creata e attivata automaticamente.

Per questo motivo all'interno dei metodi *start-*

PermanentConnection() e *stopPermanentConnection()* della classe *NetManager*, dopo aver rispettivamente incrementato o decrementato nella *Hashtable PermanentConnections* il numero di processi che stanno utilizzando quella connessione permanente, viene inviato allo Slave, se presente, il comando *SlavePermConnectionRefreshCommand*, il quale effettua la chiamata agli stessi metodi, con conseguente aggiornamento della *Hashtable* anche presso la copia passiva del Default Place; viene utilizzato sempre lo stesso comando, differenziando il tipo di operazione tramite un flag.

Fig. 10: *Aggiornamento informazioni relative alle connessioni permanenti.*

Le altre due operazioni da tenere in considerazione riguardano la gestione delle connessioni in seguito a malfunzionamento, e saranno pertanto analizzate in seguito: occorre infatti, in seguito alla caduta di un nodo, fare in modo che lo Slave riattivi tutte le connessioni permanenti che il Master aveva stabilito con altri domini, e inoltre che avvisi i domini che a loro volta avevano stabilito connessioni permanenti con il Master del fatto che esso è stato sostituito con un nuovo Place, lo Slave, verso il quale devono essere ora indirizzate tutte le connessioni.

2.5 Agent Manager

L'Agent Manager, anch'esso uno dei costituenti dell'*Environment* di un Place, rappresenta il gestore degli agenti contenuti nel Place. Brevemente occorre ricordare la gestione degli agenti in SOMA: un agente è un oggetto passivo e non un thread (Java non permette infatti la serializzazione dei threads), pertanto il flusso di esecuzione di un agente è affidato ad un componente chiamato *worker* (uno per ciascun agente), il quale, non appena un agente arriva in un Place, viene creato e diventa a tutti gli effetti responsabile della sua gestione, specialmente in merito alla mobilità. Una volta che l'agente lascia il Place, il corrispondente *worker* viene distrutto.

Un agente comunica poi con il Place in cui si trova attraverso un oggetto di classe *AgentSystem*, il cui riferimento è contenuto all'interno di *AgentManager*; tale variabile viene aggiornata dal sistema dopo ogni migrazione dell'agente, e rappresenta l'unico modo effettivo di interazione verso il mondo esterno.

2.5.1 Agenti e replicazione

Uno dei vantaggi, molto significativo, che si può ottenere avendo a disposizione uno Slave, è quello consistente nella possibilità di poter riprendere l'esecuzione degli agenti sulla copia passiva in seguito alla caduta del nodo sul quale risiedeva il Default Place Master. Il problema è stato affrontato seguendo questa idea di base: quando un agente migra presso un (Default) Place, dopo la creazione del corrispondente *worker* esso viene inviato anche presso lo Slave, se presente, tramite il comando *SlaveTransportCommand*. Giunto presso

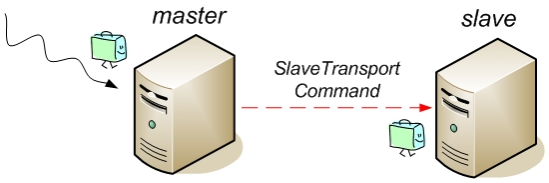

Fig. 11: *Invio dell'agente allo Slave, che non lo mette in esecuzione.*

questo nodo, viene anche in questo caso creato un *worker*, il quale però non viene attivato, rimane cioè in uno stato idle. L'agente comincia invece la propria esecuzione presso il nodo Master, ed in tal caso si prospettano due possibilità: l'agente termina la propria esecuzione su quel nodo, cioè migra altrove oppure muore, e in tal caso occorre semplicemente rimuovere il *worker* precedentemente creato (e mai attivato!) dallo Slave, oppure il Master può subire un malfunzionamento e di conseguenza l'esecuzione dell'agente viene interrotta. In quest'ultimo caso, al momento della rilevazione del guasto, lo Slave deve provvedere a mettere in esecuzione tutti i *worker* degli agenti in suo possesso, i quali non sono stati cancellati e pertanto significa che gli agenti ad essi associati

non avevano terminato correttamente l'esecuzione. Una volta terminata l'esecuzione presso uno Slave, l'agente può poi proseguire normalmente il proprio cammino. La rimozione del *worker* presso lo Slave avviene tramite il comando *RemoveAgentCommand*, inviato dal Master allo Slave non appena l'agente termina la propria attività su quel Place.

