

Università degli Studi di Roma Tor Vergata



Facoltà di Ingegneria

Corso di Informatica Mobile

**SLASH**

Professore:  
Vincenzo Grassi

Studenti:  
Simone Notargiacomo  
Lorenzo Tavernese  
Ibrahim Khalili

Anno Accademico 2007-2008

# Indice

<b>Abstract</b>	<b>v</b>
<b>1 Introduzione</b>	<b>1</b>
1.1 Specifiche progetto . . . . .	1
1.1.1 Logica applicazione . . . . .	1
1.1.2 Ambiente d'uso . . . . .	2
1.1.3 Lavoro progettuale . . . . .	3
1.2 Distributed Shared Memory . . . . .	3
1.3 Mobile Agent . . . . .	4
1.4 JADE . . . . .	5
<b>2 Architettura</b>	<b>8</b>
2.1 Sistema . . . . .	8
2.1.1 Composizione Nodi . . . . .	8
2.1.2 Composizione Rete . . . . .	10
2.2 Funzionamento . . . . .	11
2.2.1 Scenario d'uso . . . . .	11
2.2.2 Distributed Shared Memory . . . . .	12
2.2.3 Context Manager . . . . .	13
2.2.4 SLA Checker . . . . .	14
<b>3 Dsm</b>	<b>16</b>
3.1 DsmData . . . . .	16
3.2 DsmClient . . . . .	17

3.3	DsmServer . . . . .	18
<b>4</b>	<b>Modello Matematico</b>	<b>19</b>
4.1	Valori risorse . . . . .	19
4.1.1	Cpu . . . . .	20
4.1.2	Ram . . . . .	20
4.1.3	Memory . . . . .	20
4.1.4	Energy . . . . .	21
4.1.5	Latency . . . . .	22
4.1.6	Reliability . . . . .	22
4.1.7	ReqInterval . . . . .	22
4.2	SLA Checker . . . . .	23
4.2.1	Algoritmo controllo . . . . .	23
<b>5</b>	<b>Implementazione</b>	<b>27</b>
<b>6</b>	<b>Simulazione</b>	<b>28</b>
6.1	Esecuzione . . . . .	28
6.1.1	Scenario . . . . .	29
6.1.2	Generazione contratti SLA . . . . .	29
6.1.3	Migrazione . . . . .	29
6.2	Analisi dati . . . . .	33
6.2.1	Indice . . . . .	33
6.2.2	Cpu . . . . .	36
6.2.3	Ram . . . . .	38
6.2.4	Memory . . . . .	41
6.2.5	Energy . . . . .	43
6.2.6	Latency . . . . .	46
6.2.7	Reliability . . . . .	48
6.2.8	ReqInterval . . . . .	50
<b>7</b>	<b>Conclusioni e sviluppi futuri</b>	<b>51</b>

<b>8</b>	<b>Appendice</b>	<b>52</b>
8.1	SLARequesterBehaviour.java . . . . .	52
8.2	SLAReqReceiverBehaviour.java . . . . .	53
8.3	ContextProducerBehaviour.java . . . . .	54
8.4	ViolationConsumerBehaviour.java . . . . .	55
8.5	RmConsumerBehaviour.java . . . . .	55
8.6	ContextManagerAgent.java . . . . .	57
8.7	CpuResourceBehaviour.java . . . . .	58
8.8	RamResourceBehaviour.java . . . . .	59
8.9	MemoryResourceBehaviour.java . . . . .	60
8.10	EnergyResourceBehaviour.java . . . . .	60
8.11	LatencyResourceBehaviour.java . . . . .	61
8.12	ReliabilityResourceBehaviour.java . . . . .	62
8.13	ReqIntervalResourceBehaviour.java . . . . .	63
8.14	RamRmAgent.java . . . . .	64
8.15	MemoryRmAgent.java . . . . .	64
8.16	EnergyRmAgent.java . . . . .	64
8.17	CpuRmAgent.java . . . . .	65
8.18	LatencyRmAgent.java . . . . .	65
8.19	ReliabilityRmAgent.java . . . . .	65
8.20	ReqIntervalRmAgent.java . . . . .	66
8.21	AgentLauncher.java . . . . .	66
8.22	SLAReceiverBehaviour.java . . . . .	68
8.23	SLACheckerBehaviour.java . . . . .	69
8.24	SLAStarterBehaviour.java . . . . .	72
8.25	ContextConsumerBehaviour.java . . . . .	73
8.26	MigrationUtil.java . . . . .	74
8.27	SLACheckerAgent.java . . . . .	74
8.28	ReqIntervalBehaviour.java . . . . .	76
8.29	NotifyReceiverBehaviour.java . . . . .	77
8.30	LatencyBehaviour.java . . . . .	78

## INDICE

---

8.31	RamBehaviour.java . . . . .	79
8.32	CpuBehaviour.java . . . . .	80
8.33	MemoryBehaviour.java . . . . .	81
8.34	EnergyBehaviour.java . . . . .	82
8.35	ReliabilityBehaviour.java . . . . .	84
8.36	CpuAgent.java . . . . .	85
8.37	ResourceAgent.java . . . . .	86
8.38	MemoryAgent.java . . . . .	86
8.39	EnergyAgent.java . . . . .	87
8.40	ReliabilityAgent.java . . . . .	88
8.41	ReqIntervalAgent.java . . . . .	89
8.42	LatencyAgent.java . . . . .	90
8.43	RamAgent.java . . . . .	91
8.44	DFUtil.java . . . . .	92
8.45	Context.java . . . . .	94
8.46	Notify.java . . . . .	99
8.47	SLAContract.java . . . . .	100
8.48	DataWriter.java . . . . .	101
8.49	PropertiesReader.java . . . . .	103
8.50	DsmServerBehaviour.java . . . . .	104
8.51	DsmDataManager.java . . . . .	107
8.52	Tuple.java . . . . .	109
8.53	DsmServerAgent.java . . . . .	111
8.54	DsmClient.java . . . . .	111

# Abstract

Nel mondo dell'Informatica si è diffuso da molto tempo il bisogno di sviluppare applicazioni per dispositivi mobili, quali notebook, smartphone, tablet ed altro ancora; in particolare si è raggiunta la necessità di sviluppare applicazioni distribuite per dispositivi mobili, ovvero il *Mobile Computing*. Sono nate molte nuove tecniche che hanno apportato migliorie alla comunicazione tra dispositivi, come la stipulazione di *SLA* (Service Level Agreement: Contratti basati sul livello di servizio) tra richiedenti e fornitori di servizi, ed infine le *dsm* (Distributed Shared Memory). Durante il corso di Informatica Mobile è stato richiesto di realizzare un sistema che simulasse il comportamento di alcuni richiedenti e fornitori dopo aver stipulato un contratto sulla qualità del servizio, sfruttando *dsm* per la comunicazione dei dati. In questa relazione verranno trattate le problematiche incontrate nella progettazione del sistema, ed in particolare tutte le scelte implementative effettuate, nonché i modelli matematici ed empirici. Inoltre verranno presentati anche dei test di esecuzione con i relativi dati sulle performance; infine si riporteranno le conclusioni a cui si è giunti ed i possibili sviluppi futuri.

# Capitolo 1

## Introduzione

Per comprendere a fondo l'entità di tale progetto si riportano di seguito le specifiche del progetto, in quanto consente al lettore di entrare nell'ottica del problema, inoltre si fornirà una breve panoramica relativa al paradigma DSM, agli agenti mobili[4] e al framework JADE.

### 1.1 Specifiche progetto

#### 1.1.1 Logica applicazione

Si richiede di progettare una architettura di supporto al monitoraggio e controllo di SLA in ambiente (possibilmente) mobile. L'architettura del servizio è basata sulla definizione di un certo numero di componenti “logici”: SLA Checker (SC), Context Manager (CM), Resource Monitor (RM). Tali entità interagiscono tra loro unicamente tramite il meccanismo DSM (tuple space). Il ruolo di tali entità viene descritto come segue:

**SLAchecker (SC):** data una coppia fornitore/richiedente servizio, che ha stipulato un SLA, lo SC ha il compito di controllare il rispetto dei parametri del contratto sia da parte del fornitore che del richiedente, e segnalare eventuali violazioni ad entrambi. A questo scopo, SC raccoglie informazioni fornite da opportuni componenti di tipo Monitor

presenti sia sul nodo del fornitore che del richiedente, relative a (per esempio):

- tempo di risposta osservato per una richiesta;
- affidabilità (completamento con successo) di una richiesta;
- intervallo di tempo tra due richieste consecutive.

I dati “grezzi” ricevuti dai componenti di monitoraggio vengono elaborati da SC per calcolare i valori degli indici di interesse.

**Context Manager (CM):** è un componente associato a un particolare nodo di elaborazione e il suo ruolo è quello di fornire informazioni su vari tipi parametri che caratterizzano il contesto di esecuzione di componenti presenti su quel nodo, p.es.:

- utilizzazione cpu;
- RAM disponibile;
- memoria stabile (disco, o altro) disponibile;
- tipo di rete e banda disponibile;
- energia disponibile.

**Resource Monitor (RM):** un componente di questo tipo fornisce le informazioni relative a una delle risorse elencate sopra.

### 1.1.2 Ambiente d’uso

L’ambiente in cui si immagina che il servizio di controllo di SLA venga realizzato è costituito, in generale, da una molteplicità di nodi (fissi o mobili) con vari livelli di disponibilità di risorse interne (memoria, cpu, ecc.), connessi tra loro da infrastrutture di comunicazione di varia qualità. Su tali nodi sono in esecuzione componenti che offrono/richiedono servizi. Ogni volta che una



coppia fornitore/richiedente[2][5]<sup>1</sup> stipula un SLA, il controllo di questo SLA viene affidato a un componente SC.

### 1.1.3 Lavoro progettuale

Si richiede di progettare e realizzare, utilizzando la piattaforma JADE (<http://jade.tilab.com>), l'architettura indicata nella sezione precedente. In particolare, occorre definire una localizzazione dei componenti e organizzazione del modello DSM (basato sulla realizzazione di uno o più tuple space) che sia adeguata alla esecuzione del servizio di controllo SLA in un ambiente possibilmente mobile, caratterizzato da possibile scarsità di risorse per i componenti in esecuzione su determinati nodi. Il livello di adeguatezza andrà valutato rispetto alla capacità di ottimizzare misure di prestazione quali:

- traffico generato su rete;
- consumo di energia da parte di nodi mobili;
- carico computazionale/di memorizzazione per nodi mobili;

tenendo anche conto del fatto che il contesto (disponibilità di risorse) in cui opera il servizio di controllo SLA può variare nel tempo, per esempio per effetto della mobilità di alcuni nodi.

## 1.2 Distributed Shared Memory

Il paradigma DSM[2] fornisce agli host in un sistema distribuito la vista di uno spazio comune condiviso attraverso spazi di indirizzamento disgiunti, in cui la sincronizzazione e la comunicazione fra i partecipanti avvengono tramite operazioni sui dati comuni. La nozione di *tuple space* è stato originariamente integrato in Linda, e fornisce una semplice e potente astrazione per accedere alla memoria condivisa. Un *tuple space* è composto di una collezione di tuple

---

<sup>1</sup>non è un vero sistema publisher/subscriber ma una simulazione necessaria a rendere l'idea del contesto

ordinate, accessibili in egual modo da tutti gli host del sistema distribuito. La comunicazione tra hosts avviene tramite l'inserimento/rimozione di tuple nel/da *tuple space*. Possono essere eseguite tre operazioni di base: *out()* per esportare una tupla nel *tuple space*, *in()* per importare (e rimuovere) una tupla e *read()* per leggere, senza rimuovere, una tupla. Il modello di interazione offre disaccoppiamento sia spaziale che temporale, pochè consumatore e produttore non hanno bisogno di conoscersi e il creatore di una tupla non ha bisogno di sapere l'uso che verrà fatto di tale tupla. Nonostante tutto, non si ha un disaccoppiamento da un punto di vista della sincronizzazione dal lato del consumatore.

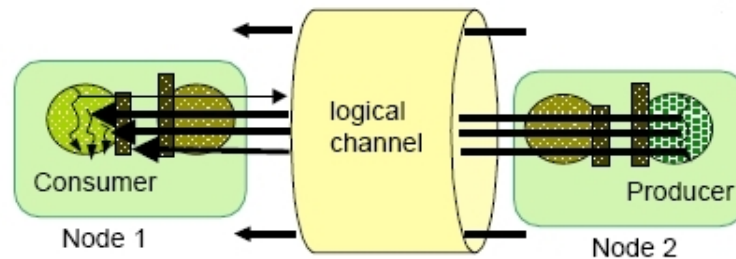


Figura 1.1: DSM

### 1.3 Mobile Agent

Un agente mobile[3] è un componente software in grado di traserirsi su nodi remoti di una rete e di interagire con le risorse nei nodi visitati, scoprendo i servizi offerti. Dalla definizione fornita si arguisce che l'Agente deve possedere un certo grado di *intelligenza*, sia per le decisioni che deve prendere una volta spedito, sia per la capacità di memorizzare i risultati ottenuti su ciascun nodo (capacità di mantenere ed aggiornare il suo stato). La possibilità per un agente di muoversi e di conseguenza il comportamento che questo può assumere si deve basare sugli obiettivi definiti nel suo stato; tale concetto di mobilità unito a quello di agente ha permesso di conferire a questo componente la capacità di cooperare al fine di poter risolvere un dato problema. Il

processo di trasferimento del codice e del relativo stato può avvenire secondo due modalità differenti:

- Clonazione[3]: il codice e lo stato vengono duplicati nel nodo di arrivo senza rimuoverli nel nodo di partenza
- Migrazione[3]: il codice e lo stato vengono copiati nel nuovo nodo e rimossi dal vecchio.

Terminato il processo di migrazione l'agente deve essere in grado di interagire con le risorse locali al fine di usufruire dei servizi offerti dal nodo ospite. Il meccanismo di interazione con le varie risorse presenti viene facilitato grazie all'utilizzo della tecnologia ad oggetti, al contrario del procedimento di *discovery* dei servizi, il quale prevede un modello di *introspezione* degli oggetti, che mantenga le informazioni accessibili dinamicamente. Si può quindi intuire che se da un punto di vista teorico si ha grande convenienza nell'utilizzo di Agenti, in pratica non sempre risulta possibile usufruire di tali vantaggi. E' importante evitare di cadere nella tentazione assolutista con la quale si cerca di applicare una tecnologia in ogni possibile dominio o circostanza. Possiamo sicuramente affermare che la soluzione ad Agenti può essere più conveniente rispetto ad altri approcci a seconda di circostanze quali domini, infrastrutture di rete, compatibilità con altre tecnologie di contorno.

### 1.4 JADE

JADE è un framework interamente realizzato in Java, che permette di semplificare l'implementazione di *Multi Agent Systems* (MAS), associando diversi container agli agenti, i quali hanno la possibilità di comunicare sulla stessa o su piattaforme differenti. I vari containers possono essere raggruppati formando così una *platform*.

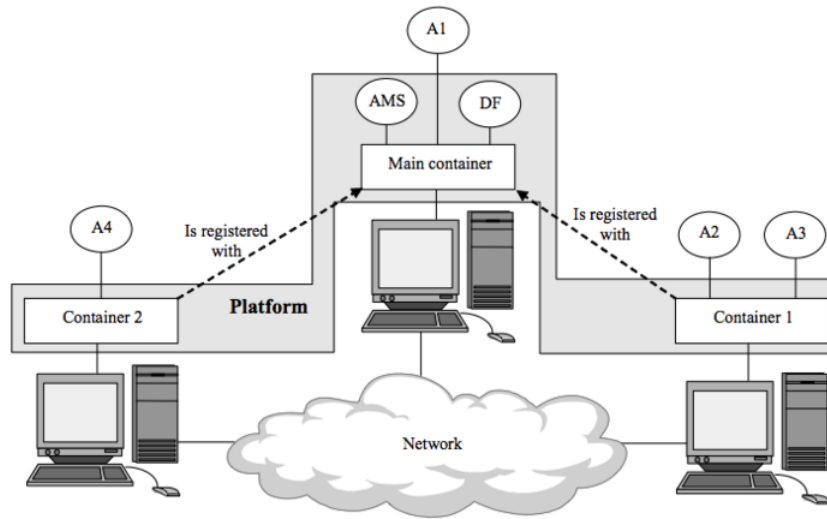


Figura 1.2: architettura di JADE

Ogni platform deve avere un *Main Container*, il quale possiede due agenti specializzati chiamati rispettivamente *AMS agent* e *DF agent*:

- L'*AMS* (Agent Management System) ha il compito di svolgere la funzione di supervisore controllando l'accesso e l'uso della piattaforma da parte degli agenti. Inoltre mantiene un registro degli identificatori degli agenti (AID) e del loro stato, in quanto ogni agente è obbligato a registrarsi per ottenere un AID valido .
- Il *DF* (Directory Facilitator) ha il compito di implementare un servizio di pagine gialle che pubblicizza i servizi degli agenti della piattaforma in modo che altri agenti che richiedono tali servizi possono trovarli.

Le varie operazioni che gli agenti sono in grado di compiere vengono definite attraverso oggetti di tipo *Behaviour*, i quali forniscono l'implementazione dei task richiesti. Esistono diversi tipi di behaviour predefiniti (composite behaviour, sequential behaviour, parallel behaviour, sender behaviour), tuttavia è possibile definire behaviour personalizzati a seconda della complessità del

task richiesto. Il modello di comunicazione utilizzato dagli agenti per interagire si basa sullo standard ACL (Agent Communication Language), nel quale i messaggi sono scambiati in modo asincrono. La struttura dei vari messaggi viene definita all'interno della classe java (`jade.lang.acl.ACLMessage`) fornita dal framework stesso. Tale classe fornisce i metodi per impostare e ottenere i valori dei campi definiti da FIPA (Foundation for Intelligent Physical Agent) per l' ACL.

# Capitolo 2

## Architettura

Nella specifica del problema (rif. 1) è stato riportato il funzionamento del sistema da realizzare e i relativi componenti necessari. Per adempiere alle richieste della specifica si è deciso di sviluppare l'applicazione dando priorità ai punti fondamentali, dopodiché sono stati trattati gli aspetti secondari come la taratura dei parametri e i meccanismi di richiesta dei servizi. Nella prossima sezione verrà riportata l'architettura del sistema.

### 2.1 Sistema

#### 2.1.1 Composizione Nodi

Tutta l'applicazione è stata progettata e sviluppata considerando la presenza di molteplici nodi richiedenti e fornitori, per rendere possibile una simulazione più reale e complessa. Inizialmente sono state prese delle decisioni inerenti la composizione generale di ogni nodo della nostra rete, in particolare si è deciso di supportare nodi di due tipi:

**wired:** nodo fisso collegato tramite cavo;

**wireless:** nodo mobile collegato tramite *wireless*.

Effettuando tale scelta si sono scatenate un'altra serie di decisioni attinenti l'hardware dei nodi, ovvero l'energia, la memoria ram, il disco e il carico

della cpu. Tali componenti sono molto dipendenti dal tipo di collegamento del nodo, in quanto un link di tipo wireless ha bisogno di maggiori calcoli e quindi un utilizzo di energia maggiore.

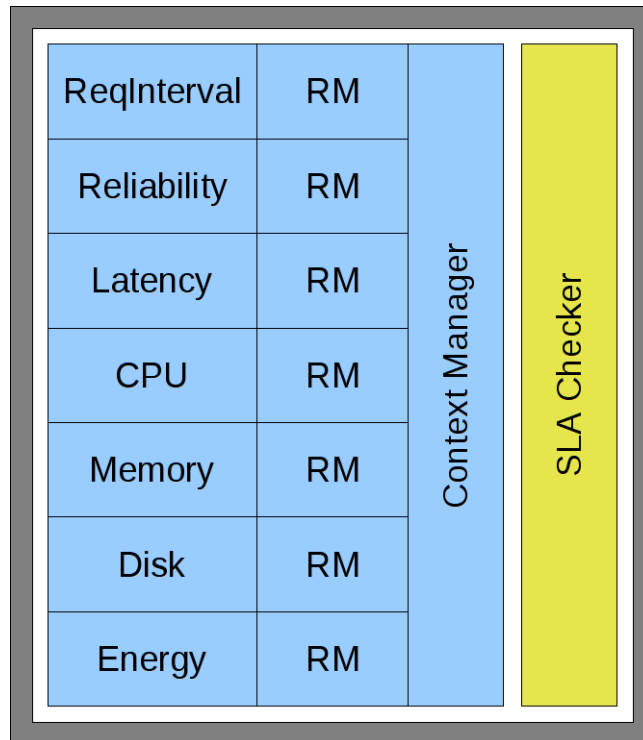


Figura 2.1: Componenti Nodo

Dalla figura 2.1 si può notare come è strutturato un singolo nodo, in particolare si può notare come i componenti da monitorare (cpu, memory, disk, energy, latency, reliability, reqInterval) siano in stretto contatto uno a uno con un *Resource Monitor*. I *Resource Monitor* immettono informazioni dei componenti monitorati sul dsm. Il *Context Manager* si occupa di raccogliere tali dati e raggrupparli in un solo oggetto che viene a sua volta immesso sul dsm. Lo *SLA Checker* (o *SC*), si trova in genere a contatto con il *CM* questo perchè raccoglie principalmente le informazioni sul contesto dei vari nodi. Tutte queste parti elencate saranno chiamate d'ora in poi *Agenti*, considerando che ci si trova in ambiente di programmazione ad agenti (ovvero

*Jade*). Gli agenti delle risorse sono stati realizzati per consentire una simulazione più realistica, ovvero ogni agente risorsa non fa altro che generare un valore di utilizzazione basato su vari parametri (ved. cap 4).

### 2.1.2 Composizione Rete

La rete che si è deciso di creare è formata da tanti di questi nodi, sia richiedenti che fornitori. Nello specifico ogni nodo fornitore si occupa di fornire un servizio generico, il quale viene registrato nelle pagine gialle del sistema. Ogni richiedente, invece, può richiedere il servizio ad uno qualsiasi dei fornitori disponibili. Questa operazione è stata resa possibile per fare in modo che la simulazione si attenesse ad un tipico caso reale. Per quanto riguarda lo *SLA Checker* si è progettato il sistema considerando che tale agente dovrebbe *migrare* da un nodo all'altro in base alle condizioni del contesto. Un esempio della rete in questione con 3 nodi può essere visto in figura 2.2. Si può notare che lo *SC* si trova solo su uno dei nodi della rete in quanto deve poter migrare fra di loro.

