Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Sede di Modena

Corso di Laurea in Ingegneria Informatica – *Nuovo Ordinamento*

Studio della Sicurezza in una Piattaforma ad Agenti

un caso concreto: la piattaforma SEWASIE

Relatore:
Prof. Sonia Bergamaschi

Candidato:
Giacomo Benincasa

Anno Accademico 2004-2005

RINGRAZIAMENTI

Desidero ringraziare la prof. Sonia Bergamaschi e l'ing. Mirko Orsini per l'aiuto fornito durante la realizzazione di questo elaborato e per la disponibilità dimostrata e l'ing. Jonathan Gelati per le dritte fornitemi all'inizio del lavoro.

Ringrazio inoltre la mia famiglia per il sostegno durante il percorso di studio.

PAROLE CHIAVE
agenti software
sistemi multi-agente
sicurezza
jade
jaas

Indice

Introduzione

Lo stato dell'arte della ricerca sui sistemi ad agenti mostra l'utilità del paradigma nella progettazione e implementazione di servizi distribuiti in un ampio spettro di domini applicativi. Alcuni tra i principali domini, quali il commercio elettronico e la ricerca d'informazioni distribuite, sono molto sensibili alle problematiche di sicurezza. Nelle applicazioni ad agenti mobili, infatti, interagiscono una molteplicità di entità che eseguono per conto di persone od organizzazioni diverse. In particolare, un agente può spostarsi su nodi al di fuori dell'amministrazione del proprio datore. Garantire la sicurezza significa allora garantire, da una parte, che gli agenti non violino in alcun modo i nodi che visitano e, dall'altra, che i nodi eseguano gli agenti correttamente, senza violarne l'integrità o la privacy.

Il progetto SEWASIE (SEmantic Webs and AgentS in Integrated Economies), progetto guidato dall'Università di Modena e Reggio Emilia, si poneva l'obiettivo di sviluppare un motore di ricerca intelligente in grado di integrare ed arricchire informazioni raccolte da sorgenti di dati eterogenee. L'integrazione delle sorgenti viene fatta sfruttando il sistema sviluppato nell' ambito del progetto MOMIS (Mediator EnvirOnment for Multiple Information Sources), al quale hanno partecipato il Dipartimento di Scienze dell'Ingegneria di Modena e il Dipartimento di Scienze dell'Informazione di Milano.

Per la realizzazione del motore che rende possibile la raccolta e l'integrazione delle informazioni, è stata adottata un'architettura basata su agenti software.

La piattaforma ad agenti realizzata nell'ambito di questo progetto era interessata da un problema di particolare rilevanza: l'impossibilità di effettuare il *join* tra i container periferici ed il main-container della piattaforma, ospitati su macchine differenti, in presenza di firewall. A causa di questo problema venne fatta la scelta di eseguire l'intera piattaforma in un'unica macchina, venendo così meno la distribuzione della piattaforma.

La tesi si pone l'obiettivo di studiare metodologie che consentano di risolvere il suddetto problema. Si vuole inoltre analizzare il problema della sicurezza in un sistema ad agenti, in particolare verranno studiate metodologie che consentano la realizzazione di una piattaforma ad agenti distribuita e sicura, facendo riferimento alla piattaforma ad agenti del progetto SEWASIE.

Il testo è suddiviso in due parti principali: la prima contenente i capitoli di base necessari per la comprensione del resto dell'elaborato mentre la seconda descrive il lavoro effettuato nell'ambito di questa tesi. Nel primo capitolo (Agenti, Ambienti e Sistemi Multi-Agente) verranno introdotte le nozioni fondamentali riguardo il paradigma ad agenti, oltre che gli standard architetturali fondamentali caratterizzanti un sistema di agenti previsti dalla FIPA (Foundation for Intelligent Physical Agent). Infine verranno descritte brevemente le caratteristiche del progetto JADE. Il secondo capitolo (La Piattaforma SEWASIE) tratta in breve il funzionamento della piattaforma ad agenti del progetto SEWASIE. In particolare verranno esaminate le diverse tipologie di agenti e le loro interazioni. Nel terzo capitolo (Considerazioni sulla Sicurezza in Sistemi Multiagente) vengono descritte le problematiche di sicurezza che interessano i sistemi distribuiti, con particolare riferimento ai sistemi multi-agente. Nel quarto capitolo (Gestione della Sicurezza in Jade) vengono analizzati i meccanismi di sicurezza implementati nel pacchetto JADE-S oltre alle limitazioni e problemi che questi portano. Il capitolo quinto (Configurazione della Piattaforma) offre invece una descrizione delle modifiche alla configurazone della piattaforma ad agenti del progetto SEWASIE in modo che questa supporti i permessi estesi di JADE e in modo da integrare le funzionalità del pacchetto base di JADE con JADE-LEAP. Nel sesto ed ultimo capitolo vengono presentate riflessioni sul lavoro svolto, sul suo possibile proseguimento ed alcune considerazioni sullo stato dell'arte attuale della gestione della sicurezza in sistemi distribuiti multiagente.