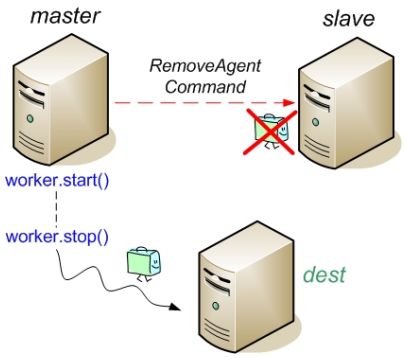

Fig. 12: *Comportamento del sistema nel caso in cui l'agente termini la propria esecuzione sul Master.*

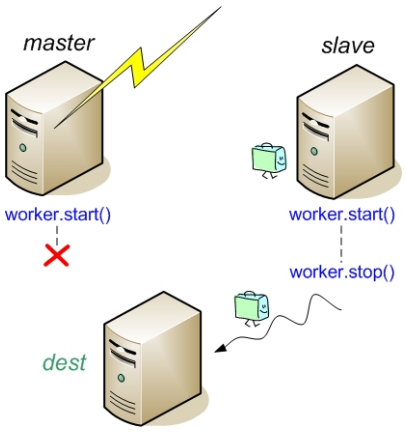

Fig. 13: *Comportamento del sistema in caso di malfunzionamento: l'esecuzione dell'agente riprende sullo Slave dall'inizio.*

È inoltre possibile che un nodo previsto all'interno del cammino di un agente cada prima che l'agente migri effettivamente su di esso; questo non costituisce un problema, in quanto il comando *AgentTransportCommand*, responsabile del trasporto dell'agente da un Place ad un altro, si basa sugli indirizzi contenuti in quel momento all'interno del DNS del Place sul quale l'agente si trova. Pertanto se un nodo cade e diventa attivo il suo Slave, questa informazione è già contenuta all'interno del DNS, e l'agente viene indirizzato automaticamente verso lo Slave (in quel momento attivo) del nodo destinazione.

Altro caso possibile si presenta quando l'agente sta eseguendo presso lo Slave e contemporaneamente il nodo Master torna attivo (secondo le specifiche quest'ultimo dovrebbe ora riprendere il pieno controllo della situazione); per evitare una inutile migrazione (l'agente dovrebbe essere spostato sul Master e ricominciare qui da capo la propria esecuzione), l'agente non viene reso partecipe di tale nuova situazione, e finisce pertanto le proprie operazioni presso lo Slave per poter poi continuare regolarmente il proprio corso. Ciò è possibile in quanto lo Slave a tutti gli effetti è in grado di sostituirsi pienamente al Master in qualsiasi momento, e quindi di trattare nella maniera opportuna gli agenti attraverso il suo *AgentManager*.

2.6 In caso di guasto

Come è già stato evidenziato più volte, per ipotesi lo Slave controlla il Master per verificarne il corretto funzionamento, e, in caso ciò non fosse vero, prende il suo posto. Non appena però il Master torna attivo, e il controllo di questa condizione spetta sempre allo Slave, quest'ultimo deve mettersi nuovamente da parte per costituire una semplice copia passiva.

2.6.1 Caduta del Master

Il thread, in esecuzione presso lo Slave, che si occupa di verificare il corretto funzionamento del Master, interroga quest'ultimo ogni 30 secondi; tale durata è per ipotesi fissa e riguarda ogni nodo, ma si potrebbe pensare di renderla variabile dando ad esempio la possibilità all'utente di specificare il dato al momento della creazione dello Slave. Ad ogni scadenza temporale lo Slave tenta di inviare al Master un comando vuoto, chiamato *ReqAliveCommand*, il quale, se inviato correttamente, dá garanzia sul fatto che il Master sia effettivamente attivo; se invece risulta impossibile inviare tale comando, significa che il Master non è più

presente, probabilmente in seguito alla caduta del nodo corrispondente. Viene in tal caso lanciata un'eccezione, alla ricezione della quale lo Slave, che fino a quel momento si è mantenuto completamente sincronizzato al Master, provvede ad effettuare tutte le operazioni necessarie per sostituirsi al Default Place del quale costituisce la copia:

1. Inserimento del proprio PlaceInfo nel DNS tramite *PutDomainCommand*
2. Inserimento del proprio PlaceInfo nel PNS tramite *PutPlaceCommand*
3. Aggiornamento delle connessioni permanenti che il Master aveva stabilito con altri domini
4. Aggiornamento delle connessioni permanenti che altri domini avevano stabilito con il Master
5. Esecuzione di tutti gli agenti che eventualmente erano stati interrotti in seguito alla caduta del nodo

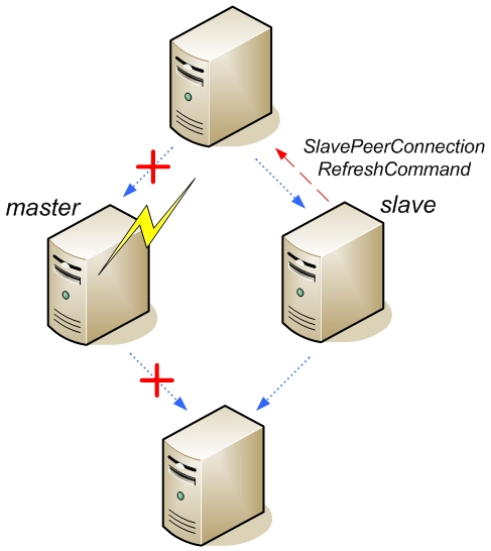

Fig. 14: *Aggiornamento connessioni permanenti da e verso lo Slave.*

L'invio del *PutDomainCommand* assicura, per quanto visto nella prima parte della trattazione, un aggiornamento completo anche di tutti gli altri domini della gerarchia; allo stesso modo *PutPlaceCommand* propaga la nuova identità del nodo a tutti i Place di quel dominio e, per quanto già detto in proposito, aggiorna anche le connessioni permanenti con essi. L'ultimo punto evidenzia quanto appena detto in merito agli agenti: al momento dell'attivazione, lo Slave deve provvedere ad eseguire tutti i *worker* degli agenti che non erano riusciti a completare la propria esecuzione, riattivandoli dall'inizio.

Le connessioni permanenti che, invece, il Master aveva stabilito con altri domini (su esplicita richiesta, come già detto più volte), sono aggiornate inviando ad ogni Place contenuto all'interno di *PermanentConnections* (mantenuta sincronizzata con quella del Master) un comando vuoto; SOMA infatti, similmente a come avviene per le connessioni con i Place all'interno del dominio, al momento di inviare un comando verso un Default Place, se si accorge che la corrispondente connessione permanente con esso non è attiva, la crea e la salva all'interno del *ConnectionStore*.

Tuttavia ciò non basta, in quanto occorre anche avvisare tutti i Default Place del fatto che ora lo Slave è il nodo attivo, permettendo ad essi di aggiornare l'eventuale connessione permanente che avevano stabilito con il corrispondente Master. Per fare ciò viene inviato ad essi il comando *SlavePeerConnectionRefreshCommand*, che, una volta giunto presso il Place di destinazione, controlla se erano state stabilite o meno delle connessioni con lo Slave scansionando la *Hashtable* e confrontando l'identificativo dello Slave (uguale a quello del Master per ipotesi) con quelli presenti; se viene trovata una corrispondenza, viene inviato a tale Place il solito comando vuoto, tramite il quale SOMA provvede a riavviare la connessione.

2.6.2 Riattivazione del Master

Lo stesso thread che si occupa di rilevare malfunzionamenti presso il Master, viene impiegato anche per verificare l'eventuale istante al quale il Master torna nuovamente attivo; il controllo viene effettuato sempre ogni 30 secondi, alla scadenza dei quali lo Slave tenta di connettersi al nodo caduto. Non appena vi riesce, cioè non viene lanciata alcuna eccezione, provvede a fermare tutte le connessioni permanenti attive e ad inviare al suo Master il comando *ActiveMasterCommand*, che è molto simile al comando *SendInfoCommand* inviato dal Master allo Slave in fase di presentazione. Esso contiene infatti tutte le informazioni di stato necessarie: DNS,

PNS, riferimenti gerarchici e connessioni permanenti; una volta giunto a destinazione, dopo aver impostato il nuovo stato, il comando provvede poi ad eseguire le stesse identiche operazioni descritte nella fase precedente eccetto l'ultima, cioè il riavvio degli agenti. Per ipotesi infatti gli eventuali agenti in esecuzione sullo Slave terminano in quel Place il loro flusso operativo.