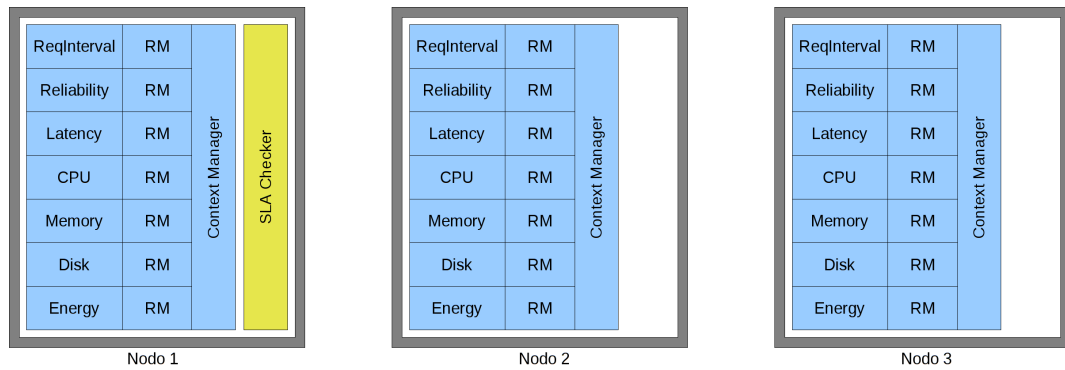


Figura 2.2: Esempio di rete con 3 nodi



## 2.2 Funzionamento

Per spiegare il funzionamento del sistema in questione si è deciso di definire uno scenario d'uso e quindi descrivere il comportamento dei relativi nodi.

### 2.2.1 Scenario d'uso

Un tipico scenario d'uso potrebbe essere una rete con 3 nodi di cui:

- 1 nodo fornitore;
- 2 nodi richiedenti;

in particolare ci troviamo in una situazione in cui ognuno dei due nodi richiede un servizio che si attenga al contratto prestabilito con il nodo fornitore. Ogni contratto contiene le seguenti informazioni:

**Publisher:** fornitore del servizio;

**Subscriber:** richiedente del servizio;

**ReqInterval:** intervallo di tempo tra le richieste del richiedente;

**Latency:** tempo impiegato per espletare il servizio;

**Reliability:** affidabilità di servizio da parte del fornitore.

Tali informazioni servono per fare in modo che sia il fornitore che il richiedente facciamo il possibile per attenersi ai valori specificati nel contratto. Su uno dei nodi del sistema è presente lo *SLAChecker* che si occupa di controllare la validità di tutti i contratti stipulati; nel caso in cui le condizioni di un contratto vengano violate da uno dei due nodi allora lo *SC* informerà immediatamente entrambi i nodi di tale condizione. Tutti i componenti dei nodi sono fortemente dipendenti dalla presenza dello *SC*, in quanto comporta un aumento oneroso in termini di calcoli. A causa delle risorse limitate di alcuni nodi (come l'energia) è necessario effettuare un controllo sullo stato attuale dei componenti dei nodi per evitare di sovraccaricarlo, in modo tale che in

caso di necessità lo *SC* migri su un nodo con condizioni di carico migliori. La selezione del nodo migliore viene fatta utilizzando una specifica politica di selezione. Un esempio di migrazione può essere visto nelle due figure seguenti, in cui nel primo caso lo *SC* si trova sul nodo 1, mentre nel secondo caso lo *SC* è migrato sul nodo 2 a causa di scarsità di risorse sul nodo 1.

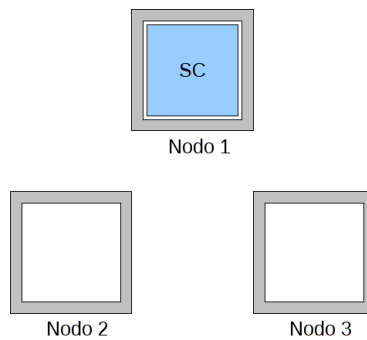


Figura 2.3: Esempio di scenario

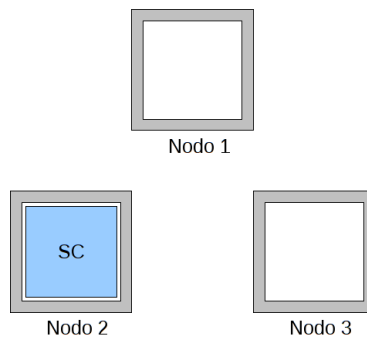


Figura 2.4: Esempio di scenario

### 2.2.2 Distributed Shared Memory

Per consentire la comunicazione tra i nodi del sistema è stato aggiunto un ulteriore livello nell'architettura in questione, ovvero il *dsm*. Tale componente è stato progettato sfruttando i meccanismi di comunicazione di Jade,

in particolare lo scambio di messaggi tra server e client dsm avviene tramite messaggi di Jade.

### **DsmServer**

E' stato realizzato un agente che si occupa principalmente della ricezione e la computazione delle richieste. Nello specifico quando il server riceve una specifica richiesta effettua la rispettiva operazione sul database dsm.

### **DsmData**

Questo componente è necessario per la gestione del dsm e delle tuple, in quanto consente operazioni di **IN** , **OUT** , **UPDATE** e **READ** .

### **DsmClient**

Quest'ultimo componente è fondamentale per ogni agente che deve inviare o ricevere informazioni dal tuple space, infatti consente l'invio delle richieste direttamente all'agente server che provvederà ad esaurirle.

## **2.2.3 Context Manager**

Il Context Manager, come già anticipato, si occupa di raccogliere tutte le informazioni sulle risorse del proprio su un contenitore chiamato **Context** , nello specifico il contenuto è il seguente:

**cpu:** valore del processore;

**ram:** valore della ram presente nel nodo;

**memory:** valore percentuale dell'occupazione di memoria del nodo;

**energy:** valore di energia residua;

**latency:** latenza di trasmissione attuale del nodo publisher;

**reliability:** valore dell'affidabilità del nodo;

**reqInterval:** frequenza di richieste;

**location:** contiene informazioni su cui risiede CM;

**network:** tipo di rete del nodo;

**bandwidth:** banda di rete del nodo.

Per raccogliere tali informazioni il CM effettua delle operazioni di **IN** sul dsm ad intervalli predefiniti. Raccolti i dati immette il contesto sul dsm tramite operazioni di **OUT**. Il CM si occupa inoltre di richiedere ed accettare contratti SLA in base al tipo di nodo in cui si trova, se il nodo è di tipo *publisher* accetta contratti invece se è di tipo *subscriber* richiede contratti. Una volta accettato un contratto viene rinviato sul dsm tramite **OUT** per essere passato allo SC.

### 2.2.4 SLA Checker

È il componente su cui si basa tutto il funzionamento del sistema realizzato, in quanto è l'agente che si occupa di controllare la validità dei contratti e migrare quando necessario. Per effettuare tali operazioni è necessario essere a conoscenza del contesto dei vari nodi e dei contratti SLA stipulati. Lo SC richiede continuamente i contesti dei vari nodi dal dsm, inoltre memorizza tutti i contratti che sono stati stipulati per essere a conoscenza delle relazioni tra i nodi. La conoscenza di tutte le informazioni descritte è necessaria all'algoritmo di *checking*, in quanto ha bisogno di molteplici informazioni per scoprire il miglior nodo su cui migrare (ved. 4).

<

# Capitolo 3

## Dsm

In questo capitolo verrà riportato dettagliatamente il funzionamento e la strutturazione del componente dsm necessario alla comunicazione tra agenti, nello specifico verrà spiegato come è stato realizzato il tuple space e tutti i meccanismi ad esso correlati. Per il progetto in questione sono stati realizzati i comandi prettamente necessari, che sono: IN, OUT, UPDATE e READ [1].

### 3.1 DsmData

Per la gestione dei dati è stato realizzato una sorta di database volatile tramite lo sfruttamento di *Hashtable* e *Queue*. In particolare si realizza una struttura del genere:

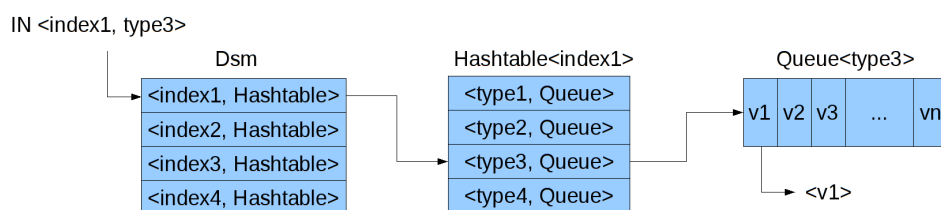


Figura 3.1: Esempio di IN nel Dsm

Dalla figura è riportato un esempio delle operazioni eseguite sul database nel momento in cui si effettua una richiesta **IN** `<index1, type3>`, ovvero si

richiede un valore di tipo 3 che si trova nell'indice 1. Le operazioni eseguite sono elencate di seguito:

1. Quando perviene la richiesta al db si cerca immediatamente la presenza di una chiave `index1` nell'hashtable di primo livello;
2. se la chiave è presente si prende l'hashtable associata e si procede con il controllo della presenza della chiave `type3` all'interno dell'hashtable di secondo livello;
3. se la chiave è presente si prende la coda che gli è associata e si estrae il valore che si trova in testa.

Operazioni analoghe vengono effettuate per gli altri tipi di operazioni; in particolare le operazioni implementate hanno il seguente comportamento:

**OUT:** equivale alla tupla OUT `<index1, type3, v1>` e consiste nell'inserire un valore sul tuple space.

**IN:** è rappresentata dalla tupla IN `<index1, type3>` ed il suo scopo è reperire il valore in testa alla coda e quindi eliminarlo dal tuple space.

**READ:** tupla uguale a IN, ma ha l'unica differenza di non eliminare il valore dopo averlo reperito.

**UPDATE:** tupla come OUT con la differenza che se il valore esiste lo sostituisce.

## 3.2 DsmClient

Questo componente consente la comunicazione con il server dsm sfruttando lo scambio di messaggi che mette a disposizione Jade con il suo *framework*. Il modulo in questione consente due meccanismi principali che sono: l'invio di messaggi bloccanti e l'invio di messaggi non bloccanti. Per quanto riguarda i messaggi bloccanti (IN e READ) si ha:

1. Il Client invia la richiesta al server ed attende la risposta;
2. Il Server riceve la richiesta, la esaurisce ed invia la risposta;
3. Il Client riceve la richiesta e la ritorna all'agente chiamante.

I messaggi non bloccanti di tipo OUT e UPDATE, invece, vengono inviati nel seguente modo:

1. Il Client invia la richiesta al server e ritorna all'agente chiamante;
2. Il Server riceve la richiesta e la esaurisce;

### 3.3 DsmServer

Tale componente può essere inteso come un *proxy* in quanto non fa altro che tradurre le richieste del client a operazioni sul **DsmData** e quindi ritornare una risposta (in caso di messaggi bloccanti). Una caratteristica importante di questo agente è che può migrare tra i vari nodi in base alla disponibilità delle risorse dei nodi, nello specifico non fa altro che seguire lo *SC*, ovvero quando lo SC migra su un altro nodo il DsmServer lo segue (rif SC).



## Capitolo 4

# Modello Matematico

Il modello matematico riguarda tutta la parte generativa dei valori delle risorse e dell'indice di occupazione dei nodi. Molte delle relazioni che sono state utilizzate sono state ricavate empiricamente grazie ad un'operazione di *tuning* che ha consentito di raggiungere una situazione il più reale possibile.

### 4.1 Valori risorse

Le risorse del sistema realizzato possono essere suddivise in due gruppi: dipendenti dalla presenza dello SC (e Dsm) e indipendenti dalla loro presenza. Quanto detto è stato fatto per rappresentare più realmente un caso normale. Si consideri il seguente esempio, ovvero un nodo mobile con connessione *wireless* e un nodo fisso con connessione *wired*. Il nodo mobile viene collegato e scollegato dalla rete elettrica quando necessario. Sul nodo mobile sono in esecuzione delle normali operazioni di richiesta di informazioni verso il nodo fisso. Sul nodo fisso è in esecuzione il server che risponde alle risposte di vari nodi mobili. Lo SC è al momento sul nodo fisso, questo implica un carico computazionale molto elevato per tale nodo, quindi se necessario tale agente SC migrerà sul nodo mobile che al momento ha molte risorse disponibili. Per consentire questo meccanismo di migrazione è stato necessario realizzare delle formule di generazione adattabili al loro stato attuale (presenza o non

presenza dello SC oppure collegato o scollegato dalla rete elettrica). Tutti i valori generati rappresentano delle percentuali. Di seguito vengono riportate tutte le formule usate per la generazione.

#### 4.1.1 Cpu

La risorsa *cpu* dipende fortemente dalla presenza dello SC, infatti si è deciso di applicare la seguenti relazioni, la prima nel caso “senza SC”

$$cpu = (\alpha \cdot 100) \bmod 60 \quad (4.1)$$

in cui  $\alpha$  rappresenta un valore casuale da 0 a 1. Il modulo 60 è stato usato per limitare il valore a 60. Nel caso “con SC” invece è stata usata la seguente relazione:

$$cpu = ((\alpha \cdot 100) \bmod 60) + 40 \quad (4.2)$$

in cui si aggiunge il valore 40 che rappresenta il carico supplementare dovuto alla presenza dello SC.

#### 4.1.2 Ram

La risorsa viene generata come per la *cpu*.

$$ram = (\alpha \cdot 100) \bmod 60 \quad (4.3)$$

Nel caso “con SC” invece è stata usata la seguente relazione:

$$ram = ((\alpha \cdot 100) \bmod 60) + 40 \quad (4.4)$$

#### 4.1.3 Memory

La memoria ha un comportamento leggermente più complesso, infatti è stato realizzato un meccanismo di riempimento e svuotamento della memoria dipendente dalla presenza dello SC. All’avvio del nodo viene generato un valore iniziale tramite la seguente formula:

$$memory = (\alpha \cdot 100) \bmod 60 \quad (4.5)$$

Ad ogni *tick* si modifica il valore nel seguente modo:

$$memory = memory + 0.2 \cdot ((\alpha \cdot 100) \bmod 10) \quad (4.6)$$

Quest'ultima relazione consiste nell'aggiungere al valore precedente il 20% di un valore modulo 10, grazie a ciò si ottiene un incremento molto lieve dell'utilizzazione della memoria. Nel caso "senza SC" invece si ha

$$memory = memory - 0.2 \cdot ((\alpha \cdot 100) \bmod 10) \quad (4.7)$$

a differenza del caso precedente si sottrae il nuovo valore da quello vecchio, questo sta ad indicare una diminuzione dell'utilizzazione. Inoltre nel caso in cui la memoria raggiunge valori minori di 0 o maggiori 100 si rigenera il valore della memoria con la formula usata inizialmente (ved. 4.5).

#### 4.1.4 Energy

Nel caso del componente riguardante l'energia sono state applicate delle formule più adattative, in quanto il suo valore dipende dal collegamento o scollegamento alla rete elettrica e dalla presenza o non presenza dello SC. Inoltre dipende dal tipo di connessione, ovvero se è wireless o wired. All'avvio viene generato il valore tramite la seguente formula.

$$memory = (\alpha \cdot 100) \bmod 60 \quad (4.8)$$

Ad ogni tick dell'agente si può avere una delle seguenti formule in base alle condizioni. In caso di connessione wired si ha:

$$energy = 100 \quad (4.9)$$

considerando che il wired è stato assunto come nodo fisso collegato alla rete elettrica. In caso di connessione wireless si deve distinguere dal caso in cui è connesso alla rete elettrica (`powerOn`) e il caso in cui non lo è (`!powerOn`). Per rendere reale i valori generati si è realizzato un meccanismo per gestire la connessione e sconnessione dalla rete elettrica, ovvero:

$$\text{se } energy \leq 10 \text{ then } powerOn = true \quad (4.10)$$

$$\text{se } energy \geq 99 \text{ then } powerOn = false \quad (4.11)$$

Con la rete elettrica collegata si ha:

$$energy = energy + 0.8 \text{ se SC non presente,} \quad (4.12)$$

$$energy = energy + 0.6 \text{ se SC presente} \quad (4.13)$$

in caso di rete elettrica scollegata si ha:

$$energy = energy - 0.2 \text{ se SC non presente,} \quad (4.14)$$

$$energy = energy - 1 \text{ se SC presente} \quad (4.15)$$

#### 4.1.5 Latency

Nel caso della latenza è stato utilizzato un meccanismo dipendente solo dalla presenza dello SC e dal tipo di connessione. In particolare se nel nodo è presente lo SC oppure ha una connessione wireless si ha:

$$latency = ((\alpha_1 \cdot 100) \bmod 60) + \left( \left( \frac{\alpha_2 \cdot 100}{\frac{\beta}{50}} \right) \bmod 40 \right) \quad (4.16)$$

come si può notare vengono generati due numeri, uno modulo 60 ed uno modulo 40 in modo da avere un numero massimo di 100. Inoltre è presente un valore  $\beta$  che rappresenta la banda del nodo.

#### 4.1.6 Reliability

Il valore della affidabilità del canale viene generato nello stesso modo della latenza, infatti si ha:

$$reliability = ((\alpha_1 \cdot 100) \bmod 60) + \left( \left( \frac{\alpha_2 \cdot 100}{\frac{\beta}{50}} \right) \bmod 40 \right) \quad (4.17)$$

#### 4.1.7 ReqInterval

In quest'ultima risorsa la generazione avviene molto semplicemente generando un numero casuale da 0 a 100.

$$reqInterval = \alpha \cdot 100 \quad (4.18)$$

## 4.2 SLA Checker

Lo SC, essendo il componente centrale di tutto il sistema, ha richiesto un lavoro molto accurato per la realizzazione di un buon algoritmo e di un indice di valutazione efficiente. Per algoritmo si intendono le operazioni che vengono svolte dallo SC ciclicamente (ad ogni tick dell'agente), ovvero il controllo sulla validità dei contratti e il controllo della necessità di migrazione. Per indice di valutazione si intende il valore che viene generato per ogni nodo dai valori del suo contesto, tale indice viene utilizzato dallo SC per trovare il nodo su cui eventualmente migrare.

### 4.2.1 Algoritmo controllo

L'algoritmo realizzato consente il controllo della violazione dei contratti e il controllo della necessità di migrazione. In particolare tali operazioni possono essere effettuate grazie alla conoscenza di tutte le informazioni sui nodi del sistema, ovvero il contesto e i contratti. L'algoritmo è suddiviso in due parti, la prima si occupa del controllo della validità dei contratti.

#### SLA Contract checking

1. si prende un contratto SLA dalla lista dei contratti;
2. si recuperano i dati del contesto dei due partecipanti al contratto (Publisher, Subscriber);
3. si controlla la validità del contratto tramite la relazione seguente:

$$latency_{pub} > latency_{sla}$$

OR

$$reliability_{pub} > reliability_{sla}$$

OR

$$reqInt_{sub} > reqInt_{sla}$$

4. se tale relazione risulta valida allora il contratto è stato violato.

Dalla relazione riportata si può notare la presenza dei valori di `latency` , `reliability` e `reqInt` che hanno come pedice `pub` , `sub` e `sla` che indicano rispettivamente il Publisher, il Subscriber e il contratto SLA. La relazione riportata non fa altro che controllare se il publisher viola il contratto non garantendo la latenza e l'affidabilità precedente, oppure se il Subscriber viola il contratto effettuando più richieste di quante stabilite dal contratto SLA.

### SC Migration checker

La seconda parte di cui si occupa lo SC è la verifica della necessità di migrazione su altri nodi. In particolare tale operazione consiste nel valutare l'indice del contesto di ogni nodo, controllando se il valore supera una certa soglia prefissata, nel qual caso sarebbe necessario migrare su un nuovo nodo. Il calcolo dell'indice di utilizzazione viene effettuato con la seguente formula:

$$index = cpu \cdot 0.23 + ram \cdot 0.23 + memory \cdot 0.23 + (100 - energy) \cdot 0.3 \quad (4.19)$$

Il valore *index* calcolato rappresenta una percentuale che va da 0 a 100 che indica l'utilizzazione del nodo valutato. L'indice è il fulcro dell'algoritmo di *Migration checking*, in quanto la migrazione si basa proprio sul confronto di tale valore. Verranno ora riportati i passi dell'algoritmo:

1. Innanzitutto si recupera il nodo migliore su cui migrare tramite la funzione `getBestNode()` ;
2. si seleziona il contesto attuale del nodo in cui si trova lo SC;
3. si verifica che l'indice del contesto attuale sia maggiore dell'indice di migrazione;
4. se la condizione è verificata si effettua la migrazione sul nodo risultato migliore;

5. si notifica, tramite dsm, l'avvenimento della migrazione a tutti nodi presenti;
6. se la condizione non si verifica non viene effettuata la migrazione.

### Best Node

La funzione `getBestNode()` riportata nell'algoritmo precedente viene descritta dettagliatamente di seguito:

```
AID next, iter;
Enumeration<AID> keys = sc.getContextTable().keys();
Context context;
float cpu, ram, memory, energy;
float total=500, iter=0;

while(keys.hasMoreElements()) {
    next = keys.nextElement();
    context = sc.getContextTable().get(next);
    if(context!=null) {
        cpu = context.getCpu();
        ram = context.getRam();
        memory = context.getMemory();
        energy = context.getEnergy();
        iter = cpu + ram + memory + (1-energy);
        if(cpu < Context.CPU_LIMIT && ram < Context.RAM_LIMIT && memory <
            Context.MEMORY_LIMIT && energy > Context.ENERGY_LIMIT && iter <
                total) {
            best = next;
            total = iter;
        }
    }
}
```

Lo scopo dell'algoritmo riportato è di recuperare il contesto che ha i valori “migliori”, ovvero che si attiene alla seguente relazione:

$$cpu < cpu_{limit} \text{ AND } ram < ram_{limit} \text{ AND}$$

$$memory < memory_{limit} \text{ AND } energy > energy_{limit} \text{ AND } \theta < min$$

in cui i valori con pedice *limit* indicano il massimo numero per cui si accetta il valore come “migliore”.  $\theta$  invece rappresenta il minimo attuale, calcolato

tramite 4.20

$$\theta = cpu + ram + memory + (1 - energy) \quad (4.20)$$

in ogni ciclo viene calcolato tale valore e se risulta essere minore del minimo attuale si sostituisce *min* con tale valore.



# Capitolo 5

# Implementazione

Questo breve capitolo presenta una panoramica dell'applicazione realizzata, in particolare viene riportato lo schema *UML* delle dipendenze tra i package per avere un'idea di come è strutturato il sistema.