Parte I

Capitolo 1

Agenti, Ambienti e Sistemi Multi-Agente

In questo capitolo verranno introdotte le nozioni fondamentali riguardo il paradigma ad agenti, oltre che gli standard (FIPA) architetturali fondamentali caratterizzanti un sistema di agenti.

1.1 Gli Ambienti

Per ambiente si intende *una rete comprendendente svariati host, geograficamente distribuita ma logicamente accessibile.* Gli ambienti del sistema reale possiedono diverse proprietà da cui dipende la complessità della realizzazione dell'infrastruttura software che li descrive. Per *infrastruttura* si intende quell'entità in grado di garantire agli agenti una visione reciproca, iterazioni con la piattaforma e altri agenti, lo scambio di informazioni e, opzionalmente, la mobilità su un'altro host. Tali proprietà possono essere identificate nell'*accessibilità,* nel grado di controllabilità da parte di un agente e nel *tipo di evoluzione*. Entrando più nel dettaglio, Russel e Norving rintracciano negli ambienti le seguenti caratteristiche:

accessibilità / inaccessibilità: un ambiente è accessibile quando gli agenti possono ottenere informazioni sul medesimo, complete, accurate e aggiornate. Più un ambiente è accessibile più l'implementazione del sistema di agenti sarà semplice.

determinabilità / indeterminabilità: un sitema è deterministico se ogni azione provoca un cambio di stato univoco.

episodietà / non-episodietà: un ambiente è episodico se l'insieme degli stati è discreto.

staticità / dinamicità: si ha un ambiente statico quando lo stato non cambia se non in risposta all'azione di un agente.

discretezza / continuità: un ambiente è discreto quando esiste solo un numero finito di azioni consentite.

1.2 Gli Agenti

Esistono varie definizioni di agente, quella forse più completa venne enunciata da Russel e Norving nel 1995, i quali definiscono gli agenti come: *entità capaci di percepire ed agire in maniera razionale, ovvero di eseguire compiti in modo giusto puntando alla massimizzazione della propria misura di prestazione, in base alle esperienze fornite da qualsiasi conoscenza predefinita o acquisita*.

La definizione enunciata da Russel e Norving comprende tutte le principali proprietà che determinano un agente, ovvero:

Autonomia: ossia la capacità di effettuare operazioni senza direttive esterne e la capacità di intervenire sul proprio stato, tutto ciò senza l'intervento di un'entità sovra ordinata.

Reattività: ovvero la capacità di reagire alle modifiche dell'ambiente che li circonda, ogni qual volta tali cambiamenti influenzino il loro obiettivo.

Capacità di comunicare: capacità di comunicare con il mondo esterno, interagire con un utente o con altri agenti.

Abilità Sociale: ossia la capacità di interagire con gli altri agenti, che possono anche essere persone, sfruttando un linguaggio di comunicazione ed, eventualmente, cooperando.

Altre caratteristiche di rilevante importanza sono:

Proattività: capacità di generare eventi nell'ambiente circostante, iniziare delle interazioni con altri agenti, coordinare le attività di differenti agenti stimolandoli a produrre certe risposte.

Adattamento: gli agenti devono essere in grado di migliorare le loro performance nel tempo.

Mobilità: la capacità di muoversi all'interno di una rete.