Fig. 15: *Riabilitazione del Master da parte dello Slave.*

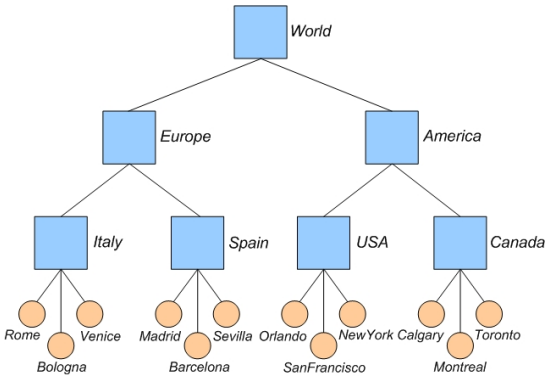

Fig. 16: *Gerarchia SOMA utilizzata per i test.*

3 Test

I test sul sistema, appoggiato a Java 2 SDK 1.4.2 e corredato dell'infrastruttura di replicazione appena spiegata, sono stati effettuati lavorando dal menu testuale di SOMA; nel tentativo di costruire un ambiente che fosse il più significativo possibile, è stata costruita una gerarchia di domini, visibili in figura. Poichè gli scopi di questa trattazione erano esclusivamente concentrati sulla replicazione Master-Slave di un Default Place (e sulle relative operazioni di coordinamento a contorno), è stato deciso di non implementare anche un servizio che sfruttasse la tecnologia ad agenti mobili, ma di limitarsi ad effettuare la verifica della correttezza di quanto costruito tramite l'impiego di un semplice agente, in grado comunque di mettere in evidenza la corretta risposta del sistema in seguito a malfunzionamenti.

Dopo aver effettuato tutti i test possibili relativamente ad operazioni su DNS, PNS e in merito alla creazione e rimozione di connessioni permanenti, sono state simulate diverse cadute di differenti nodi, ottenendo in tutti i casi una risposta del sistema coerente a quanto desiderato. È stato quindi poi creato l'agente *HelloAgent*, il quale si limita ad una funzione semplicissima: recupera dal DNS del dominio all'interno del quale viene lanciato i riferimenti a tutti gli altri domini, tramite i quali costruisce il proprio cammino. Non appena, procedendo sul suo percorso, si trova in un Default Place, recupera anche i riferimenti agli eventuali Place di quel dominio, che visita immediatamente; l'unica operazione che esegue presso ciascun Place è quella di stampare a video un messaggio. Durante il cammino dell'agente, reso realisticamente più lungo grazie all'inserimento di temporizzazioni, sono stati simulate le situazioni più delicate: esecuzione normale sul Master, caduta di un nodo mentre l'agente è in esecuzione su di esso, caduta di un nodo previsto sul percorso dell'agente ma non ancora visitato, ed infine riattivazione del Master mentre l'agente è in esecuzione sullo Slave. Anche in tutti questi casi il sistema ha risposto in maniera efficiente e corretta in base alle specifiche.

4 Conclusioni

Grazie allo svolgimento di questa trattazione è stato possibile approfondire due aspetti molto importanti nell'ambito delle Reti di Calcolatori: innanzitutto lo studio del funzionamento di un sistema concreto ad agenti mobili, SOMA, che ha messo in luce le diverse caratteristiche e problematiche che devono essere affrontate nella costruzione di una infrastruttura di questo tipo. E poi è stato possibile entrare in stretto contatto con una reale esigenza di replicazione, affiancata da tutta una relativa serie di operazioni a contorno: scelta dei componenti che devono essere soggetti a replicazione, creazione di protocolli per la rilevazione e l'aggiornamento dello stato, ed in generale tutte le

azioni atte a coordinare le diverse entità coinvolte.

Tutti i test hanno dato un esito positivo, anche se sono state impiegate delle strutture piuttosto semplici; tuttavia lo scopo di questa trattazione era quello di intervenire ad un livello piuttosto basso, onde estendere il sistema già presente con una nuova ed importante funzionalità. Si potrebbe pensare infatti di effettuare altri test sull'infrastruttura di replicazione, magari creando servizi maggiormente complessi basati su più agenti; il sistema dovrebbe essere in grado di affrontare anche tali situazioni, sebbene sempre sottostanti alle forti ipotesi che, per ovvi motivi realizzativi, sono state effettuate fin dall'inizio.

Contents