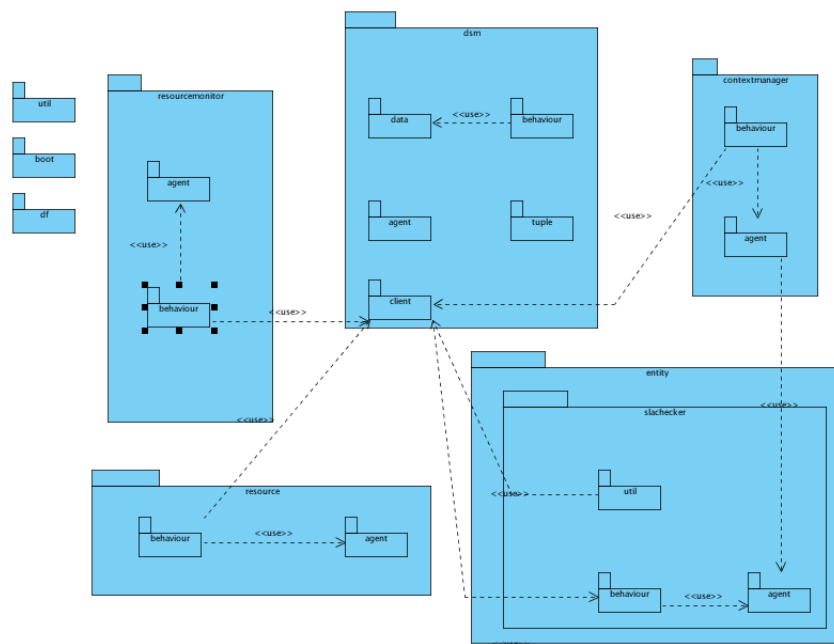


Figura 5.1: Relazioni tra Package

# Capitolo 6

## Simulazione

Per valutare le prestazioni del sistema realizzato si sono effettuate numerose simulazioni. Tutti i test condotti sono stati basati su una specifica configurazione di nodi, in particolare 1 nodo Publisher e 2 Subscriber. I due nodi consumer richiedono la stipulazione di un contratto al producer, il quale espleta tutte le richieste. Si riporta ora un esempio di esecuzione con relativa descrizione ed analisi dei dati.

### 6.1 Esecuzione

In principio si avvia il nodo Producer il quale andrà in attesa di richieste dei consumatori. In seguito si avviano i due Consumer (o Subscriber) i quali richiedono ed ottengono un contratto con il Subscriber. Stabiliti i contratti inizia la vera fase di simulazione in cui ogni nodo genera valori casuali per le proprie risorse e lo SC raccoglie tali dati per valutare gli SLA e la necessità di migrazione. Per eseguire il sistema bisogna utilizzare i seguenti comandi;

```
ant runGui
```

per avviare il MainContainer

```
ant -f build2.xml AgentLauncher
```

che avvia i tre nodi in tre container differenti.

### 6.1.1 Scenario

I tre nodi dello scenario d'esecuzione hanno le seguenti caratteristiche:

1. Publisher con connessione WIRED e 512Kb di banda;
2. subscriber con connessione WIRELESS e 256Kb di banda;
3. subscriber con connessione WIRELESS e 256Kb di banda.

### 6.1.2 Generazione contratti SLA

Il contratto SLA, ovvero i livelli di latency, reliability e reqInterval vengono generati casualmente dal publisher nel momento in cui raccolgono una richiesta di contratto. Tale operazione può essere vista nel log di esecuzione del programma simulativo.

1. Il Subscriber richiede un contratto con un qualsiasi publisher disponibile;
2. Il publisher legge la richiesta dal dsm;

```
SLAContract-request received from cm2
```

3. il publisher genera i parametri del contratto casualmente e lo immette sul dsm;
4. lo SC rileva la presenza del contratto e lo aggiunge alla sua lista.

```
Added Contract
```

### 6.1.3 Migrazione

Inizialmente lo SC risiede sul primo nodo, ovvero il publisher (ved. 6.1).

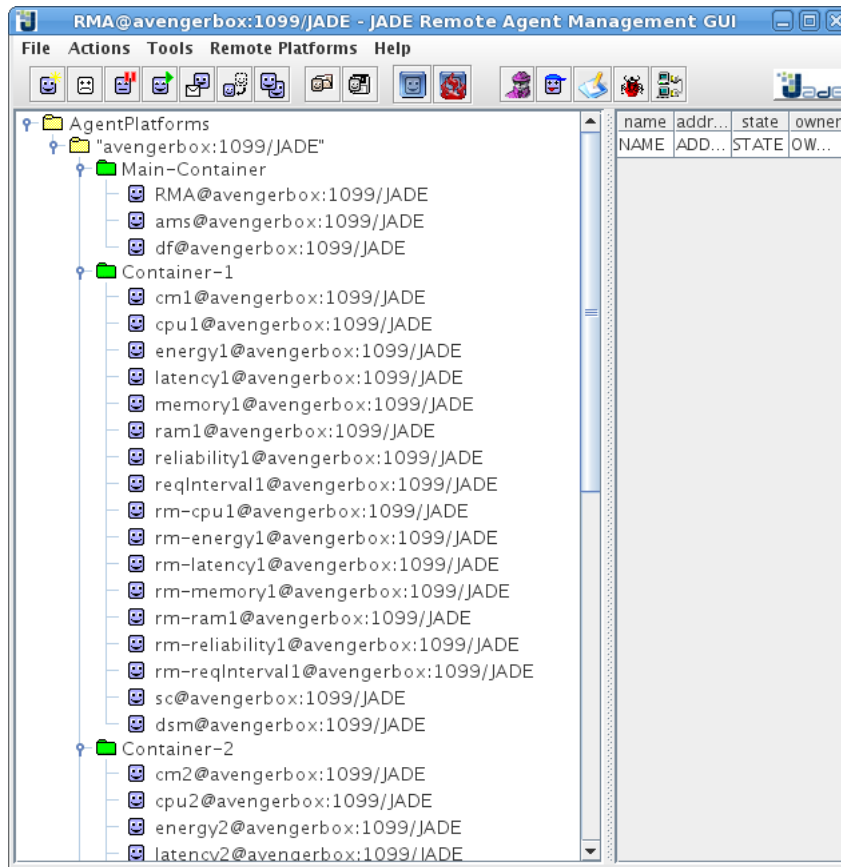


Figura 6.1: Migrazione non avvenuta

In questa figura si può notare la presenza del dsm e dello SC sul primo nodo, in quanto la simulazione è stata appena avviata. Dal log si può rilevare che lo SC ha già iniziato a controllare la necessità di migrazione e che al momento si trova sul Container-1 ovvero il nodo 1. Il valore index del nodo attuale non ha ancora superato il valore di soglia (52), di conseguenza l'SC rimane sul nodo attuale.

```
Sono su Container-1
Context[cm3]--> cpu: 40.092556, ram: 16.937687, memory: 34.849556, energy:
81.61731, index: 26.647161
Context[cm1]--> cpu: 57.77024, ram: 51.60415, memory: 45.732647, energy:
100.0, index: 35.674618
Context[cm2]--> cpu: 4.5756726, ram: 52.93144, memory: 40.222347, energy:
98.34838, index: 22.97326
```

```
Associated with: cm1
Actual context—> cpu: 57.77024, ram: 51.60415, memory: 45.732647, energy:
100.0, index: 35.674618
```

Nel momento in cui il carico su tale nodo raggiunge valori troppo alti avverrà la migrazione. Infatti poco tempo dopo si ha una situazione del genere:

```
Sono su Container-1
Context[cm3]--> cpu: 10.354793, ram: 26.346369, memory: 2.7747908, energy:
75.6174, index: 16.394249
Context[cm1]--> cpu: 77.09479, ram: 64.88432, memory: 87.20903, energy:
100.0, index: 52.713272
Context[cm2]--> cpu: 5.725228, ram: 59.014675, memory: 7.089407, energy:
92.148476, index: 18.876198
Associated with: cm1
Actual context—> cpu: 77.09479, ram: 64.88432, memory: 87.20903, energy:
100.0, index: <52.713272>
Migration to cm3
Moving DSM to 3

Sono su Container-3
Context[cm1]--> cpu: 7.0192904, ram: 18.302485, memory: 86.83973, energy:
100.0, index: 25.797146
Context[cm2]--> cpu: 25.42843, ram: 30.698774, memory: 4.063858, energy:
91.34849, index: 16.4394
Context[cm3]--> cpu: 3.4131508, ram: 16.183126, memory: 7.618082, energy:
74.61742, index: 13.874079
Associated with: cm3
Actual context—> cpu: 3.4131508, ram: 16.183126, memory: 7.618082, energy:
74.61742, index: <13.874079>
```

l'indice sul nodo attuale ha raggiunto un valore di 57.71 a causa di un elevato uso delle risorse disponibili, in particolare della della memoria. Dalla figura 6.2 si può notare che il dsm e l'SC si sono spostati sul Container-3. Su quest'ultimo nodo l'indice è adesso di 21.1.

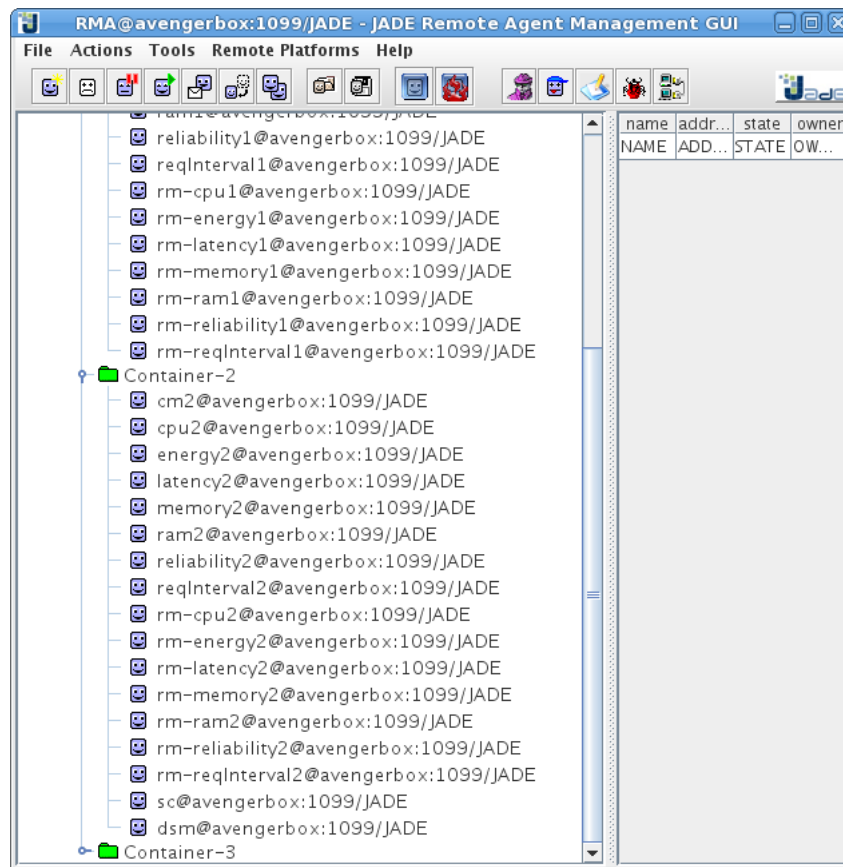


Figura 6.2: Migrazione su secondo nodo

Il nodo 3 è di tipo WIRELESS e quindi ha una energia limitata, il che implica un tempo di residenza per l'SC e il dsm molto più ridotto, in quanto l'indice raggiungerà rapidamente valori alti.

```
Sono su Container-3
Context[cm1]--> cpu: 0.81155604, ram: 33.999382, memory: 51.11227, energy:
    100.0, index: 19.762337
Context[cm2]--> cpu: 36.432972, ram: 58.424694, memory: 1.620206, energy:
    84.948586, index: 26.705338
Context[cm3]--> cpu: 67.67117, ram: 95.22946, memory: 26.253895, energy:
    49.21744, index: 58.74031
Associated with: cm3
Actual context--> cpu: 67.67117, ram: 95.22946, memory: 26.253895, energy:
    49.21744, index: <58.74031>
Migration to cm1
```

```
Moving DSM to 1

Sono su Container-1
Context[cm1]--> cpu: 50.261562, ram: 62.669903, memory: 48.456142, energy:
100.0, index: 37.11915
Context[cm2]--> cpu: 1.1113803, ram: 6.8676476, memory: 18.00424, energy:
83.9486, index: 10.791573
Context[cm3]--> cpu: 83.216156, ram: 71.41142, memory: 32.01974, energy:
44.21744, index: 59.663654
Associated with: cm1
Actual context--> cpu: 50.261562, ram: 62.669903, memory: 48.456142, energy:
100.0, index: <37.11915>
```

L'energia del nodo 3 è scesa da 49 a 44 il che ha costretto la migrazione sul nodo meno carico, ovvero l'uno (WIRED). In seguito vi è stata anche una violazione del contratto ed è stata segnalata ad entrambi i nodi partecipanti al contratto, i quali dovranno adattarsi per evitare di ripetere la violazione.

```
Violated with latency 23.106468 > 82.419365, reliability 41.978447 >
99.44324, reqInterval 79.21008 > 76.23332
```

Dal log si può notare che è stato il subscriber ad eccedere in richieste, infatti il reqInterval è inteso come frequenza.

## 6.2 Analisi dati

In questa sezione verranno esposti e commentati i grafici relativi alla simulazione, in quanto molto più esplicativi dei log riportati sopra. Tutti i grafici riportati avranno come ascisse i secondi che non rappresentano un valore reale ma simulato, infatti servono per dare un senso più reale alla simulazione.

### 6.2.1 Indice

L'indice è il valore più importante del test eseguito, in quanto la migrazione si basa su di esso. Come passo fondamentale vengono ora riportati i grafici relativi agli indici.

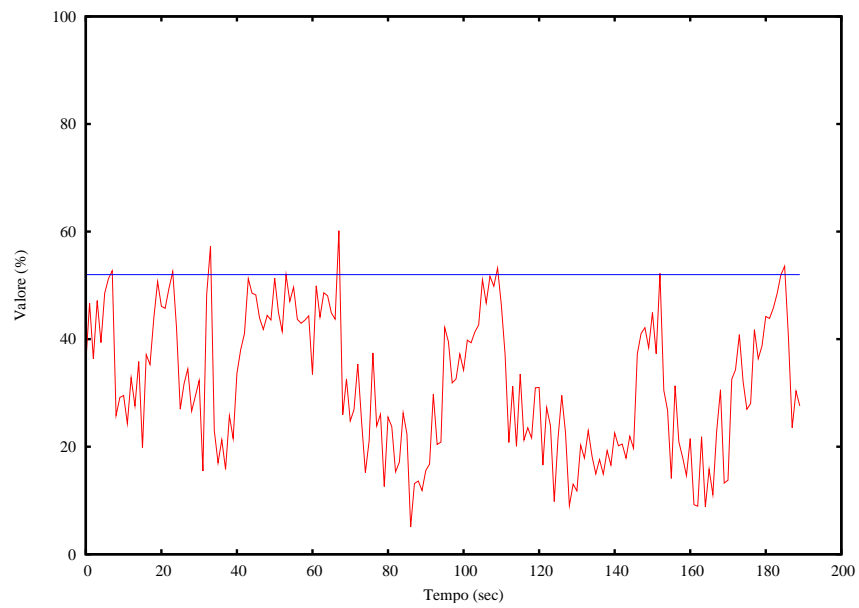


Figura 6.3: Indice primo nodo

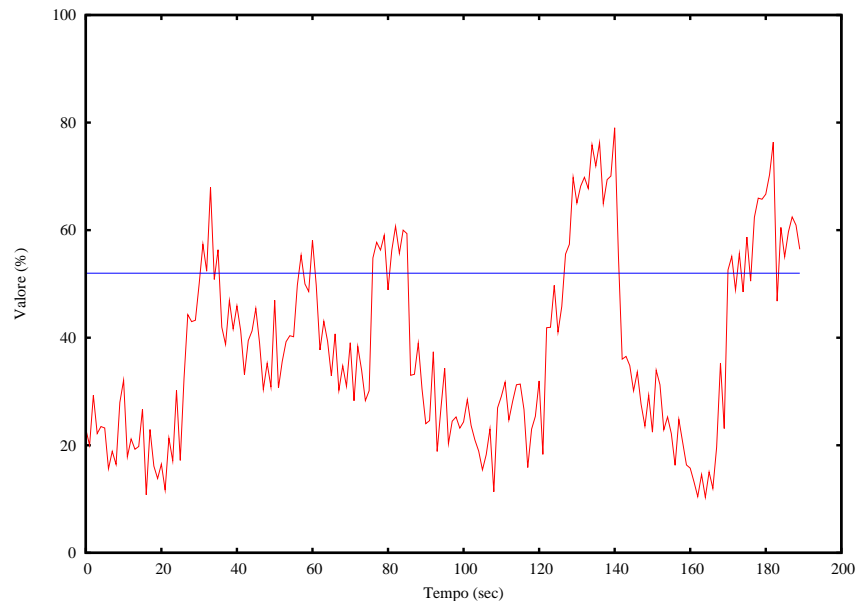


Figura 6.4: Indice secondo nodo



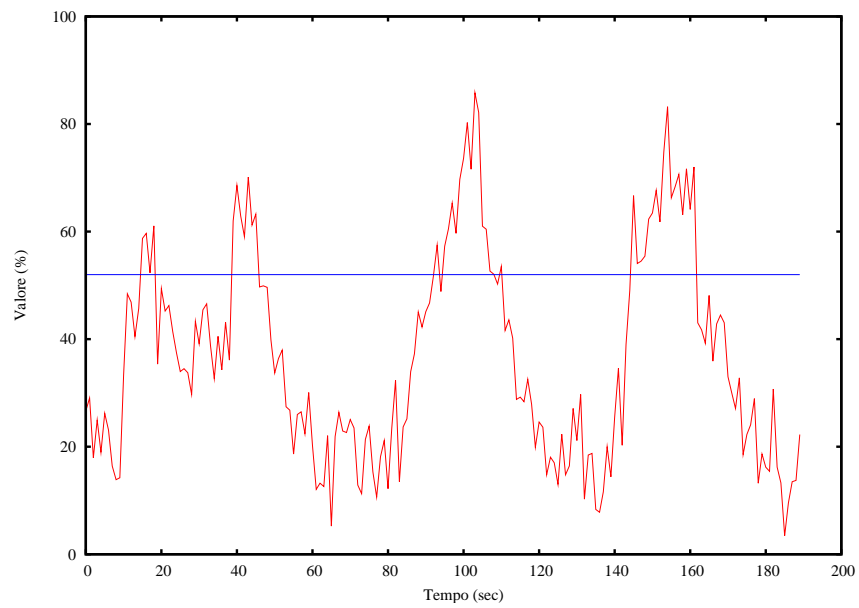


Figura 6.5: Indice terzo nodo

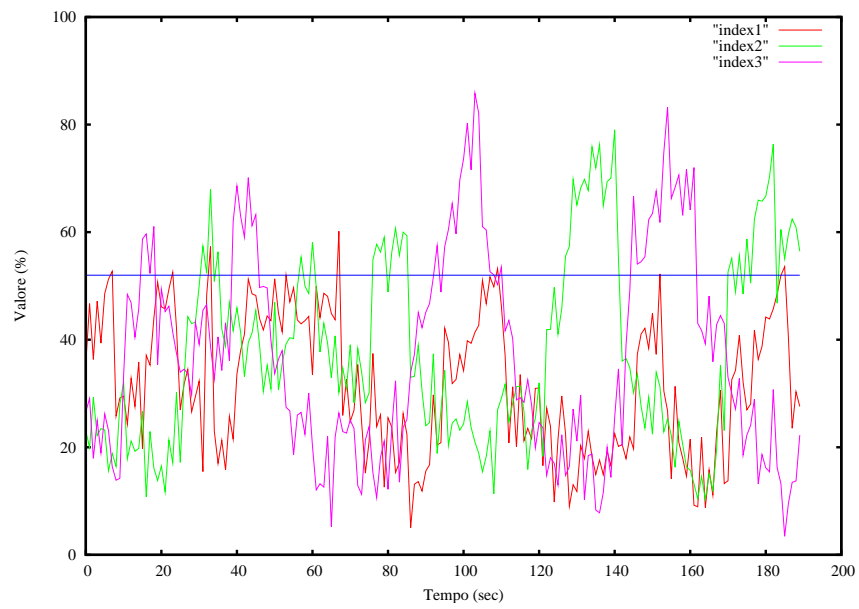


Figura 6.6: Indice tutti i nodi

Nelle figure 6.3, 6.4, 6.5 e 6.6 si può notare la presenza di una retta di colore blu, questa rappresenta il limite dell'indice, ovvero 52. Per comprendere meglio i grafici basta sapere che quando i punti superano la retta blu si ha una migrazione dal nodo corrispondente ad un altro. Ad esempio in figura 6.3 avviene la prima migrazione al secondo 8 circa.

### 6.2.2 Cpu

La cpu è molto sensibile alla migrazione, infatti quando lo SC si trova su un nodo si può notare un incremento del 40% sull'utilizzazione della cpu locale. Tale comportamento può essere facilmente estrapolato dai grafici riportati.