Razionalità: l'agente agisce allo scopo di conseguire un obiettivo.

Nozioni mentali: capacità di possedere conoscenze, ricordare esperienze, avere visioni proprie dell'ambiente che li circonda e sugli altri gruppi con cui collabora, a seconda del livello delle sue abilità sociali; inoltre l'agente deve essere in grado di apprendere dalle esperienze, interagendo con l'ambiente e con gli ù altri tramite rapporti sociali.

Benevolenza: un agente fa sempre ciò che gli si comanda.

Veridicità: un agente non deve comunicare false informazioni.

Persistenza: capacità di permanere nel tempo, ciò significa che la durata della vita di un agente è superiore alla durata dei compiti che esegue di base. Un agente continua ad esistere con uno stato interno, in modo da poter eseguire interazioni successive, ma non è detto che la caratteristica di persistenza possa essere mantenuta in maniera indefinita. Tipicamente un agente sarà modellato definendo un compito durante il quale può compiere diverse interazioni finite e delle risorse interne che possono essere riprodotte o consumate.

Vitalità: è la capacità di sopravvivenza di un agente, nel senso che, l'agente può riuscire a risolvere situazioni anomale che lo porterebbero, altrimenti, in uno stato d'instabilità che comprometterebbe la sua persistenza.

1.3 Sistemi Multi-Agente

Un sistema multi-agente (Multi-Agent Systems, MAS) è un sistema in cui gli agenti intelligenti interagiscono e coesistono tra di loro per soddisfare un certo insieme di obiettivi, allo scopo di portare a termine un certo insieme di compiti.

Molte delle proprietà precedentemente elencate relative agli ambienti, derivano dal fatto che un agente deve essere in grado di interagire con questi.

I MAS hanno ormai acquisito un'importanza strategica nei campi dell'ingegneria del software, delle discipline scientifiche e dell'innovazione tecnologica. I sistemi del mondo reale sono spesso caratterizzati da un alto grado di complessità dunque un solo agente spesso non è in grado di adempiere il proprio compito a causa della limitatezza delle risorse che questo possiede. Anche in fase di progettazione, disegnare le iterazioni tra più agenti semplici è preferibile rispetto al progetto di un unico agente complesso; inoltre, la possibilità di eseguire concorrentemente e quindi parallelizzare i task, spesso porta notevoli benefici per quel che riguarda le performance del sistema. L'importanza dei MAS risiede dunque nelle loro capacità di risolvere problemi in sistemi altamente distribuiti che sono troppo ampi affinché un unico agente intelligente possa normalmente risolverli. Soluzioni che usano efficientemente sorgenti di informazione distribuite nello spazio, come la raccolta informazioni su Internet possono essere trovate sfruttando le caratteristiche dei MAS. Tali caratteristiche riportate in [14] sono:

Efficienza computazionale: deriva dal carattere altamente distribuito di questi sistemi.

Robustezza: la capacità di sostenere l'imprevedibilità intrinseca di ambienti aperti e complessi nei quali gli agenti interagiscono scambiandosi informazioni.

Affidabilità: la capacità di sopperire ad errori o problemi nei componenti.

Estensibilità: deriva dal fatto che le capacità di un agente operante su un problema possono essere modificate.

Manutenibiltà: deriva dal fatto che un sistema di agenti è modulare.

Reattività: dovuta al fatto che, sempre grazie alla modularità, gli agenti sono in grado di trattare anomalie in modo locale, senza propagarle all'intero sistema.

Flessibilità: è dovuta alla capacità degli agenti di avere differenti possibilità di organizzarsi in maniera attiva per risolvere un problema.

Riusabilità: agenti con determinate funzionalità, possono offrire i propri servizi ad altri gruppi di agenti per la risoluzione di diversi problemi.