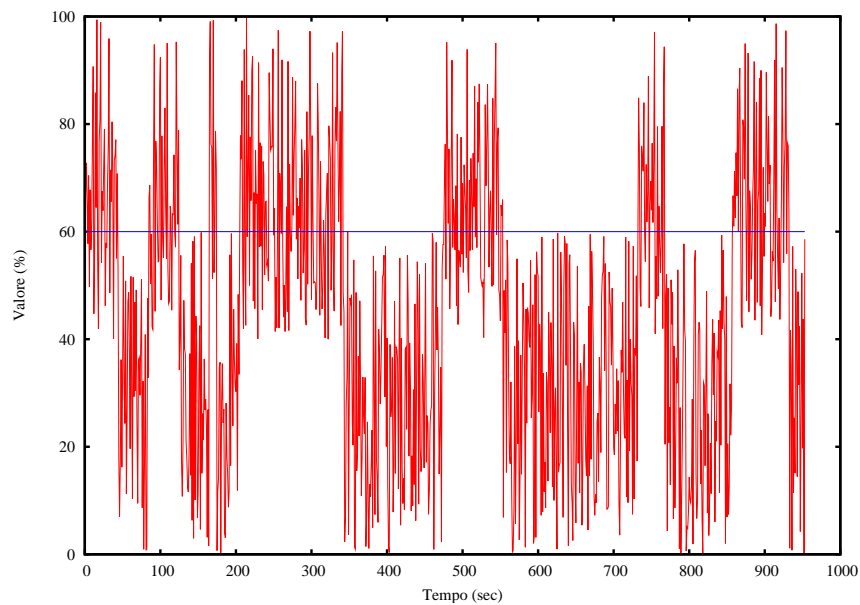


Figura 6.7: Cpu primo nodo

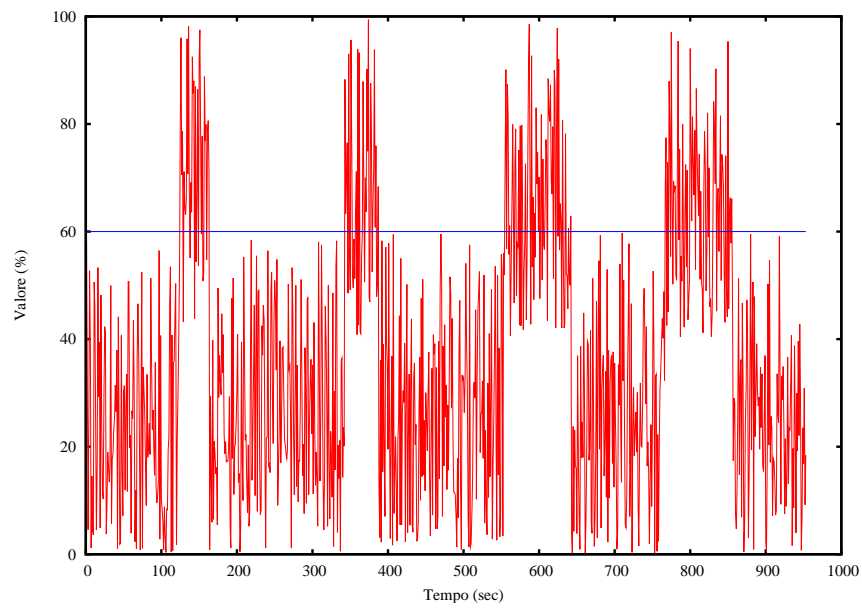


Figura 6.8: Cpu secondo nodo

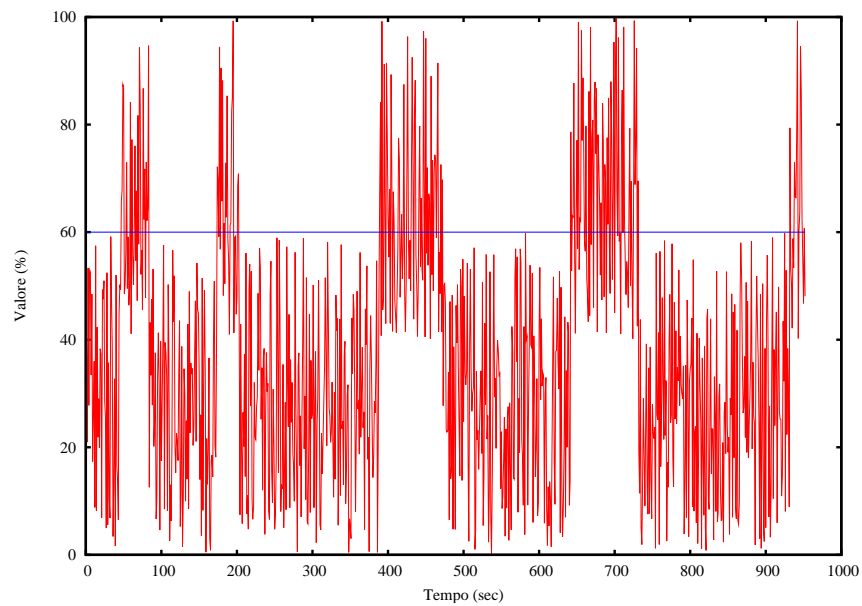


Figura 6.9: Cpu terzo nodo

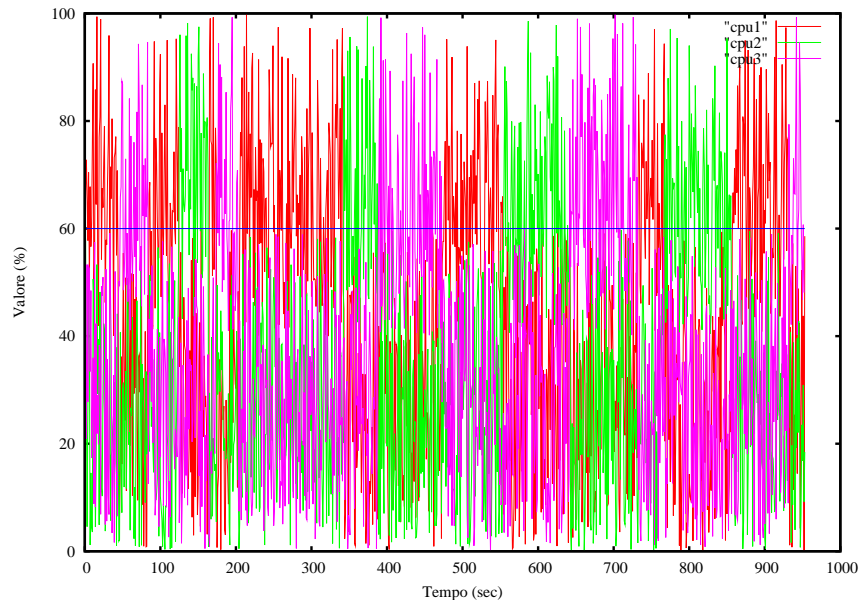


Figura 6.10: Cpu tutti i nodi

In tali grafici è presente una retta sul valore 60% per distinguere i nodi che hanno l'SC dagli altri, in particolare quando i valori si trovano oltre 60% è presente lo SC sul nodo corrispondente. In presenza di una migrazione avviene un brusco cambiamento di carico, infatti si scende sotto il 60%. Nella figura 6.10 sono graficati i tre nodi, questo crea confusione ma rende più semplicemente l'idea di come migri lo SC da un nodo ad un altro. Nello specifico si può notare che nella parte superiore al 60% si trova sempre un nodo nel singolo istante di tempo, questo è causato dalla presenza di un solo SC che si muove tra i nodi.

### 6.2.3 Ram

La ram ha una sensibilità come la cpu, infatti i valore vengono generati con le stesse formule. Dai grafici che seguono si possono notare i picchi di carico in corrispondenza delle migrazioni.

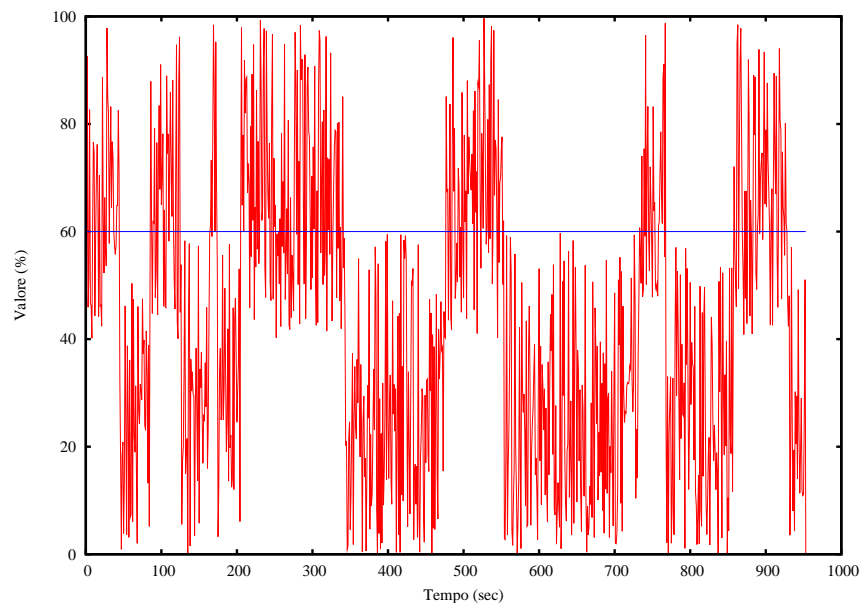


Figura 6.11: Ram primo nodo

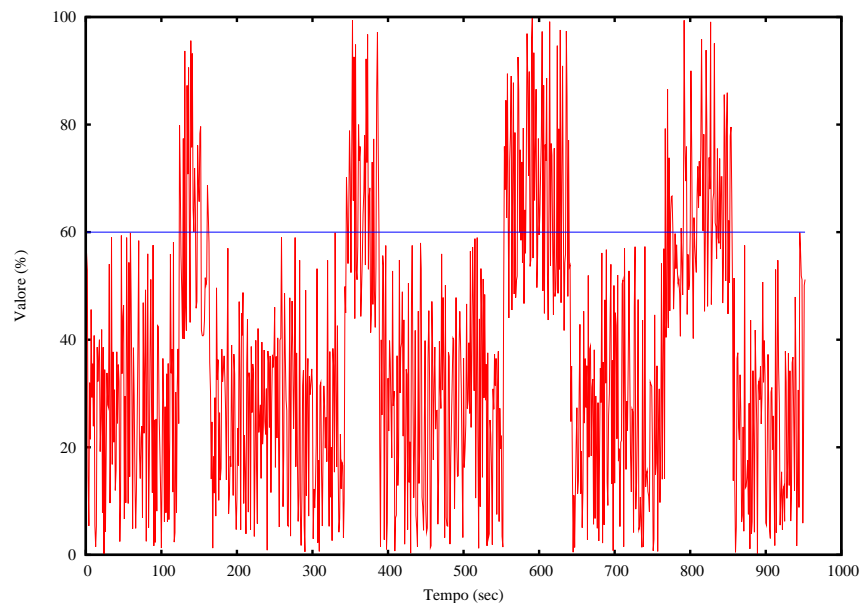


Figura 6.12: Ram secondo nodo

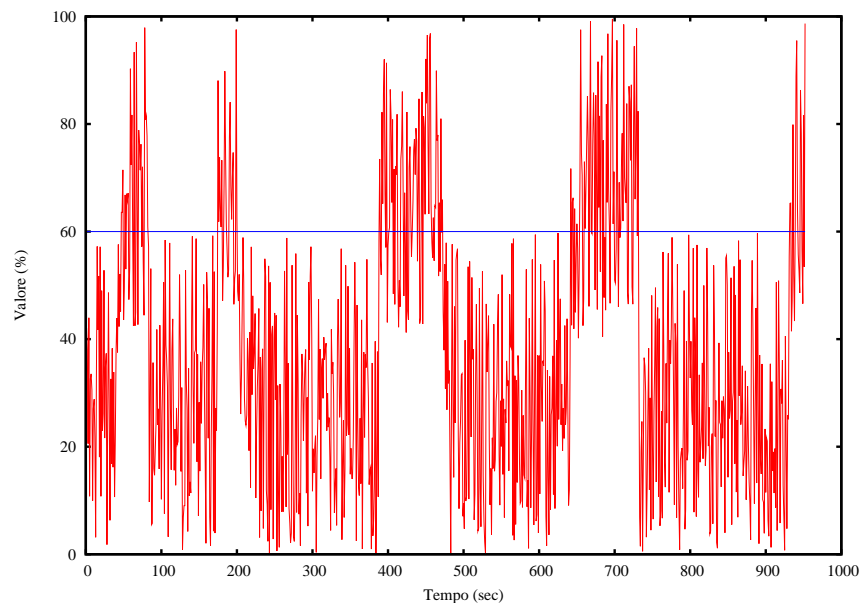


Figura 6.13: Ram terzo nodo

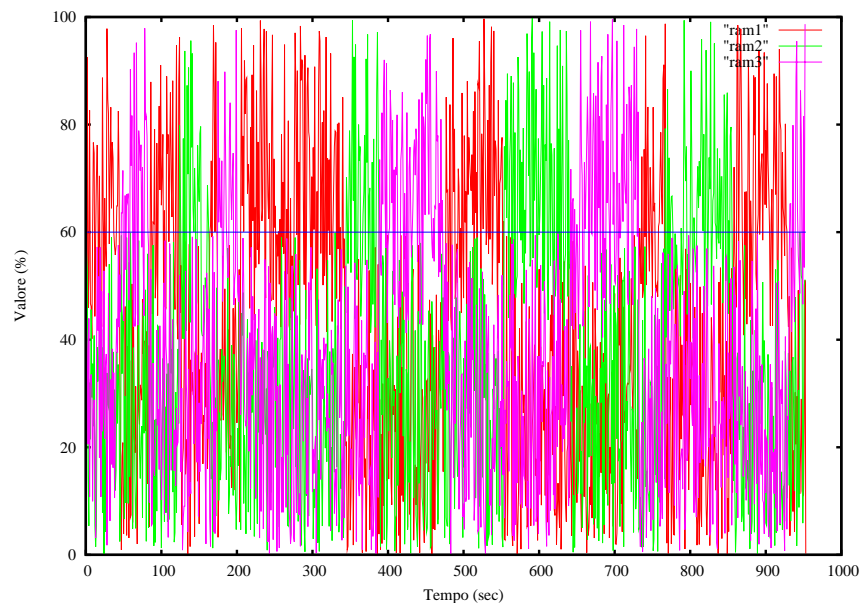


Figura 6.14: Ram tutti i nodi

### 6.2.4 Memory

La memoria è un parametro molto più dipendente dalle migrazioni, infatti in corrispondenza di esse si ha un incremento di utilizzazione costante ma non istantaneo. Dai grafici 6.15, 6.16, 6.17 e 6.18 si può notare che quando i valori di memoria raggiungono la soglia dello 0 o del 100 si ha un picco di traffico verticale, questo non è dovuto alle migrazioni ma potrebbe essere associato, ad esempio, a generazioni di file temporanei da parte del nodo. In corrispondenza di incrementi di memoria costanti vuol dire che è presente l'SC, invece quando si hanno cali costanti si ha il contrario, ovvero l'SC è migrato su un altro nodo.

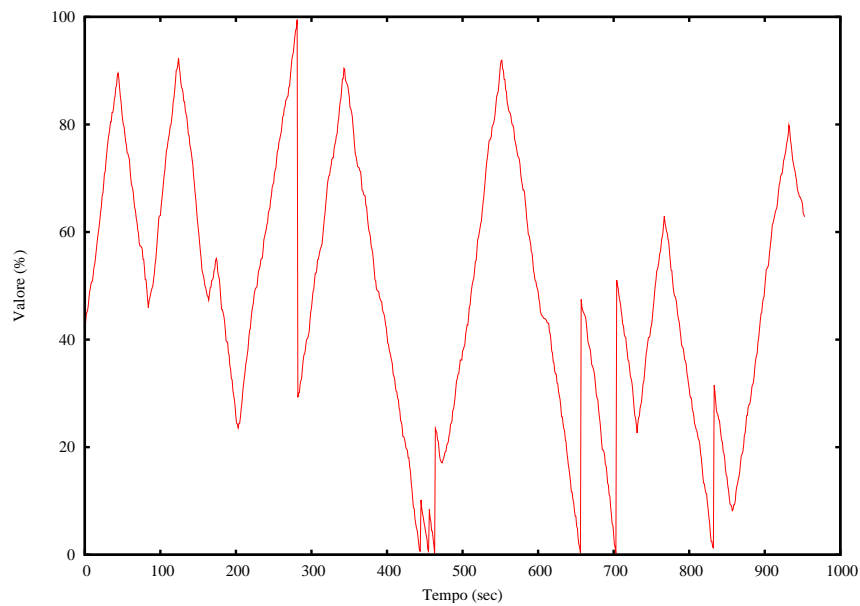


Figura 6.15: Memory primo nodo

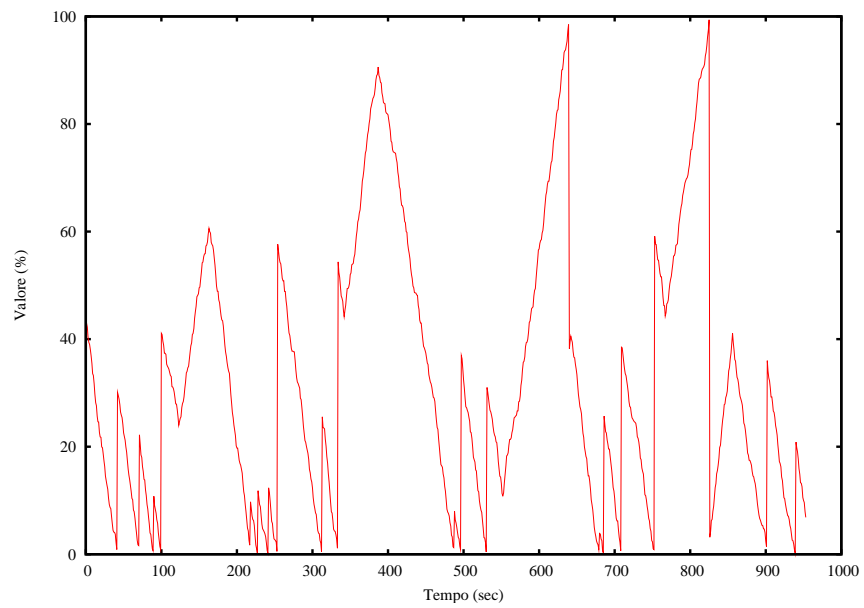


Figura 6.16: Memory secondo nodo

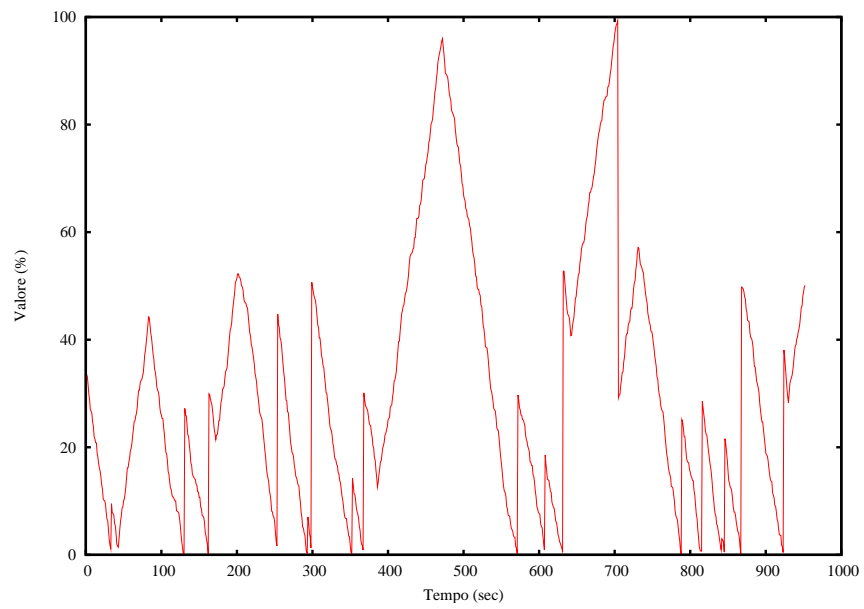


Figura 6.17: Memory terzo nodo



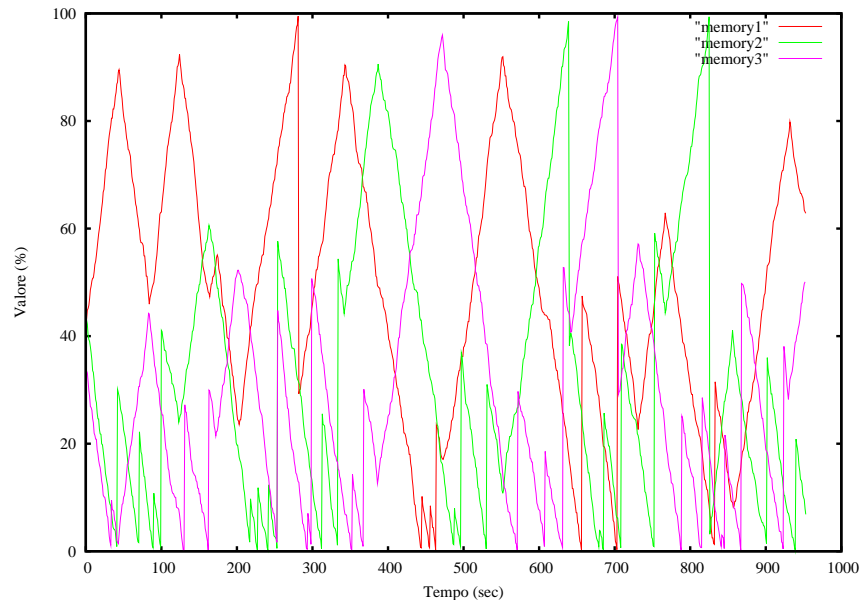


Figura 6.18: Memory tutti i nodi

### 6.2.5 Energy

L'energia è il parametro che varia più regolarmente in quanto dipende dal tipo di connessione e dalla presenza delle rete elettrica. Tale comportamento può essere notato dai grafici 6.19, 6.20, 6.21 e 6.22. In particolare in presenza dell'SC su un nodo si ha un utilizzo di corrente maggiore, nello specifico se il nodo è connesso alla rete elettrica si ha un caricamento più lento, essendo parte dell'energia sprecata a causa dell'SC; anche in caso di scollegamento dalla rete elettrica si ha uno spreco di energia maggiore causato dallo SC. Dalle figure si può anche notare che in corrispondenza del 10% si ha un incremento di carica, essendo il nodo ricollegato alla rete elettrica, invece al raggiungimento del 100% viene scollegato, considerando che il caricamento è completo. Un ulteriore particolare si può notare in figura 6.19, in quanto il nodo è di tipo fisso e quindi con corrente sempre disponibile, infatti il valore di energia è sempre 100%.

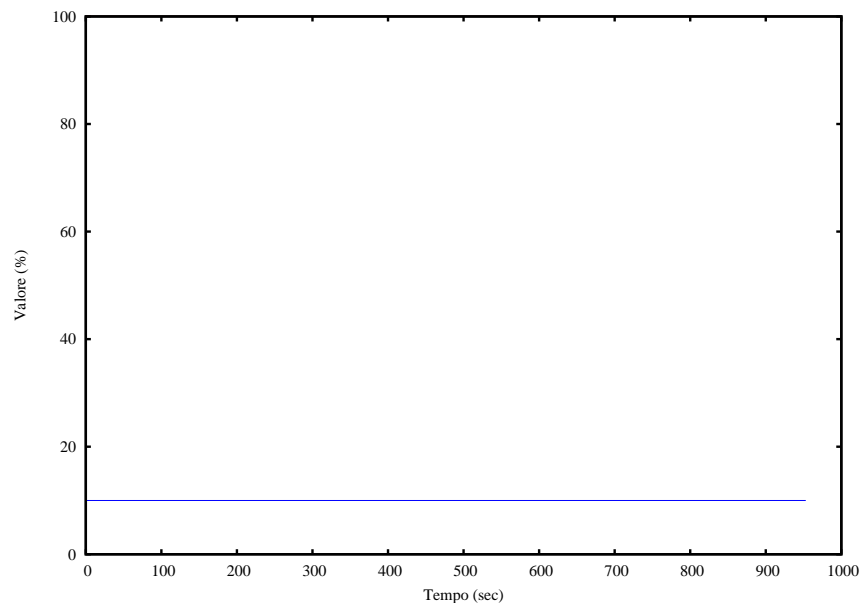


Figura 6.19: Energy primo nodo

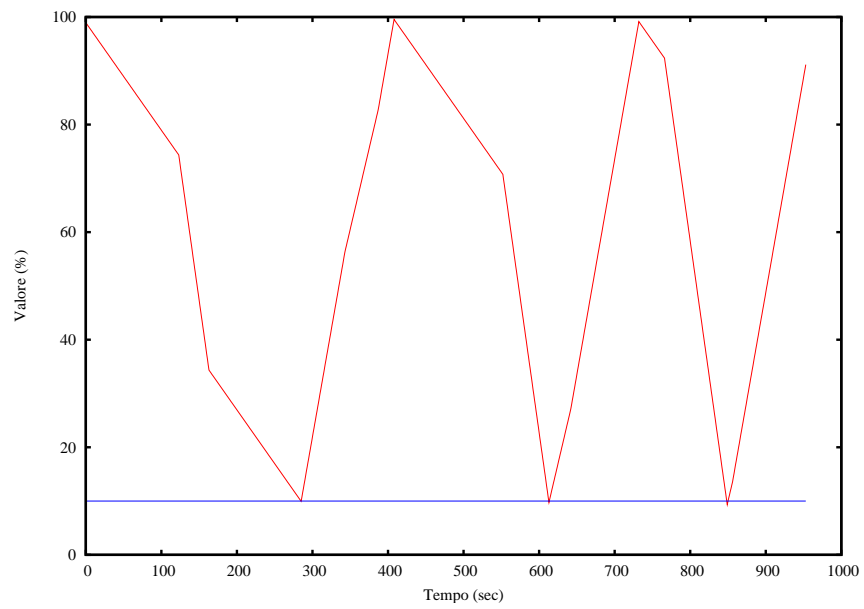


Figura 6.20: Energy secondo nodo

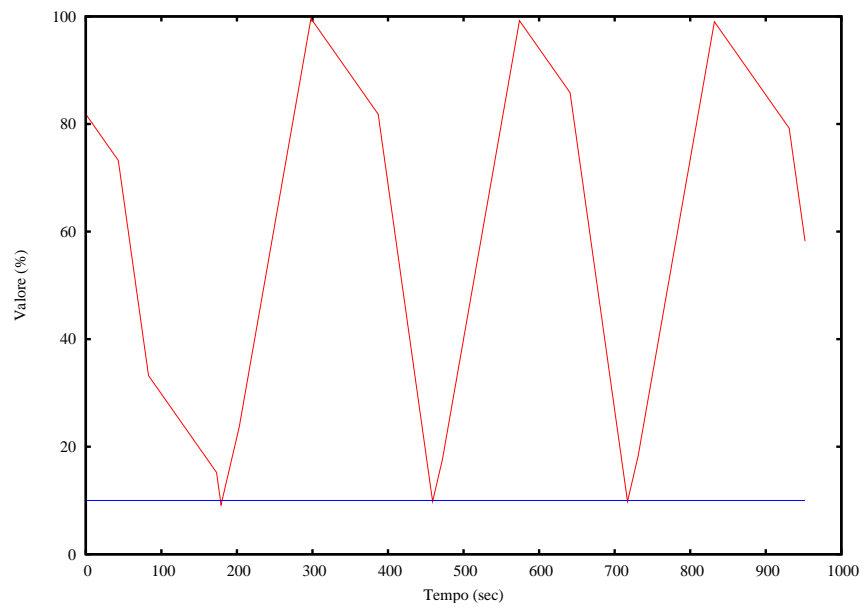


Figura 6.21: Energy terzo nodo

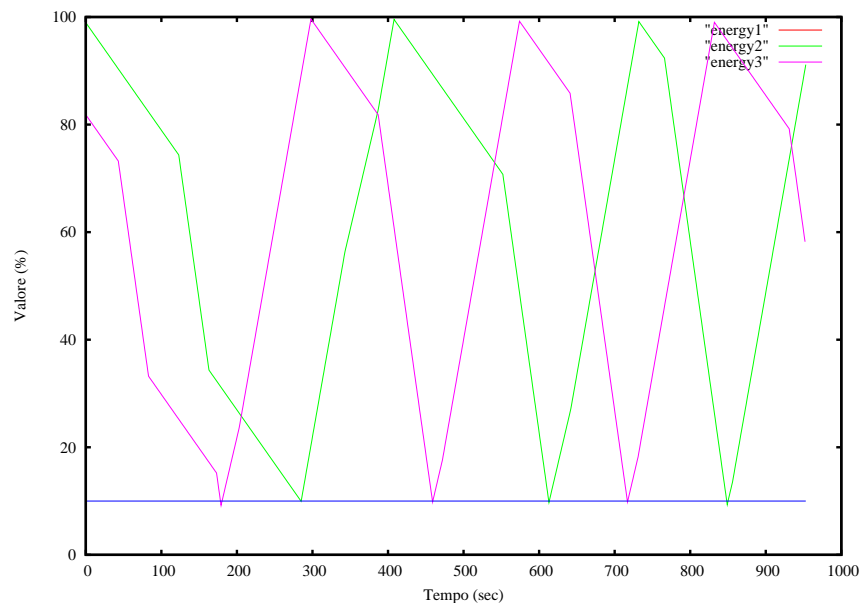


Figura 6.22: Energy tutti i nodi

### 6.2.6 Latency

Quest'ultimo valore generato dipende dalla presenza dell'SC, dal tipo di linea e dalla banda, come già riportato nel modello matematico. Nei seguenti grafici si possono vedere gli andamenti:

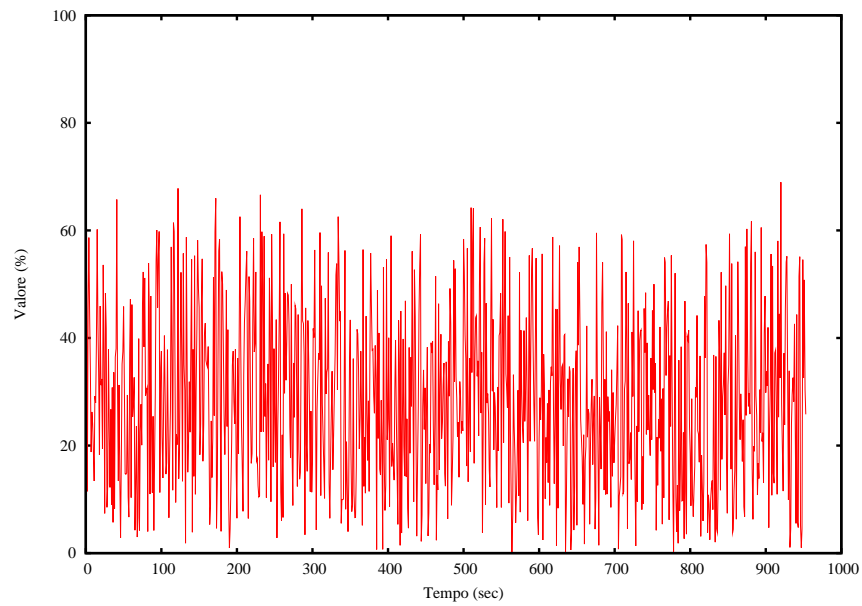


Figura 6.23: Latency primo nodo

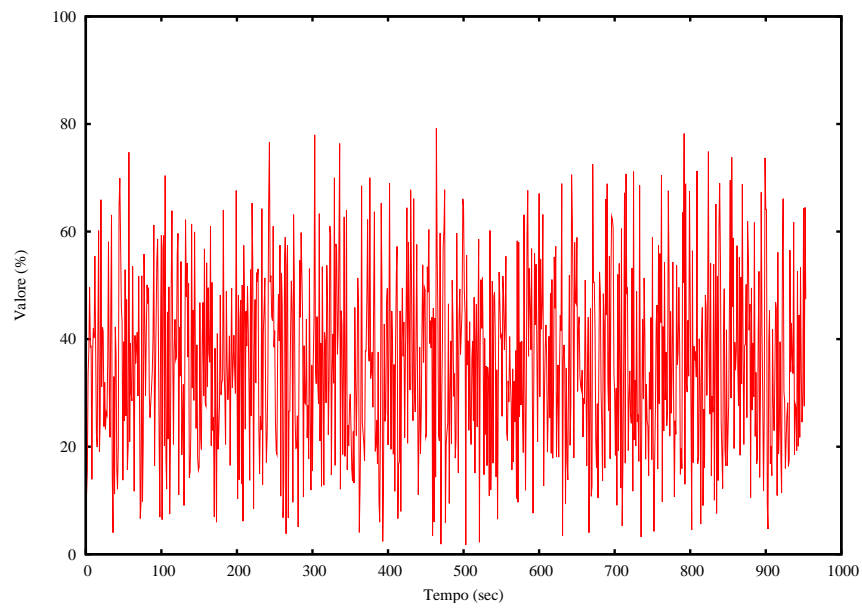


Figura 6.24: Latency secondo nodo

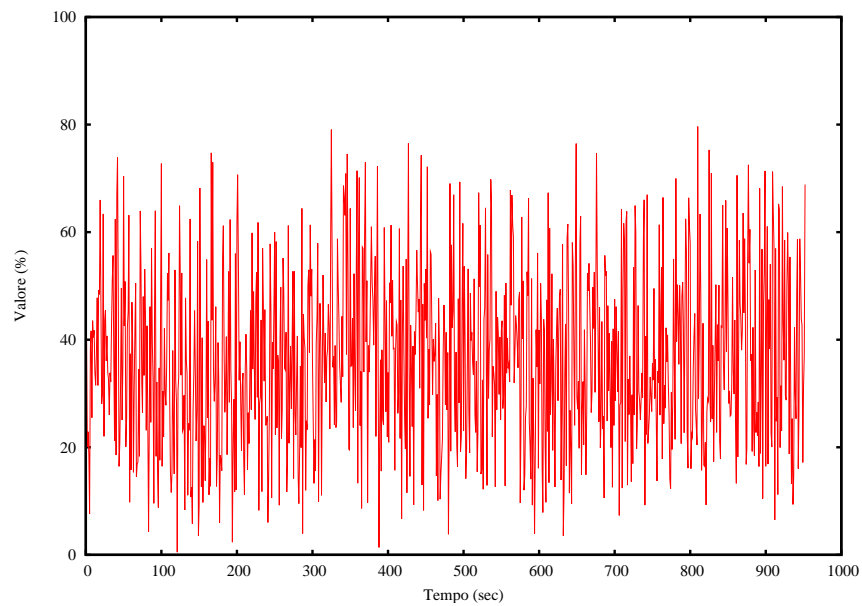


Figura 6.25: Latency terzo nodo

I valori in figura 6.23 è molto più basso rispetto ai valori in figura 6.24 e 6.25, a causa della tipologia di connessione diversa.

### 6.2.7 Reliability

Questo valore è simile alla latenza ed anch'esso dipende fortemente dal tipo di connessione e dalla banda ma non dall'SC, il quale non influisce sulle caratteristiche della qualità della connessione. Nelle figure si può notare la differenza di valori tra il caso 6.26 e i restanti.

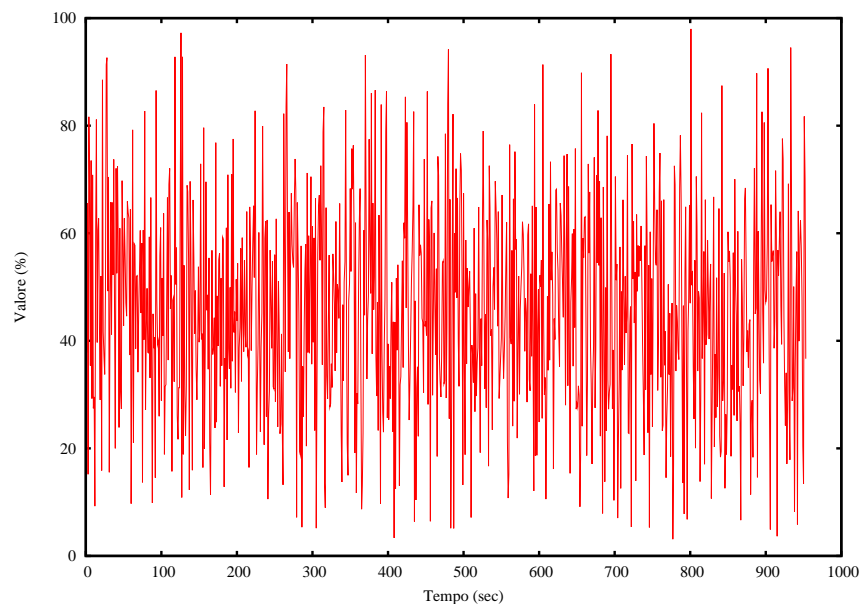


Figura 6.26: Reliability primo nodo

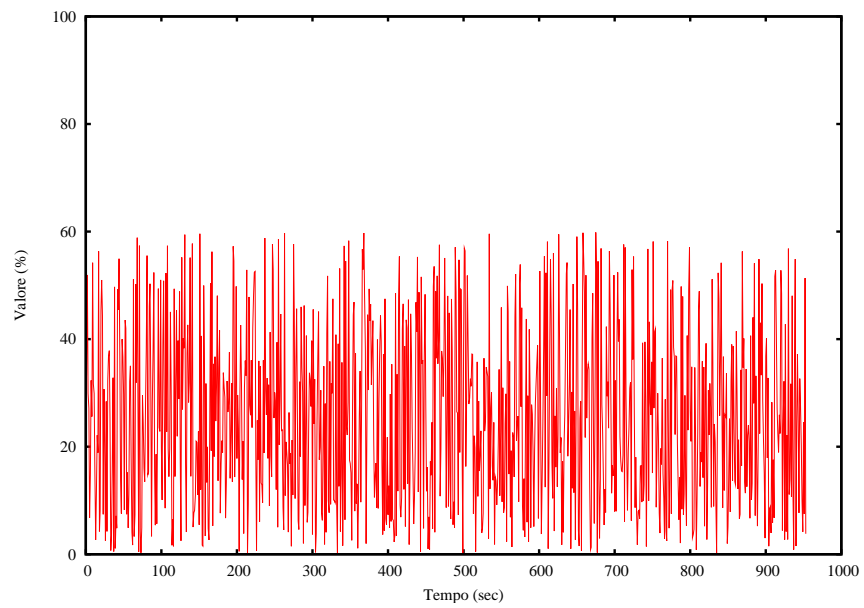


Figura 6.27: Reliability secondo nodo

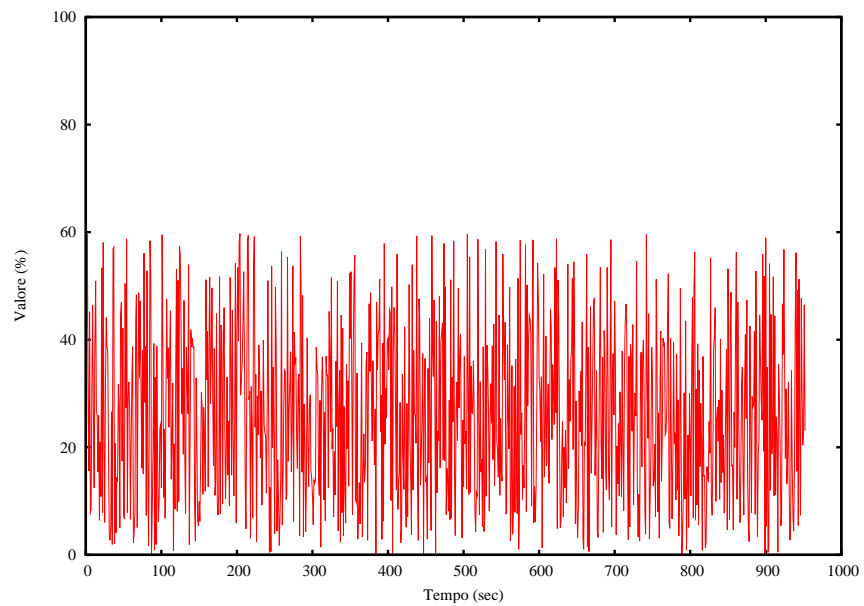


Figura 6.28: Reliability terzo nodo

### **6.2.8 ReqInterval**

Quest'ultimo tipo di valore è strettamente casuale e può assumere valori tra lo 0% e il 100%.



## Capitolo 7

### Conclusioni e sviluppi futuri

Il sistema realizzato ha consentito di valutare la presenza di un container dsm in un ambiente mobile con contratti SLA. Nello specifico si è valutato con quale frequenza lo SC e il DSM migrano tra differenti nodi. Sono stati eseguiti più test e si è variato i parametri per raggiungere uno scenario il più reale possibile, cercando un compromesso tra efficienza e qualità. Dalle simulazioni effettuate si è giunti alla conclusione che una migrazione tra i vari nodi consente un aumento dell'efficienza di tutti i nodi, in quanto ognuno partecipa alla computazione equamente dando il proprio contributo. Il contributo è proporzionale al tipo di nodo, ovvero dipende dal tipo di connessione, dalla banda e dalla presenza della rete elettrica. Infine si è pensato che in futuro sarebbero possibili ulteriori sviluppi come l'aggiunta di altre politiche di migrazione, anche se l'attuale ha dato ottimi risultati, inoltre un'altra miglioria potrebbe essere apportata al DSM, cercando di evitare la serializzazione completa su database, che potrebbe essere evitata serializzando solamente le scritture.

# Capitolo 8

## Appendice

### 8.1 SLARequesterBehaviour.java

```
package slash.contextmanager.behaviour;

import jade.core.AID;
import jade.core.behaviours.OneShotBehaviour;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import slash.contextmanager.agent.ContextManagerAgent;
import slash.df.DFUtil;
import slash.dsm.client.DsmClient;

public class SLARequesterBehaviour extends OneShotBehaviour {

    private static final long serialVersionUID = 1897715821469100876L;

    private DsmClient dsmClient;
    private ContextManagerAgent cm;

    public SLARequesterBehaviour(ContextManagerAgent agent) {
        this.cm = agent;
        dsmClient = new DsmClient(agent);
    }

    public void action() {
        try {
            DFAgentDescription[] res = DFUtil.search(myAgent, "
                publisher");
            int index = (int)(Math.random()%(res.length-1));
            AID publisher = res[index].getName();
```

```
        System.out.println("Selected publisher: "+publisher.
            getLocalName());

        dsmClient.out("slacontract", "slacontract-request",
            myAgent.getLocalName());
    } catch (Exception e) {
        e.printStackTrace();
    }
}

}
```

## 8.2 SLAReqReceiverBehaviour.java

```
package slash.contextmanager.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.contextmanager.agent.ContextManagerAgent;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.entity.SLAContract;
import slash.util.PropertiesReader;

public class SLAReqReceiverBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 1897715821469100876L;

    private DsmClient dsmClient;
    private ContextManagerAgent cm;

    public SLAReqReceiverBehaviour(ContextManagerAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            slareqreceiver.tick")));
        this.cm = agent;
        this.dsmClient = new DsmClient(agent);
    }

    public float genLatency() {
        return (float)((Math.random()*100)%30)+70;
    }

    public float genReliability() {
        return (float)((Math.random()*100)%30)+70;
    }
}
```

```
public float genReqInterval() {
    return (float)((Math.random()*100)%30)+70;
}

protected void onTick() {
    Tuple req;
    if((req=(Tuple)dsmClient.in("slacontract", "slacontract-
        request"))!=null) {
        if(req.getValue()!=null) {
            System.out.println("SLAContract-request
                received from "+req.getValue());
            AID requester = new AID((String)req.getValue
                (), AID.ISLOCALNAME);
            SLAContract contract = new SLAContract(
                myAgent.getAID(), requester, this.
                genLatency(), this.genReliability(),
                this.genReqInterval());
            dsmClient.out("slacontract", "slacontract",
                contract);
        }
    }
}
```

## 8.3 ContextProducerBehaviour.java

```
package slash.contextmanager.behaviour;

import jade.core.behaviours.TickerBehaviour;
import slash.contextmanager.agent.ContextManagerAgent;
import slash.dsm.client.DsmClient;
import slash.util.PropertiesReader;

public class ContextProducerBehaviour extends TickerBehaviour{

    private static final long serialVersionUID = -884539926477308910L;

    private ContextManagerAgent cm;
    private DsmClient dsmClient;

    public ContextProducerBehaviour(ContextManagerAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            contextproducer.tick")));
        this.cm = agent;
        this.dsmClient = new DsmClient(agent);
    }
}
```

```
        protected void onTick() {
            dsmClient.out(myAgent.getLocalName(), "context", cm.getContext());
        }
    }
}
```

## 8.4 ViolationConsumerBehaviour.java

```
package slash.contextmanager.behaviour;

import jade.core.behaviours.TickerBehaviour;
import slash.contextmanager.agent.ContextManagerAgent;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.util.PropertiesReader;

public class ViolationConsumerBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -5411307240223204565L;

    private ContextManagerAgent cm;
    private DsmClient dsmClient;

    public ViolationConsumerBehaviour(ContextManagerAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            violationreceiver.tick")));
        this.cm = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
        Tuple tuple = dsmClient.in(myAgent.getLocalName(), "
            slacontract-violation");
        if(tuple!=null && tuple.getValue()!=null)
            System.out.println("Contract violated!!!");
    }
}
```

## 8.5 RmConsumerBehaviour.java

```
package slash.contextmanager.behaviour;
```

```
import jade.core.behaviours.TickerBehaviour;
import slash.contextmanager.agent.ContextManagerAgent;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.util.PropertiesReader;

public class RmConsumerBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 6020734463253999461L;

    private ContextManagerAgent agent;
    private DsmClient dsmClient;

    public RmConsumerBehaviour(ContextManagerAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            resourceconsumer.tick")));
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
        Tuple tuple = dsmClient.read(myAgent.getLocalName(), "rm-cpu
            ");
        if(tuple!=null) {
            agent.getContext().addCpuValue((Float)tuple.getValue
                ());
        }

        tuple = dsmClient.read(myAgent.getLocalName(), "rm-energy");
        if(tuple!=null)
            agent.getContext().addEnergyValue((Float)tuple.
                getValue());

        tuple = dsmClient.read(myAgent.getLocalName(), "rm-memory");
        if(tuple!=null)
            agent.getContext().addMemoryValue((Float)tuple.
                getValue());

        tuple = dsmClient.read(myAgent.getLocalName(), "rm-ram");
        if(tuple!=null)
            agent.getContext().addRamValue((Float)tuple.getValue
                ());

        tuple = dsmClient.read(myAgent.getLocalName(), "rm-latency")
            ;
        if(tuple!=null)
```

```
agent.getContext().addLatencyValue((Float)tuple.
    getValue());

tuple = dsmClient.read(myAgent.getLocalName(), "rm-
    reliability");
if(tuple!=null)
    agent.getContext().addReliabilityValue((Float)tuple.
        getValue());

tuple = dsmClient.read(myAgent.getLocalName(), "rm-
    reqInterval");
if(tuple!=null)
    agent.getContext().addReqIntervalValue((Float)tuple.
        getValue());
}
}
```

## 8.6 ContextManagerAgent.java

```
package slash.contextmanager.agent;

import jade.core.Agent;
import slash.contextmanager.behaviour.ContextProducerBehaviour;
import slash.contextmanager.behaviour.RmConsumerBehaviour;
import slash.contextmanager.behaviour.SLAReqReceiverBehaviour;
import slash.contextmanager.behaviour.SLARequesterBehaviour;
import slash.contextmanager.behaviour.ViolationConsumerBehaviour;
import slash.df.DFUtil;
import slash.entity.Context;

public class ContextManagerAgent extends Agent {

    private static final long serialVersionUID = 5985223370274396652L;

    private Context context;

    protected void setup() {
        Object[] args = this.getArguments();
        System.out.println("CpuRmAgent: "+this.getName()+" , args len
            : "+args.length);
        if(args.length>0 && args[0].toString().equals("publisher"))
        {
            DFUtil.register(this, this.getLocalName(), "
                publisher");
        }
    }
}
```

```
        this.addBehaviour(new SLAReqReceiverBehaviour(this))
        ;
    }
    else if(args.length>0 && args[0].toString().equals("
        subscriber")) {
        this.addBehaviour(new SLARequesterBehaviour(this));
    }

    System.out.println("ContextManagerAgent: "+this.getName());
    this.context = new Context();
    this.context.setLocation(this.here());
    this.addBehaviour(new RmConsumerBehaviour(this));
    this.addBehaviour(new ViolationConsumerBehaviour(this));
    this.addBehaviour(new ContextProducerBehaviour(this));

}

public Context getContext() {
    return this.context;
}

public void setContext(Context context) {
    this.context = context;
}
}
```

## 8.7 CpuResourceBehaviour.java

```
package slash.resourcemonitor.behaviour;

import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.resourcemonitor.agent.CpuRmAgent;
import slash.util.PropertiesReader;

public class CpuResourceBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 6020734463253999461L;

    private CpuRmAgent agent;
    private DsmClient dsmClient;

    public CpuResourceBehaviour(CpuRmAgent agent) {
```



```
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            resourceconsumer.tick")));
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
        Tuple tuple = dsmClient.read(myAgent.getLocalName(), "cpu");
        if(tuple!=null) {
            dsmClient.update(myAgent.getLocalName(), "rm-cpu",
                tuple.getValue());
        }
    }
}
```

## 8.8 RamResourceBehaviour.java

```
package slash.resourcemonitor.behaviour;

import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.resourcemonitor.agent.RamRmAgent;
import slash.util.PropertiesReader;

public class RamResourceBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 6020734463253999461L;

    private RamRmAgent agent;
    private DsmClient dsmClient;

    public RamResourceBehaviour(RamRmAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            resourceconsumer.tick")));
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
        Tuple tuple = dsmClient.read(myAgent.getLocalName(), "ram");
        if(tuple!=null) {
            dsmClient.update(myAgent.getLocalName(), "rm-ram",
                tuple.getValue());
        }
    }
}
```

```
    }  
}
```

## 8.9 MemoryResourceBehaviour.java

```
package slash.resourcemonitor.behaviour;  
  
import jade.core.behaviours.TickerBehaviour;  
import slash.dsm.client.DsmClient;  
import slash.dsm.tuple.Tuple;  
import slash.resourcemonitor.agent.MemoryRmAgent;  
import slash.util.PropertiesReader;  
  
public class MemoryResourceBehaviour extends TickerBehaviour {  
  
    private static final long serialVersionUID = 6020734463253999461L;  
  
    private MemoryRmAgent agent;  
    private DsmClient dsmClient;  
  
    public MemoryResourceBehaviour(MemoryRmAgent agent) {  
        super(agent, Integer.parseInt(PropertiesReader.getProperty("resourceconsumer.tick")));  
        this.agent = agent;  
        this.dsmClient = new DsmClient(agent);  
    }  
  
    protected void onTick() {  
        Tuple tuple = dsmClient.read(myAgent.getLocalName(), "memory");  
        if(tuple!=null) {  
            dsmClient.update(myAgent.getLocalName(), "rm-memory", tuple.getValue());  
        }  
    }  
}
```

## 8.10 EnergyResourceBehaviour.java

```
package slash.resourcemonitor.behaviour;
```

```
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.resourcemonitor.agent.EnergyRmAgent;
import slash.util.PropertiesReader;

public class EnergyResourceBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 6020734463253999461L;

    private EnergyRmAgent agent;
    private DsmClient dsmClient;

    public EnergyResourceBehaviour(EnergyRmAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            resourceconsumer.tick")));
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
        Tuple tuple = dsmClient.read(myAgent.getLocalName(), "energy
            ");
        if(tuple!=null) {
            dsmClient.update(myAgent.getLocalName(), "rm-energy"
                , tuple.getValue());
        }
    }
}
```

## 8.11 LatencyResourceBehaviour.java

```
package slash.resourcemonitor.behaviour;

import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.resourcemonitor.agent.LatencyRmAgent;
import slash.util.PropertiesReader;

public class LatencyResourceBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 6020734463253999461L;

    private LatencyRmAgent agent;
```

```
private DsmClient dsmClient;

public LatencyResourceBehaviour(LatencyRmAgent agent) {
    super(agent, Integer.parseInt(PropertiesReader.getProperty("
        resourceconsumer.tick")));
    this.agent = agent;
    this.dsmClient = new DsmClient(agent);
}

protected void onTick() {
    Tuple tuple = dsmClient.read(myAgent.getLocalName(), "
        latency");
    if(tuple!=null) {
        dsmClient.update(myAgent.getLocalName(), "rm-latency
            ", tuple.getValue());
    }
}
}
```

## 8.12 ReliabilityResourceBehaviour.java

```
package slash.resourcemonitor.behaviour;

import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.resourcemonitor.agent.ReliabilityRmAgent;
import slash.util.PropertiesReader;

public class ReliabilityResourceBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 6020734463253999461L;

    private ReliabilityRmAgent agent;
    private DsmClient dsmClient;

    public ReliabilityResourceBehaviour(ReliabilityRmAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            resourceconsumer.tick")));
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
```

```
        Tuple tuple = dsmClient.read(myAgent.getLocalName(), "
            reliability");
        if(tuple!=null) {
            dsmClient.update(myAgent.getLocalName(), "rm-
                reliability", tuple.getValue());
        }
    }
}
```

## 8.13 ReqIntervalResourceBehaviour.java

```
package slash.resourcemonitor.behaviour;

import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.resourcemonitor.agent.ReqIntervalRmAgent;
import slash.util.PropertiesReader;

public class ReqIntervalResourceBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 6020734463253999461L;

    private ReqIntervalRmAgent agent;
    private DsmClient dsmClient;

    public ReqIntervalResourceBehaviour(ReqIntervalRmAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            resourceconsumer.tick")));
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
        Tuple tuple = dsmClient.read(myAgent.getLocalName(), "
            reqInterval");
        if(tuple!=null) {
            dsmClient.update(myAgent.getLocalName(), "rm-
                reqInterval", tuple.getValue());
        }
    }
}
```

## 8.14 RamRmAgent.java

```
package slash.resourcemonitor.agent;

import jade.core.Agent;
import slash.resourcemonitor.behaviour.RamResourceBehaviour;

public class RamRmAgent extends Agent {

    private static final long serialVersionUID = 2514514536861766799L;

    protected void setup() {
        this.addBehaviour(new RamResourceBehaviour(this));
    }
}
```

## 8.15 MemoryRmAgent.java

```
package slash.resourcemonitor.agent;

import jade.core.Agent;
import slash.resourcemonitor.behaviour.MemoryResourceBehaviour;

public class MemoryRmAgent extends Agent {

    private static final long serialVersionUID = 2514514536861766799L;

    protected void setup() {
        this.addBehaviour(new MemoryResourceBehaviour(this));
    }
}
```

## 8.16 EnergyRmAgent.java

```
package slash.resourcemonitor.agent;

import jade.core.Agent;
import slash.resourcemonitor.behaviour.EnergyResourceBehaviour;

public class EnergyRmAgent extends Agent {

    private static final long serialVersionUID = 2514514536861766799L;
```

```
        protected void setup() {
            this.addBehaviour(new EnergyResourceBehaviour(this));
        }
    }
```

## 8.17 CpuRmAgent.java

```
package slash.resourcemonitor.agent;

import jade.core.Agent;
import slash.resourcemonitor.behaviour.CpuResourceBehaviour;

public class CpuRmAgent extends Agent {

    private static final long serialVersionUID = 2514514536861766799L;

    protected void setup() {
        this.addBehaviour(new CpuResourceBehaviour(this));
    }
}
```

## 8.18 LatencyRmAgent.java

```
package slash.resourcemonitor.agent;

import jade.core.Agent;
import slash.resourcemonitor.behaviour.LatencyResourceBehaviour;

public class LatencyRmAgent extends Agent {

    private static final long serialVersionUID = 2514514536861766799L;

    protected void setup() {
        this.addBehaviour(new LatencyResourceBehaviour(this));
    }
}
```

## 8.19 ReliabilityRmAgent.java

```
package slash.resourcemonitor.agent;
```

```
import jade.core.Agent;
import slash.resourcemonitor.behaviour.ReliabilityResourceBehaviour;

public class ReliabilityRmAgent extends Agent {

    private static final long serialVersionUID = 2514514536861766799L;

    protected void setup() {
        this.addBehaviour(new ReliabilityResourceBehaviour(this));
    }
}
```

## 8.20 ReqIntervalRmAgent.java

```
package slash.resourcemonitor.agent;

import jade.core.Agent;
import slash.resourcemonitor.behaviour.ReqIntervalResourceBehaviour;

public class ReqIntervalRmAgent extends Agent {

    private static final long serialVersionUID = 2514514536861766799L;

    protected void setup() {
        this.addBehaviour(new ReqIntervalResourceBehaviour(this));
    }
}
```

## 8.21 AgentLauncher.java

```
package slash.boot;

import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.core.Runtime;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;

public class AgentLauncher {

    private static int counter = 1;
    private static boolean scEnabled = false;
```



```
private static boolean dsmEnabled = false;

public static void launchAgent(String name, String path, Object[]
    args, ContainerController cc) {
    try {
        AgentController agent = cc.createNewAgent(name, path
            , args);
        agent.start();
    } catch (StaleProxyException e) {
        e.printStackTrace();
    }
}

public static ContainerController createContainer() {
    Runtime rt = Runtime.instance();
    Profile p = new ProfileImpl();
    ContainerController cc = rt.createAgentContainer(p);
    return cc;
}

public static void launchNode(String network, int bandwidth, String
    type) {

    ContainerController cc = createContainer();

    Object[] arg = new Object[2];
    arg[0] = network;
    arg[1] = Integer.valueOf(bandwidth);

    if(!dsmEnabled) {
        launchAgent("dsm", "slash.dsm.agent.DsmServerAgent",
            null, cc);
        dsmEnabled = true;
    }

    if(!scEnabled) {
        launchAgent("sc", "slash.slachecker.agent.
            SLACheckerAgent", null, cc);
        scEnabled = true;
    }
    launchAgent("cm"+counter, "slash.contextmanager.agent.
        ContextManagerAgent", new Object[]{type}, cc);

    launchAgent("cpu"+counter, "slash.resource.agent.CpuAgent",
        arg, cc);
}
```

```
        launchAgent("energy"+counter, "slash.resource.agent.
            EnergyAgent", arg, cc);
        launchAgent("memory"+counter, "slash.resource.agent.
            MemoryAgent", arg, cc);
        launchAgent("ram"+counter, "slash.resource.agent.RamAgent",
            arg, cc);

        launchAgent("latency"+counter, "slash.resource.agent.
            LatencyAgent", arg, cc);
        launchAgent("reliability"+counter, "slash.resource.agent.
            ReliabilityAgent", arg, cc);
        launchAgent("reqInterval"+counter, "slash.resource.agent.
            ReqIntervalAgent", arg, cc);

        launchAgent("rm-cpu"+counter, "slash.resourcemonitor.agent.
            CpuRmAgent", arg, cc);
        launchAgent("rm-energy"+counter, "slash.resourcemonitor.
            agent.EnergyRmAgent", arg, cc);
        launchAgent("rm-memory"+counter, "slash.resourcemonitor.
            agent.MemoryRmAgent", arg, cc);
        launchAgent("rm-ram"+counter, "slash.resourcemonitor.agent.
            RamRmAgent", arg, cc);

        launchAgent("rm-latency"+counter, "slash.resourcemonitor.
            agent.LatencyRmAgent", arg, cc);
        launchAgent("rm-reliability"+counter, "slash.resourcemonitor
            .agent.ReliabilityRmAgent", arg, cc);
        launchAgent("rm-reqInterval"+counter, "slash.resourcemonitor
            .agent.ReqIntervalRmAgent", arg, cc);

        counter++;
    }

    public static void main(String[] args) {

        launchNode("wired", 512, "publisher");
        launchNode("wireless", 256, "subscriber");
        launchNode("wireless", 256, "subscriber");
    }
}
```

## 8.22 SLAReceiverBehaviour.java

```
package slash.slachecker.behaviour;
```

```
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.entity.SLAContract;
import slash.slachecker.agent.SLACheckerAgent;
import slash.util.PropertiesReader;

public class SLAReceiverBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -3897026802009387366L;

    private SLACheckerAgent sc;
    private DsmClient dsmClient;

    public SLAReceiverBehaviour(SLACheckerAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("slareceiver.tick")));
        this.sc = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
        Tuple tuple = dsmClient.in("slacontract", "slacontract");
        if(tuple!=null && tuple.getValue()!=null) {
            System.out.println("Added Contract: "+((SLAContract)
                tuple.getValue()).getLatency());
            sc.getContractList().add((SLAContract)tuple.getValue());
        }
    }
}
```

## 8.23 SLACheckerBehaviour.java

```
package slash.slachecker.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;

import java.util.Enumeration;

import slash.dsm.client.DsmClient;
import slash.entity.Context;
import slash.entity.Notify;
import slash.entity.SLAContract;
import slash.slachecker.agent.SLACheckerAgent;
```

```
import slash.util.DataWriter;
import slash.util.PropertiesReader;

public class SLACheckerBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -500981357584073158L;

    private SLACheckerAgent sc;
    private DsmClient dsmClient;
    private int check;

    public SLACheckerBehaviour(SLACheckerAgent sc) {
        super(sc, Integer.parseInt(PropertiesReader.getProperty("
            slachecker.tick")));
        this.check = Integer.parseInt(PropertiesReader.getProperty("
            slachecker.check"));
        this.sc = sc;
        this.dsmClient = new DsmClient(sc);
    }

    protected void onTick() {
        System.out.println("Sono su "+myAgent.getName());

        for(int i=0; i<sc.getContractList().size(); i++) {
            SLAContract contract = sc.getContractList().get(i);
            Context status = sc.getContextTable().get(contract.
                getPublisher());
            Context statusSub = sc.getContextTable().get(
                contract.getSubscriber());
            float lat=0, rel=0, req=0;
            if((status!=null) && (((lat=status.getAvgLatency())
                > contract.getLatency()) || ((rel=status.
                getAvgReliability()) > contract.getReliability())
                || ((req=statusSub.getAvgReqInterval()) >
                contract.getReqInterval())) {

                System.out.println("Violated with latency "+
                    lat+" > "+contract.getLatency()+",
                    reliability "+rel+" > "+contract.
                    getReliability()+", reqInterval "+req+"
                    > "+contract.getReqInterval());

                dsmClient.out(contract.getPublisher().
                    getLocalName(), "slacontract-violation",
                    contract);
                dsmClient.out(contract.getSubscriber().
                    getLocalName(), "slacontract-violation",
```

```
contract);  
    }  
}  
  
AID best = getBestNode();  
  
AID cm = new AID("cm"+sc.getAssociatedID(), AID.ISLOCALNAME)  
    ;  
System.out.println("Associated with: "+cm.getLocalName());  
Context context = sc.getContextTable().get(cm);  
  
float avgCpu, avgRam, avgMemory, avgEnergy;  
if(context!=null) {  
    avgCpu = context.getAvgCpu();  
    avgRam = context.getAvgRam();  
    avgMemory = context.getAvgMemory();  
    avgEnergy = context.getAvgEnergy();  
    System.out.println("Actual context --> cpu: "+avgCpu+  
        ", ram: "+avgRam+", memory: "+avgMemory+  
        ", energy: "+avgEnergy+", index: "+context.  
        calcIndex());  
    if(context.calcIndex() > check) { //52 originale  
        if(best!=null) {  
            System.out.println("queue: "+myAgent  
                .getCurQueueSize());  
            myAgent.doMove(sc.getContextTable().  
                get(best).getLocation());  
            int bestN = Integer.parseInt(String.  
                valueOf(best.getLocalName().  
                    charAt(best.getLocalName().  
                        length()-1)));  
            System.out.println("Migration to "+  
                best.getLocalName());  
  
            Notify notify = new Notify(sc.  
                getAssociatedID(), bestN);  
            dsmClient.update("notify", "notify",  
                notify);  
            sc.setAssociatedID(bestN);  
        }  
        else  
            System.out.println("IMPOSSIBILE  
                MIGRARE!!!");  
    }  
}  
}
```

```
public AID getBestNode() {
    AID best = null;
    AID next;
    Enumeration<AID> keys = sc.getContextTable().keys();
    Context context;
    float avgCpu, avgRam, avgMemory, avgEnergy;
    float total=500, iter=0;

    while(keys.hasMoreElements()) {
        next = keys.nextElement();
        context = sc.getContextTable().get(next);
        if(context!=null) {
            avgCpu = context.getAvgCpu();
            avgRam = context.getAvgRam();
            avgMemory = context.getAvgMemory();
            avgEnergy = context.getAvgEnergy();
            iter = avgCpu + avgRam + avgMemory + (1-
                avgEnergy);
            DataWriter.writeData("index"+next.
                getLocalName().charAt(2), context.
                calcIndex());
            System.out.println("Context["+next.
                getLocalName()+"]-> cpu: "+avgCpu+",
                ram: "+avgRam+", memory: "+avgMemory+",
                energy: "+avgEnergy+", index: "+context.
                calcIndex());
            if(avgCpu < Context.CPU_LIMIT && avgRam <
                Context.RAM_LIMIT &&
                    avgMemory < Context.
                        MEMORY_LIMIT &&
                        avgEnergy > Context.
                            ENERGY_LIMIT &&
                            iter < total) {
                best = next;
                total = iter;
            }
        }
    }

    return best;
}
```

## 8.24 SLAStarterBehaviour.java

```
package slash.slachecker.behaviour;
```

```
import jade.core.behaviours.OneShotBehaviour;
import slash.slachecker.agent.SLACheckerAgent;
import slash.slachecker.util.MigrationUtil;

public class SLAStarterBehaviour extends OneShotBehaviour {

    private static final long serialVersionUID = -1382758463374415598L;

    private SLACheckerAgent sc;

    public void action() {
        MigrationUtil.notifyMigration(myAgent, 0, 1); //Avvertire il
                                                    primo nodo della presenza dello sc
    }
}
```

## 8.25 ContextConsumerBehaviour.java

```
package slash.slachecker.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.entity.Context;
import slash.entity.SLAContract;
import slash.slachecker.agent.SLACheckerAgent;
import slash.util.PropertiesReader;

public class ContextConsumerBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 5235996069711181357L;

    private SLACheckerAgent sc;
    private DsmClient dsmClient;

    public ContextConsumerBehaviour(SLACheckerAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            contextconsumer.tick")));
        this.sc = agent;
        this.dsmClient = new DsmClient(agent);
    }

    protected void onTick() {
        for(int i=0; i<sc.getContractList().size(); i++) {
```

```
SLAContract contract = sc.getContractList().get(i);

Tuple tuple = dsmClient.in(contract.getPublisher().
    getLocalName(), "context");
if(tuple!=null && tuple.getValue()!=null) {
    this.sc.getContextTable().put(new AID("cm"+
        tuple.getIndex(), AID.ISLOCALNAME), (
            Context)tuple.getValue());
}

tuple = dsmClient.in(contract.getSubscriber().
    getLocalName(), "context");
if(tuple!=null && tuple.getValue()!=null) {
    this.sc.getContextTable().put(new AID("cm"+
        tuple.getIndex(), AID.ISLOCALNAME), (
            Context)tuple.getValue());
}
}
}
```

## 8.26 MigrationUtil.java

```
package slash.slachecker.util;

import jade.core.Agent;
import slash.dsm.client.DsmClient;
import slash.entity.Notify;

public class MigrationUtil {

    private static DsmClient dsmClient = null;

    public static void notifyMigration(Agent agent, int src, int dest) {
        if(dsmClient==null)
            dsmClient = new DsmClient(agent);
        Notify notify = new Notify(src, dest);

        dsmClient.out("notify", "notify", notify);
    }
}
```

## 8.27 SLACheckerAgent.java



```
/**
 *
 */
package slash.slachecker.agent;

import jade.core.AID;
import jade.core.Agent;

import java.util.Hashtable;
import java.util.LinkedList;
import java.util.List;

import slash.df.DFUtil;
import slash.entity.Context;
import slash.entity.SLAContract;
import slash.slachecker.behaviour.ContextConsumerBehaviour;
import slash.slachecker.behaviour.SLACheckerBehaviour;
import slash.slachecker.behaviour.SLAReceiverBehaviour;
import slash.slachecker.behaviour.SLAStarterBehaviour;

public class SLACheckerAgent extends Agent {

    private static final long serialVersionUID = -7918542991436312908L;

    private List<SLAContract> contractList = null;
    private Hashtable<AID, Context> contextTable = null;

    private int associatedID;

    public void setAssociatedID(int ID) {
        this.associatedID = ID;
    }

    public int getAssociatedID() {
        return this.associatedID;
    }

    public List<SLAContract> getContractList() {
        return this.contractList;
    }

    public Hashtable<AID, Context> getContextTable() {
        return this.contextTable;
    }

    protected void setup() {
        this.contractList = new LinkedList<SLAContract>();
    }
}
```

```
        this.contextTable = new Hashtable<AID, Context>();

        this.associatedID = 1;

        System.out.println("SLaChecker: "+this.getName());
        DFUtil.register(this, this.getLocalName(), "sla-checker");

        this.addBehaviour(new SLAStarterBehaviour());
        this.addBehaviour(new ContextConsumerBehaviour(this));
        this.addBehaviour(new SLAReceiverBehaviour(this));
        this.addBehaviour(new SLACheckerBehaviour(this));
    }

    protected void takeDown() {
        DFUtil.deregister(this, this.getLocalName(), "sla-checker");
    }
}
```

## 8.28 ReqIntervalBehaviour.java

```
package slash.resource.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.resource.agent.ReqIntervalAgent;
import slash.util.DataWriter;
import slash.util.PropertiesReader;

public class ReqIntervalBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -3231027298580344751L;

    private float reqInterval;

    private AID rmAid;
    private ReqIntervalAgent agent;
    private DsmClient dsmClient;

    public ReqIntervalBehaviour(AID cmAid, ReqIntervalAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            reqinterval.tick")));
        this.rmAid = cmAid;
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }
}
```

```
private float generate() {
    reqInterval = (float)((Math.random()*100)%100);
    return reqInterval;
}

protected void onTick() {
    generate();
    DataWriter.writeData(myAgent.getLocalName(), reqInterval);
    Float reqIntervalStr = reqInterval;
    dsmClient.update(agent.getLocalName(), "reqInterval", reqIntervalStr
    );
}
}
```

## 8.29 NotifyReceiverBehaviour.java

```
package slash.resource.behaviour;

import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.dsm.tuple.Tuple;
import slash.entity.Notify;
import slash.resource.agent.ResourceAgent;
import slash.util.PropertiesReader;

public class NotifyReceiverBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = 4984068838158410640L;

    private ResourceAgent ra;
    private DsmClient dsmClient;

    public NotifyReceiverBehaviour(ResourceAgent ra) {
        super(ra, Integer.parseInt(PropertiesReader.getProperty("
            notify.tick")));
        this.ra = ra;
        this.dsmClient = new DsmClient(ra);
    }

    protected void onTick() {
        Tuple tuple = dsmClient.read("notify", "notify");
        Notify notify = null;
        if(tuple!=null) {
            notify = (Notify)tuple.getValue();
        }
    }
}
```

```
        if(notify!=null) {
            int num = Integer.valueOf(String.valueOf(myAgent.
                getLocalName().charAt(myAgent.getLocalName().
                    length()-1)));

            if(notify.getSrc() == num) {
                ra.setLocalSC(false);
            }
            else if(notify.getDest() == num) {
                ra.setLocalSC(true);
            }
            else
                ra.setLocalSC(false);
        }
    }
}
```

## 8.30 LatencyBehaviour.java

```
package slash.resource.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.entity.Context;
import slash.resource.agent.LatencyAgent;
import slash.util.DataWriter;
import slash.util.PropertiesReader;

public class LatencyBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -3231027298580344751L;

    private float latency;

    private AID rmAid;
    private LatencyAgent agent;
    private DsmClient dsmClient;

    public LatencyBehaviour(AID cmAid, LatencyAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            latency.tick")));
        this.rmAid = cmAid;
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }
}
```

```
    }

    private float generate() {
        latency = (float)((Math.random()*100)%60);
        if(agent.isLocalSC() || agent.getNetwork() == Context.
            WIRELESS) {
            latency += (float)((((Math.random()*100)/(agent.
                getBandwidth()/50))%40));
        }

        return latency;
    }

    protected void onTick() {
        generate();
        DataWriter.writeData(myAgent.getLocalName(), latency);
        Float latencyStr = latency;
        dsmClient.update(agent.getLocalName(), "latency", latencyStr);
    }
}
```

## 8.31 RamBehaviour.java

```
package slash.resource.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.resource.agent.RamAgent;
import slash.util.DataWriter;
import slash.util.PropertiesReader;

public class RamBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -3231027298580344751L;

    private float ram;

    private AID cmAid;
    private RamAgent agent;
    private DsmClient dsmClient;

    public RamBehaviour(AID cmAid, RamAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("
            ram.tick")));
        this.cmAid = cmAid;
    }
}
```

```
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }

    private float generate() {
        ram = (float)((Math.random()*100)%60);
        if(agent.isLocalSC()) {
            ram += 40;
        }

        return ram;
    }

    protected void onTick() {
        generate();
        DataWriter.writeData(myAgent.getLocalName(), ram);
        Float ramStr = ram;
        dsmClient.update(agent.getLocalName(), "ram", ramStr);
    }
}
```

## 8.32 CpuBehaviour.java

```
package slash.resource.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.resource.agent.CpuAgent;
import slash.util.DataWriter;
import slash.util.PropertiesReader;

public class CpuBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -3231027298580344751L;

    private float cpu;

    private AID cmAid;
    private CpuAgent agent;
    private DsmClient dsmClient;

    public CpuBehaviour(AID cmAid, CpuAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("cpu.tick")));
        this.cmAid = cmAid;
    }
}
```

```
        this.agent = agent;

        this.dsmClient = new DsmClient(agent);
    }

    private float generate() {
        cpu = (float)((Math.random()*100)%60);
        if(agent.isLocalSC()) {
            cpu += 40;
        }

        return cpu;
    }

    protected void onTick() {
        generate();
        DataWriter.writeData(myAgent.getLocalName(), cpu);
        Float cpuStr = cpu;
        dsmClient.update(agent.getLocalName(), "cpu", cpuStr);
    }
}
```

## 8.33 MemoryBehaviour.java

```
package slash.resource.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.resource.agent.MemoryAgent;
import slash.util.DataWriter;
import slash.util.PropertiesReader;

public class MemoryBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -3231027298580344751L;

    private float memory;

    private AID cmAid;
    private MemoryAgent agent;
    private DsmClient dsmClient;

    public MemoryBehaviour(AID cmAid, MemoryAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("memory.tick")));
    }
}
```

```
        this.cmAid = cmAid;
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
        this.memory = (float)((Math.random()*100)%60);
    }

    private float generate() {
        if(agent.isLocalSC())
            memory += 0.2*(float)((Math.random()*100)%10);
        else
            memory -= 0.2*(float)((Math.random()*100)%10);

        if(memory<0 || memory>100)
            memory = (float)((Math.random()*100)%60);

        return memory;
    }

    protected void onTick() {
        generate();
        DataWriter.writeData(myAgent.getLocalName(), memory);
        Float memoryStr = memory;
        dsmClient.update(agent.getLocalName(), "memory", memoryStr);
    }
}
```

## 8.34 EnergyBehaviour.java

```
package slash.resource.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.entity.Context;
import slash.resource.agent.EnergyAgent;
import slash.util.DataWriter;
import slash.util.PropertiesReader;

public class EnergyBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -3231027298580344751L;

    private float energy;
    private boolean powerOn;

    private AID cmAid;
```



```
private EnergyAgent agent;
private DsmClient dsmClient;

public EnergyBehaviour(AID cmAid, EnergyAgent agent) {
    super(agent, Integer.parseInt(PropertiesReader.getProperty("
        energy.tick")));
    this.cmAid = cmAid;
    this.agent = agent;
    this.powerOn = false;
    this.energy = (float)(Math.random()*60)+40;
    this.dsmClient = new DsmClient(agent);
}

private float generate() {

    if(agent.getNetwork() == Context.WIRED)
        energy = 100;
    else {
        if(powerOn) {
            energy += 0.8;
            if(agent.isLocalSC() && (agent.getNetwork()
                == Context.WIRELESS))
                energy -= 0.2;
        }
        else {
            energy -= 0.2;
            if(agent.isLocalSC() && (agent.getNetwork()
                == Context.WIRELESS))
                energy -= 0.8;
        }

        if(energy <=10) {
            powerOn = true;
        }

        if(energy >=99)
            powerOn = false;
    }

    return energy;
}

protected void onTick() {
    generate();
    DataWriter.writeData(myAgent.getLocalName(), energy);
    Float energyStr = energy;
```

```
        dsmClient.update(agent.getLocalName(), "energy", energyStr);
    }
}
```

## 8.35 ReliabilityBehaviour.java

```
package slash.resource.behaviour;

import jade.core.AID;
import jade.core.behaviours.TickerBehaviour;
import slash.dsm.client.DsmClient;
import slash.entity.Context;
import slash.resource.agent.ReliabilityAgent;
import slash.util.DataWriter;
import slash.util.PropertiesReader;

public class ReliabilityBehaviour extends TickerBehaviour {

    private static final long serialVersionUID = -3231027298580344751L;

    private float reliability;

    private AID rmAid;
    private ReliabilityAgent agent;
    private DsmClient dsmClient;

    public ReliabilityBehaviour(AID cmAid, ReliabilityAgent agent) {
        super(agent, Integer.parseInt(PropertiesReader.getProperty("reliability.tick")));
        this.rmAid = cmAid;
        this.agent = agent;
        this.dsmClient = new DsmClient(agent);
    }

    private float generate() {
        reliability = (float)((Math.random()*100)%60);
        if(agent.getNetwork() == Context.WIRED)
            reliability += (float)((((Math.random()*100)*(agent.getBandwidth()/50))%40);
        return reliability;
    }

    protected void onTick() {
        generate();
        DataWriter.writeData(myAgent.getLocalName(), reliability);
        Float reliabilityStr = reliability;
    }
}
```

```
        dsmClient.update(agent.getLocalName(), "reliability", reliabilityStr
    );
    }
}
```

## 8.36 CpuAgent.java

```
package slash.resource.agent;

import jade.core.AID;
import slash.entity.Context;
import slash.resource.behaviour.CpuBehaviour;
import slash.resource.behaviour.NotifyReceiverBehaviour;

public class CpuAgent extends ResourceAgent {

    private static final long serialVersionUID = -5570077702241336240L;

    private int network;
    private int bandwidth;

    protected void setup() {
        System.out.println("Cpu: "+this.getName());

        Object[] args = this.getArguments();
        if(args.length==2) {
            if(args[0].toString().equals("wired"))
                network = Context.WIRED;
            else
                network = Context.WIRELESS;

            bandwidth = (Integer)args[1];
        }

        AID cmAid = new AID("cm"+this.getLocalName().charAt(this.
            getLocalName().length()-1), AID.ISLOCALNAME);
        this.addBehaviour(new CpuBehaviour(cmAid, this));
        this.addBehaviour(new NotifyReceiverBehaviour(this));
    }

    public int getNetwork() {
        return this.network;
    }

    public int getBandwidth() {
```

```
        return this.bandwidth;
    }
}
```

## 8.37 ResourceAgent.java

```
package slash.resource.agent;

import jade.core.AID;
import jade.core.Agent;

public class ResourceAgent extends Agent {

    private static final long serialVersionUID = -8360934789152239831L;

    private boolean localSC = false;

    protected AID rm;

    public AID getRm() {
        return this.rm;
    }

    public boolean isLocalSC() {
        return this.localSC;
    }

    public void setLocalSC(boolean localSC) {
        this.localSC = localSC;
    }
}
```

## 8.38 MemoryAgent.java

```
package slash.resource.agent;

import jade.core.AID;
import slash.entity.Context;
import slash.resource.behaviour.MemoryBehaviour;
import slash.resource.behaviour.NotifyReceiverBehaviour;

public class MemoryAgent extends ResourceAgent {
```

```
private static final long serialVersionUID = -5491334259558404278L;

private int network;
private int bandwidth;

protected void setup() {
    System.out.println("Memory: "+this.getName());

    Object[] args = this.getArguments();
    if(args.length==2) {
        if(args[0].toString().equals("wired"))
            network = Context.WIRED;
        else
            network = Context.WIRELESS;

        bandwidth = (Integer)args[1];
    }

    AID cmAid = new AID("cm"+this.getLocalName().charAt(this.
        getLocalName().length()-1), AID.ISLOCALNAME);
    this.addBehaviour(new MemoryBehaviour(cmAid, this));
    this.addBehaviour(new NotifyReceiverBehaviour(this));
}

public int getNetwork() {
    return this.network;
}

public int getBandwidth() {
    return this.bandwidth;
}
}
```

## 8.39 EnergyAgent.java

```
package slash.resource.agent;

import jade.core.AID;
import slash.entity.Context;
import slash.resource.behaviour.EnergyBehaviour;
import slash.resource.behaviour.NotifyReceiverBehaviour;

public class EnergyAgent extends ResourceAgent {

    private static final long serialVersionUID = 7577668356120531127L;
```

```
private int network;
private int bandwidth;

protected void setup() {
    System.out.println("Energy: "+this.getName());

    Object[] args = this.getArguments();
    if(args.length==2) {
        if(args[0].toString().equals("wired"))
            network = Context.WIRED;
        else
            network = Context.WIRELESS;

        bandwidth = (Integer)args[1];
    }

    AID cmAid = new AID("cm"+this.getLocalName().charAt(this.
        getLocalName().length()-1), AID.ISLOCALNAME);
    this.addBehaviour(new EnergyBehaviour(cmAid, this));
    this.addBehaviour(new NotifyReceiverBehaviour(this));
}

public int getNetwork() {
    return this.network;
}

public int getBandwidth() {
    return this.bandwidth;
}
}
```

## 8.40 ReliabilityAgent.java

```
package slash.resource.agent;

import jade.core.AID;
import slash.entity.Context;
import slash.resource.behaviour.NotifyReceiverBehaviour;
import slash.resource.behaviour.ReliabilityBehaviour;

public class ReliabilityAgent extends ResourceAgent {

    private static final long serialVersionUID = -4576966086955385673L;
```

```
private int network;
private int bandwidth;

protected void setup() {
    System.out.println("Reliability: "+this.getName());

    Object[] args = this.getArguments();
    if(args.length==2) {
        if(args[0].toString().equals("wired"))
            network = Context.WIRED;
        else
            network = Context.WIRELESS;

        bandwidth = (Integer)args[1];
    }

    this.rm = new AID("rm"+this.getLocalName().charAt(this.
        getLocalName().length()-1), AID.ISLOCALNAME);
    this.addBehaviour(new ReliabilityBehaviour(rm, this));
    this.addBehaviour(new NotifyReceiverBehaviour(this));
}

public int getNetwork() {
    return this.network;
}

public int getBandwidth() {
    return this.bandwidth;
}
}
```

## 8.41 ReqIntervalAgent.java

```
package slash.resource.agent;

import jade.core.AID;
import slash.entity.Context;
import slash.resource.behaviour.NotifyReceiverBehaviour;
import slash.resource.behaviour.ReqIntervalBehaviour;

public class ReqIntervalAgent extends ResourceAgent {

    private static final long serialVersionUID = -2452255667787453718L;

    private int network;
```

```
private int bandwidth;

protected void setup() {
    System.out.println("ReqInterval: "+this.getName());

    Object[] args = this.getArguments();
    if(args.length==2) {
        if(args[0].toString().equals("wired"))
            network = Context.WIRED;
        else
            network = Context.WIRELESS;

        bandwidth = (Integer)args[1];
    }

    this.rm = new AID("rm"+this.getLocalName().charAt(this.
        getLocalName().length()-1), AID.ISLOCALNAME);
    this.addBehaviour(new ReqIntervalBehaviour(rm, this));
    this.addBehaviour(new NotifyReceiverBehaviour(this));
}

public int getNetwork() {
    return this.network;
}

public int getBandwidth() {
    return this.bandwidth;
}
}
```

## 8.42 LatencyAgent.java

```
package slash.resource.agent;

import jade.core.AID;
import slash.entity.Context;
import slash.resource.behaviour.LatencyBehaviour;
import slash.resource.behaviour.NotifyReceiverBehaviour;

public class LatencyAgent extends ResourceAgent {

    private static final long serialVersionUID = -5626212945073184408L;

    private int network;
    private int bandwidth;
```



```
protected void setup() {
    System.out.println("Latency: "+this.getName());

    Object[] args = this.getArguments();
    if(args.length==2) {
        if(args[0].toString().equals("wired"))
            network = Context.WIRED;
        else
            network = Context.WIRELESS;

        bandwidth = (Integer)args[1];
    }

    this.rm = new AID("rm"+this.getLocalName().charAt(this.
        getLocalName().length()-1), AID.ISLOCALNAME);
    this.addBehaviour(new LatencyBehaviour(rm, this));
    this.addBehaviour(new NotifyReceiverBehaviour(this));
}

public int getNetwork() {
    return this.network;
}

public int getBandwidth() {
    return this.bandwidth;
}
}
```

## 8.43 RamAgent.java

```
package slash.resource.agent;

import jade.core.AID;
import slash.entity.Context;
import slash.resource.behaviour.NotifyReceiverBehaviour;
import slash.resource.behaviour.RamBehaviour;

public class RamAgent extends ResourceAgent {

    private static final long serialVersionUID = -630447104765764031L;

    private int network;
    private int bandwidth;
```

```
protected void setup() {
    System.out.println("Ram: "+this.getName());

    Object[] args = this.getArguments();
    if(args.length==2) {
        if(args[0].toString().equals("wired"))
            network = Context.WIRED;
        else
            network = Context.WIRELESS;

        bandwidth = (Integer)args[1];
    }

    AID cmAid = new AID("cm"+this.getLocalName().charAt(this.
        getLocalName().length()-1), AID.ISLOCALNAME);
    this.addBehaviour(new RamBehaviour(cmAid, this));
    this.addBehaviour(new NotifyReceiverBehaviour(this));
}

public int getNetwork() {
    return this.network;
}

public int getBandwidth() {
    return this.bandwidth;
}
}
```

## 8.44 DFUtil.java

```
package slash.df;

import jade.core.AID;
import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.SearchConstraints;
import jade.domain.FIPAAgentManagement.ServiceDescription;

public class DFUtil {

    public static void register(Agent agent, String name, String type) {
        try {
```

```
        System.out.println("Registering name: "+name+", type
            : "+type);
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(agent.getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setName(name);
        sd.setType(type);
        dfd.addServices(sd);
        DFService.register(agent, dfd);
    } catch (FIPAException e) {
        e.printStackTrace();
    }
}

public static DFAgentDescription[] search(Agent agent, String type)
{
    try {
        System.out.println("Searching type: "+type);
        DFAgentDescription template = new DFAgentDescription
            ();
        ServiceDescription tsd = new ServiceDescription();
        tsd.setType(type);
        template.addServices(tsd);
        SearchConstraints sc = new SearchConstraints();
        DFAgentDescription[] res = DFService.search(agent,
            template, sc);
        return res;
    } catch (FIPAException e) {
        e.printStackTrace();
    }
    return null;
}

public static AID search(Agent agent, String name, String type) {
    try {
        DFAgentDescription[] res;
        do {
            System.out.println("Searching name: "+name+"
                , type: "+type);
            DFAgentDescription template = new
                DFAgentDescription();
            ServiceDescription tsd = new
                ServiceDescription();
            tsd.setName(name);
            tsd.setType(type);
            template.addServices(tsd);
```

```
        SearchConstraints sc = new SearchConstraints
            ();
        res = DFService.search(agent, template, sc);
    }
    while(res.length==0);
    return res[0].getName();
} catch (FIPAException e) {
    e.printStackTrace();
}
return null;
}

public static void deregister(Agent agent, String name, String type)
{
    try {
        System.out.println("Deregistering name: "+name+",
            type: "+type);
        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(agent.getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setName(name);
        sd.setType(type);
        dfd.addServices(sd);
        DFService.deregister(agent, dfd);
    } catch (FIPAException e) {
        e.printStackTrace();
    }
}
}
```

## 8.45 Context.java

```
package slash.entity;

import jade.core.Location;

import java.io.Serializable;
import java.util.LinkedList;
import java.util.List;

public class Context implements Serializable {

    private static final long serialVersionUID = -4290838847903200791L;

    public static int WIRED = 0;
    public static int WIRELESS = 1;
```

```
public static float CPU_LIMIT = 60.0f;
public static float RAM_LIMIT = 60.0f;
public static float MEMORY_LIMIT = 60.0f;
public static float ENERGY_LIMIT = 20.0f;

private static int RST_INTERVAL = 30;

private List<Float> cpuList;
private List<Float> ramList;
private List<Float> memoryList;
private List<Float> energyList;
private List<Float> latencyList;
private List<Float> reliabilityList;
private List<Float> reqIntervalList;

private Location location;
private float cpu;
private float ram;
private float memory;
private int network; //WIRED, WIRELESS
private int bandwidth;
private float energy;
private float latency;
private float reliability;
private float reqInterval;

public Context() {
    this.cpuList = new LinkedList<Float>();
    this.ramList = new LinkedList<Float>();
    this.memoryList = new LinkedList<Float>();
    this.energyList = new LinkedList<Float>();
    this.latencyList = new LinkedList<Float>();
    this.reliabilityList = new LinkedList<Float>();
    this.reqIntervalList = new LinkedList<Float>();
}

public void addCpuValue(Float cpuValue) {
    if(cpuValue!=null) {
        this.cpu = cpuValue;
    }
}

public void addRamValue(Float ramValue) {
    if(ramValue!=null)
        this.ram = ramValue;
}
```

```
public void addMemoryValue(Float memoryValue) {
    if(memoryValue!=null)
        this.memory = memoryValue;
}

public void addEnergyValue(Float energyValue) {
    if(energyValue!=null)
        this.energy = energyValue;
}

public float getAvgCpu() {
    return cpu;
}

public float getAvgRam() {
    return ram;
}

public float getAvgMemory() {
    return memory;
}

public float getAvgEnergy() {
    return energy;
}

public void addLatencyValue(Float latencyValue) {
    if(latencyValue!=null) {
        if(this.latencyList.size() >= Context.RST_INTERVAL)
            this.latencyList = new LinkedList<Float>();
        this.latencyList.add(Float.valueOf(latencyValue));
    }
}

public void addReliabilityValue(Float reliabilityValue) {
    if(reliabilityValue!=null) {
        if(this.reliabilityList.size() >= Context.
            RST_INTERVAL)
            this.reliabilityList = new LinkedList<Float>
                >();
        this.reliabilityList.add(Float.valueOf(
            reliabilityValue));
    }
}

public void addReqIntervalValue(Float reqIntervalValue) {
```

```
        if(reqIntervalValue!=null) {
            if(this.reqIntervalList.size() >= Context.
                RST_INTERVAL)
                this.reqIntervalList = new LinkedList<Float>
                    >();
            this.reqIntervalList.add(Float.valueOf(
                reqIntervalValue));
        }
    }

    public float getAvgLatency() {
        latency = 0;
        for(int i=0; i<this.latencyList.size(); i++) {
            latency += this.latencyList.get(i);
        }
        latency /= this.latencyList.size();
        return latency;
    }

    public float getAvgReliability() {
        reliability = 0;
        for(int i=0; i<this.reliabilityList.size(); i++) {
            reliability += this.reliabilityList.get(i);
        }
        reliability /= this.reliabilityList.size();
        return reliability;
    }

    public float getAvgReqInterval() {
        reqInterval = 0;
        for(int i=0; i<this.reqIntervalList.size(); i++) {
            reqInterval += this.reqIntervalList.get(i);
        }
        reqInterval /= this.reqIntervalList.size();
        return reqInterval;
    }

    public void setLocation(Location location) {
        this.location = location;
    }

    public Location getLocation() {
        return this.location;
    }

    public float calcIndex() {
        return cpu*0.23f+ram*0.23f+memory*0.23f+(100-energy)*0.3f;
```

```
}

public Context(float cpu, float ram, float memory, float energy, int
network, int bandwidth, float latency, float reliability, float
reqInterval) {
    this.cpu = cpu;
    this.ram = ram;
    this.memory = memory;
    this.energy = energy;
    this.network = network;
    this.bandwidth = bandwidth;
    this.latency = latency;
    this.reliability = reliability;
    this.reqInterval = reqInterval;
}

public void setCpu(float cpu) {
    this.cpu = cpu;
}

public float getCpu() {
    return this.cpu;
}

public void setRam(float ram) {
    this.ram = ram;
}

public float getRam() {
    return this.ram;
}

public void setMemory(float memory) {
    this.memory = memory;
}

public float getMemory() {
    return this.memory;
}

public void setNetwork(int network) {
    this.network = network;
}

public int getNetwork() {
    return this.network;
}
```



```
    public void setBandwidth(int bandwidth) {
        this.bandwidth = bandwidth;
    }

    public int getBandwidth() {
        return this.bandwidth;
    }

    public void setEnergy(float energy) {
        this.energy = energy;
    }

    public float getEnergy() {
        return this.energy;
    }
}
```

## 8.46 Notify.java

```
package slash.entity;

import java.io.Serializable;

public class Notify implements Serializable {

    private static final long serialVersionUID = 6443212295017254395L;

    private int src;
    private int dest;

    public Notify(int src, int dest) {
        this.src = src;
        this.dest = dest;
    }

    public void setSrc(int src) {
        this.src = src;
    }

    public int getSrc() {
        return this.src;
    }

    public void setDest(int dest) {
        this.dest = dest;
    }
}
```

```
    }

    public int getDest() {
        return this.dest;
    }
}
```

## 8.47 SLAContract.java

```
package slash.entity;

import jade.core.AID;

import java.io.Serializable;

public class SLAContract implements Serializable {

    private static final long serialVersionUID = 1280366527966115376L;

    private AID publisher;
    private AID subscriber;
    private float latency;
    private float reliability;
    private float reqInterval;

    public SLAContract(AID publisher, AID subscriber, float latency,
        float reliability, float reqInterval) {
        this.publisher = publisher;
        this.subscriber = subscriber;
        this.latency = latency;
        this.reliability = reliability;
        this.reqInterval = reqInterval;
    }

    public SLAContract() {
        this.publisher = null;
        this.subscriber = null;
        this.latency = 0.0f;
        this.reliability = 0.0f;
        this.reqInterval = 0.0f;
    }

    public void setPublisher(AID publisher) {
        this.publisher = publisher;
    }
}
```

```
    public AID getPublisher() {
        return this.publisher;
    }

    public void setSubscriber(AID subscriber) {
        this.subscriber = subscriber;
    }

    public AID getSubscriber() {
        return this.subscriber;
    }

    public void setLatency(float latency) {
        this.latency = latency;
    }

    public float getLatency() {
        return this.latency;
    }

    public void setReliability(float reliability) {
        this.reliability = reliability;
    }

    public float getReliability() {
        return this.reliability;
    }

    public void setReqInterval(float reqInterval) {
        this.reqInterval = reqInterval;
    }

    public float getReqInterval() {
        return this.reqInterval;
    }
}
```

## 8.48 DataWriter.java

```
package slash.util;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
```

```
import java.io.IOException;
import java.util.Hashtable;

public class DataWriter {

    private static String path = "./tmp/";
    private static String ext = "";

    private static Hashtable<String, Integer> counterTable = new
        Hashtable<String, Integer>();

    public static void writeData(String filename, float data) {
        File f=new File(path+filename+ext);

        if(f.exists()){
            try {
                FileWriter fstream = new FileWriter(path+
                    filename+ext, true);
                BufferedWriter out = new BufferedWriter(
                    fstream);
                Integer counter = counterTable.get(filename)
                    ;
                counter++;
                counterTable.put(filename, counter);
                out.append(String.valueOf(counter+"\t"+data+
                    "\n"));
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        else {
            try {
                FileWriter fstream = new FileWriter(path+
                    filename+ext);
                BufferedWriter out = new BufferedWriter(
                    fstream);
                Integer counter = 0;
                counterTable.put(filename, Integer.valueOf(
                    counter));
                out.write(String.valueOf(counter+"\t"+data+"
                    \n"));
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
    }

    public static void main(String[] args) {
        for(int i=0; i<50; i++) {
            writeData("test", 2.02f);
        }
    }
}
```

## 8.49 PropertiesReader.java

```
package slash.util;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;

public class PropertiesReader {
    private static Properties properties = null;
    private static File file = null;
    private static FileInputStream fis = null;
    private static String ret = null;

    public static String getProperty(String id) {
        try {
            if(id==" " || id==null)
                return null;
            if(file==null) {
                System.out.println(System.getProperty("user.dir"));
                file = new File("conf/config.properties");
                if(!file.exists())
                    file = new File(System.getProperty("user.dir")+"conf/conf.properties");

                fis = new FileInputStream(file);
            }
            if(properties==null) {
                properties = new Properties();
                properties.load(fis);
                fis.close();
            }
        }
    }
}
```

```
        if((ret=properties.getProperty(id))!=null)
            return ret;
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

public static Properties getProperties() {
    if(properties == null) {
        getProperty("indexPath");
    }
    return properties;
}
}
```

## 8.50 DsmServerBehaviour.java

```
package slash.dsm.behaviour;

import jade.core.AID;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import java.io.IOException;

import slash.dsm.agent.DsmServerAgent;
import slash.dsm.data.DsmDataManager;
import slash.dsm.tuple.Tuple;
import slash.entity.Context;
import slash.entity.Notify;

public class DsmServerBehaviour extends CyclicBehaviour {

    private static final long serialVersionUID = 6270841980860732385L;

    private DsmDataManager dsmDataManager;
    private int associatedID;

    public DsmServerBehaviour(DsmServerAgent agent) {
        this.dsmDataManager = new DsmDataManager();
        this.associatedID = 0;
    }
}
```

```
public void action() {
    try {
        MessageTemplate mt = MessageTemplate.
            MatchConversationId("dsm");
        ACLMessage rcvMsg = myAgent.receive(mt);
        Tuple tuple = null;
        if(rcvMsg!=null) {
            tuple = (Tuple)rcvMsg.getContentObject();
            doOperation(tuple, rcvMsg.getSender());
            tuple = dsmDataManager.read("notify", "
                notify");
            if(tuple!=null && tuple.getValue()!=null) {
                Notify notify = (Notify)tuple.
                    getValue();
                int dest = notify.getDest();
                if(dest!=this.associatedID) {
                    tuple = dsmDataManager.read(
                        String.valueOf(dest), "
                            context");
                    if(tuple!=null && tuple.
                        getValue()!=null) {
                        Context context = (
                            Context)tuple.
                                getValue();
                        myAgent.doMove(
                            context.
                                getLocation());
                        System.out.println("
                            Moving DSM to "+
                                dest);
                        this.associatedID =
                            dest;
                    }
                }
            }
        }
        else
            block();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private int doOperation(Tuple tuple, AID sender) {
    try {
```

```
        if(tuple!=null) {
            if(tuple.getOperation().equals(Tuple.IN)) {
                Tuple tupleRes = dsmDataManager.in(
                    tuple.getIndex(), tuple.getType()
                );
                ACLMessage msg = new ACLMessage(
                    ACLMessage.INFORM);
                msg.addReceiver(sender);
                msg.setLanguage("English");
                msg.setConversationId("dsm");
                msg.setContentObject(tupleRes);
                myAgent.send(msg);
            }
            else if(tuple.getOperation().equals(Tuple.
                READ)) {
                Tuple tupleRes = dsmDataManager.read(
                    tuple.getIndex(), tuple.getType()
                );
                ACLMessage msg = new ACLMessage(
                    ACLMessage.INFORM);
                msg.addReceiver(sender);
                msg.setLanguage("English");
                msg.setConversationId("dsm");
                msg.setContentObject(tupleRes);
                myAgent.send(msg);
            }
            else if(tuple.getOperation().equals(Tuple.
                OUT)) {
                dsmDataManager.out(tuple.getIndex(),
                    tuple.getType(), tuple.getValue()
                );
            }
            else if(tuple.getOperation().equals(Tuple.
                UPDATE)) {
                dsmDataManager.update(tuple.getIndex()
                    (), tuple.getType(), tuple.
                    getValue());
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    return 0;
}

}
```



## 8.51 DsmDataManager.java

```
package slash.dsm.data;

import java.io.Serializable;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.LinkedList;
import java.util.Queue;

import slash.dsm.tuple.Tuple;

public class DsmDataManager implements Serializable {

    Hashtable<String, Hashtable<String, Queue<Object>>>> tupleSpace;

    public DsmDataManager() {
        this.tupleSpace = new Hashtable<String, Hashtable<String,
            Queue<Object>>>>();
    }

    public synchronized int out(String index, String type, Object value)
    {
        if(index==null)
            System.out.println("ERRORE");
        else if(this.tupleSpace.containsKey(index)) {
            if(this.tupleSpace.get(index).containsKey(type)) {
                Queue<Object> queue = this.tupleSpace.get(
                    index).get(type);
                queue.offer(value);
            }
            else {
                Hashtable<String, Queue<Object>> oldType =
                    this.tupleSpace.get(index);
                Queue<Object> queue = new LinkedList<Object>();
                queue.offer(value);
                oldType.put(type, queue);
            }
        } //Se hashtable per index non esistente
        else {
            Hashtable<String, Queue<Object>> newType = new
                Hashtable<String, Queue<Object>>();
            Queue<Object> queue = new LinkedList<Object>();
            queue.offer(value);
            newType.put(type, queue);
        }
    }
}
```

```
        this.tupleSpace.put(index, newType);
    }

    return 0;
}

public synchronized int update(String index, String type, Object
value) {
    if(index==null)
        System.out.println("ERRORE");
    else if(this.tupleSpace.containsKey(index)) {
        if(this.tupleSpace.get(index).containsKey(type)) {
            Queue<Object> queue = this.tupleSpace.get(
                index).get(type);
            queue.clear();
            queue.offer(value);
        }
        else {
            Hashtable<String, Queue<Object>> oldType =
                this.tupleSpace.get(index);
            Queue<Object> queue = new LinkedList<Object>
                >();
            queue.offer(value);
            oldType.put(type, queue);
        }
    } //Se hashtable per index non esistente
    else {
        Hashtable<String, Queue<Object>> newType = new
            Hashtable<String, Queue<Object>>();
        Queue<Object> queue = new LinkedList<Object>();
        queue.offer(value);
        newType.put(type, queue);

        this.tupleSpace.put(index, newType);
    }

    return 0;
}

public synchronized Tuple in(String index, String type) {
    if(index==null) {
        Enumeration<Hashtable<String, Queue<Object>>>
            enumeration = this.tupleSpace.elements();
        Enumeration<String> keys = this.tupleSpace.keys();
        while(enumeration.hasMoreElements() && keys.
            hasMoreElements()) {
            String key = keys.nextElement();
```

```
        Hashtable<String, Queue<Object>> typeTable =
            enumeration.nextElement();
        Queue<Object> queue = typeTable.get(type);
        if(queue!=null) {
            Object obj = queue.poll();
            return new Tuple(Tuple.IN, type, key
                , obj);
        }
    }
}
else if(this.tupleSpace.containsKey(index)) {
    if(this.tupleSpace.get(index).containsKey(type)) {
        Queue<Object> queue = this.tupleSpace.get(
            index).get(type);
        Object obj = queue.poll();
        return new Tuple(Tuple.IN, type, index, obj)
            ;
    }
}
return null;
}

public synchronized Tuple read(String index, String type) {
    if(this.tupleSpace.containsKey(index)) {
        if(this.tupleSpace.get(index).containsKey(type)) {
            Queue<Object> queue = this.tupleSpace.get(
                index).get(type);
            Object obj = queue.peek();
            return new Tuple(Tuple.IN, type, index, obj)
                ;
        }
    }
    return null;
}
}
```

## 8.52 Tuple.java

```
package slash.dsm.tuple;

import java.io.Serializable;

/**
 * tuple: <operation, type, index, value>
```

```
* ex: <in, Context, node1, object>
*/
public class Tuple implements Serializable {

    private static final long serialVersionUID = -1484825309031979328L;

    public static String IN = "in";
    public static String READ = "read";
    public static String OUT = "out";
    public static String UPDATE = "update";

    private String operation;
    private String type;
    private String index;
    private Object value;

    public Tuple(String operation, String type, String index, Object
        value) {
        this.operation = operation;
        this.type = type;
        this.index = index;
        this.value = value;
    }

    public void setOperation(String operation) {
        this.operation = operation;
    }

    public String getOperation() {
        return this.operation;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getType() {
        return this.type;
    }

    public void setIndex(String index) {
        this.index = index;
    }

    public String getIndex() {
        return this.index;
    }
}
```

```
        public void setValue(Object value) {
            this.value = value;
        }

        public Object getValue() {
            return this.value;
        }
    }
}
```

## 8.53 DsmServerAgent.java

```
package slash.dsm.agent;

import jade.core.Agent;
import slash.dsm.behaviour.DsmServerBehaviour;

public class DsmServerAgent extends Agent {

    private static final long serialVersionUID = 4497729865124407532L;

    protected void setup() {
        this.addBehaviour(new DsmServerBehaviour(this));
    }
}
```

## 8.54 DsmClient.java

```
package slash.dsm.client;

import jade.core.AID;
import jade.core.Agent;
import jade.lang.acl.ACLMessage;

import java.io.IOException;
import java.io.Serializable;

import slash.dsm.tuple.Tuple;

public class DsmClient implements Serializable {

    private Agent agent;
```

```
private AID dsm;

public DsmClient(Agent agent) {
    this.agent = agent;
    dsm = new AID("dsm", AID.ISLOCALNAME);
}

public int out(String index, String type, Object value) {
    try {
        if(!index.matches("\\D*"))
            index = index.replaceAll("\\D*", "");
        Tuple tuple = new Tuple(Tuple.OUT, type, index,
                                value);

        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(dsm);
        msg.setLanguage("English");
        msg.setConversationId("dsm");
        msg.setContentObject(tuple);
        agent.send(msg);
    } catch (IOException e) {
        e.printStackTrace();
    }

    return 0;
}

public int update(String index, String type, Object value) {
    try {
        if(!index.matches("\\D*"))
            index = index.replaceAll("\\D*", "");
        Tuple tuple = new Tuple(Tuple.UPDATE, type, index,
                                value);

        ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
        msg.addReceiver(dsm);
        msg.setLanguage("English");
        msg.setConversationId("dsm");
        msg.setContentObject(tuple);
        agent.send(msg);
    } catch (IOException e) {
        e.printStackTrace();
    }

    return 0;
}
```

```
public Tuple read(String index, String type) {
    try {
        if(!index.matches("\\D*"))
            index = index.replaceAll("\\D*", "");

        Tuple tuple = new Tuple(Tuple.READ, type, index,
                                null);

        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver(dsm);
        msg.setLanguage("English");
        msg.setConversationId("dsm");
        msg.setContentObject(tuple);
        agent.send(msg);

        ACLMessage rcvMsg = agent.blockingReceive();
        if(rcvMsg!=null) {
            Object obj = rcvMsg.getContentObject();
            if(obj!=null) {
                return ((Tuple)obj);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return null;
}

public Tuple in(String index, String type) {
    try {
        if(!index.matches("\\D*"))
            index = index.replaceAll("\\D*", "");
        Tuple tuple = new Tuple(Tuple.IN, type, index, null)
            ;

        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver(dsm);
        msg.setLanguage("English");
        msg.setConversationId("dsm");
        msg.setContentObject(tuple);
        agent.send(msg);

        ACLMessage rcvMsg = agent.blockingReceive();
        if(rcvMsg!=null) {
            Object obj = rcvMsg.getContentObject();
            if(obj!=null) {
```

```
                return ((Tuple) obj);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return null;
}
}
```



# Listings

SLARequesterBehaviour.java . . . . .	52
SLAReqReceiverBehaviour.java . . . . .	53
ContextProducerBehaviour.java . . . . .	54
ViolationConsumerBehaviour.java . . . . .	55
RmConsumerBehaviour.java . . . . .	55
ContextManagerAgent.java . . . . .	57
CpuResourceBehaviour.java . . . . .	58
RamResourceBehaviour.java . . . . .	59
MemoryResourceBehaviour.java . . . . .	60
EnergyResourceBehaviour.java . . . . .	60
LatencyResourceBehaviour.java . . . . .	61
ReliabilityResourceBehaviour.java . . . . .	62
ReqIntervalResourceBehaviour.java . . . . .	63
RamRmAgent.java . . . . .	64
MemoryRmAgent.java . . . . .	64
EnergyRmAgent.java . . . . .	64
CpuRmAgent.java . . . . .	65
LatencyRmAgent.java . . . . .	65
ReliabilityRmAgent.java . . . . .	65
ReqIntervalRmAgent.java . . . . .	66
AgentLauncher.java . . . . .	66
SLAReceiverBehaviour.java . . . . .	68
SLACheckerBehaviour.java . . . . .	69

SLAStarterBehaviour.java . . . . .	72
ContextConsumerBehaviour.java . . . . .	73
MigrationUtil.java . . . . .	74
SLACheckerAgent.java . . . . .	75
ReqIntervalBehaviour.java . . . . .	76
NotifyReceiverBehaviour.java . . . . .	77
LatencyBehaviour.java . . . . .	78
RamBehaviour.java . . . . .	79
CpuBehaviour.java . . . . .	80
MemoryBehaviour.java . . . . .	81
EnergyBehaviour.java . . . . .	82
ReliabilityBehaviour.java . . . . .	84
CpuAgent.java . . . . .	85
ResourceAgent.java . . . . .	86
MemoryAgent.java . . . . .	86
EnergyAgent.java . . . . .	87
ReliabilityAgent.java . . . . .	88
ReqIntervalAgent.java . . . . .	89
LatencyAgent.java . . . . .	90
RamAgent.java . . . . .	91
DFUtil.java . . . . .	92
Context.java . . . . .	94
Notify.java . . . . .	99
SLAContract.java . . . . .	100
DataWriter.java . . . . .	101
PropertiesReader.java . . . . .	103
DsmServerBehaviour.java . . . . .	104
DsmDataManager.java . . . . .	107
Tuple.java . . . . .	109
DsmServerAgent.java . . . . .	111
DsmClient.java . . . . .	111

# Elenco delle figure

1.1	DSM . . . . .	4
1.2	architettura di JADE . . . . .	6
2.1	Componenti Nodo . . . . .	9
2.2	Esempio di rete con 3 nodi . . . . .	10
2.3	Esempio di scenario . . . . .	12
2.4	Esempio di scenario . . . . .	12
3.1	Esempio di IN nel Dsm . . . . .	16
5.1	Relazioni tra Package . . . . .	27
6.1	Migrazione non avvenuta . . . . .	30
6.2	Migrazione su secondo nodo . . . . .	32
6.3	Indice primo nodo . . . . .	34
6.4	Indice secondo nodo . . . . .	34
6.5	Indice terzo nodo . . . . .	35
6.6	Indice tutti i nodi . . . . .	35
6.7	Cpu primo nodo . . . . .	36
6.8	Cpu secondo nodo . . . . .	37
6.9	Cpu terzo nodo . . . . .	37
6.10	Cpu tutti i nodi . . . . .	38
6.11	Ram primo nodo . . . . .	39
6.12	Ram secondo nodo . . . . .	39
6.13	Ram terzo nodo . . . . .	40

## *ELENCO DELLE FIGURE*

---

6.14 Ram tutti i nodi . . . . .	40
6.15 Memory primo nodo . . . . .	41
6.16 Memory secondo nodo . . . . .	42
6.17 Memory terzo nodo . . . . .	42
6.18 Memory tutti i nodi . . . . .	43
6.19 Energy primo nodo . . . . .	44
6.20 Energy secondo nodo . . . . .	44
6.21 Energy terzo nodo . . . . .	45
6.22 Energy tutti i nodi . . . . .	45
6.23 Latency primo nodo . . . . .	46
6.24 Latency secondo nodo . . . . .	47
6.25 Latency terzo nodo . . . . .	47
6.26 Reliability primo nodo . . . . .	48
6.27 Reliability secondo nodo . . . . .	49
6.28 Reliability terzo nodo . . . . .	49

# Indice analitico

Agenti, 9  
AMS, 6  
AMS agent, 6  
  
Behaviour, 6  
  
checking, 14  
CM, 9  
Context, 13  
Context Manager, 9  
cpu, 20  
  
DF, 6  
DF agent, 6  
discovery, 5  
dsm, v, 12  
DsmData, 18  
  
framework, 17  
  
getBestNode(), 24, 25  
  
Hashtable, 16  
  
IN, 13, 14  
IN <index1, type3>, 16, 17  
in(), 4  
index1, 17  
intelligenza, 4  
  
introspezione, 5  
  
Jade, 10  
  
latency, 24  
  
Main Container, 6  
MainContainer, 28  
migrare, 10  
Migration checking, 24  
Mobile Computing, v  
Multi Agent Systems, 5  
  
OUT, 13, 14  
OUT <index1, type3, v1>, 17  
out(), 4  
  
platform, 5  
powerOn, 21  
proxy, 18  
pub, 24  
publisher, 14  
  
Queue, 16  
  
READ, 13  
read(), 4  
reliability, 24  
reqInt, 24

Resource Monitor, 9

SC, 9–12, 18

SLA, v

sla, 24

SLA Checker, 9, 10

SLAChecker, 11

sub, 24

subscriber, 14

tick, 21

tuning, 19

tuple space, 3, 4

type3, 17

UML, 27

UPDATE, 13

wired, 19

wireless, 8, 19

# Bibliografia

- [1] Carriero-Gelernter. Linda in context. 1989.
- [2] Eugster-Felber-Guerraoui-Kermarrec. The many faces of publish/subscribe. 2003.
- [3] Fuggetta-Pico-Vigna. Understanding code mobility. 1998.
- [4] Grassi. Slide del corso di informatica mobile. 2008.
- [5] Schiller. Mobile communications. 2003.