Sia un MAS, che un singolo agente, sono caratterizzati da una propria infrastruttura. L'*infrastruttura* di un MAS può essere definita come un insieme di servizi, di convenzioni e di conoscenze, che supportano complesse interazioni fra gli agenti [15]. La complessità delle iterazioni è dovuta principalmente alla necessità degli agenti di comunicare tra di loro in ambienti eterogenei e dinamici (basti pensare Internet) in modo sicuro. Dal punto di vista dell'infrastruttura del MAS, gli agenti non sono altro che entità autonome e socialmente consapevoli che interagiscono tra di loro, oltre che con i componenti dell'infrastruttura, utilizzando comportamenti standardizzati dalle regole del MAS. In altre parole il MAS sa *cosa* l'agente è in grado di fare, ma non sa *come* lo fa, le metodologie di risoluzione dei problemi da parte di un agente risultano all'infrastruttura come una scatola nera. Negli ultimi anni, l'interesse verso i MAS è notevolmente aumentato, ciò ha spinto società, prevalentemente di telecomunicazioni e informatica, ma anche comunità di ricercatori e programmatori, ad orientare i loro studi verso la standardizzazione della tecnologia multi-agente.

1.4 FIPA

L'organizzazione FIPA (Foundation for Intelligent Physical Agent), della quale fanno parte una quarantina di grandi compagnie, prevalentemente appartenenti al campo dell'*Information Technology*, ha pubblicato una serie di specifiche aperte allo scopo di garantire la massima compatibilità tra piattaforme ad agenti realizzati con diverse tecnologie. Queste specifiche possono essere classificate, come risulta in [8], in:

Component specifications: descrivono la soluzione a problemi tecnici.

Informative specifications: sono una sorta di guida per l'industria che si vuole avvicinare a queste tecnologie.

In particolare vengono forniti modelli in materia di:

- creazione,
- registrazione,
- ricerca,
- comunicazione,
- migrazione ed
- eliminazione

degli agenti mobili.

1.4.1 Specifiche FIPA

Secondo le specifiche della FIPA, un sistema basato su agenti mobili deve possedere le seguenti caratteristiche:

Agent Platform (AP): è l'infrastruttura fisica nella quale possono essere eseguiti gli agenti. Gli agenti infatti possono venire eseguiti su una singola macchina, ma, soprattutto, possono essere eseguiti su una rete. In quest'ultimo caso la piattaforma è rappresentata da tutti i nodi della rete.

Message Transport Service (MTS): è il modello di comunicazione utilizzato dagli agenti per comunicare.

Agent: intesi come processi software che possiedono un proprio ciclo di vita che viene spesso gestito dalla piattaforma. Le specifiche FIPA delineano quali sono gli stati in cui può trovarsi un agente durante il suo ciclo di vita. All'inizio del ciclo vitale l'agente si trova nello stato **initiated** al quale segue lo stato **active**. Dallo stato active l'agente può passare allo stato **transit**

(ad esempio durante un'operazione di spostamento da un nodo della rete ad un altro), oppure **waiting**, o ancora **suspended**. L'agente può continuare ad assumere i seguenti stati fino alla conclusione del suo stato vitale.

Directory Facilitator (DF): è un agente che viene avviato automaticamente dalla piattaforma. Il suo scopo è quello di interagire con gli altri agenti fornendo loro informazioni sulla posizione degli altri agenti all'interno della piattaforma e sui servizi che questi offrono.

Agent Management System (AMS): è la massima autorità del sistema di agenti e contiene gli identificativi (**Agent IDentifier (AID):**) degli agenti registrati sulla piattaforma.

Per garantire le compatibilità fra i sistemi è di fondamentale rilevanza porre precise specifiche per quel che riguarda la comunicazione tra gli agenti. La comunicazione tra gli agenti avviene tramite lo scambio di messaggi codificati in ACL (Agent Communication Language). Il protocollo per lo scambio di messaggi ACL, secondo le specifiche FIPA, è composto da cinque livelli:

Protocol: definisce le regole necessarie alla formulazione del dialogo fra gli agenti

Communicative Act: indica lo scopo che l'agente vuole raggiungere con l'atto comunicativo

Messaging: definisce le pseudo-informazioni contenute nel messaggio

Content language: indica la sintassi utilizzata per esprimere il contenuto del messaggio.

Ontology: definisce il vocabolario delle espressioni utilizzate nella comunicazione.

Perchè lo scambio di messaggi possa venire effettuato è sufficiente che un agente sia a conoscenza del nome dell'agente a cui vuole spedire il messaggio.

1.5 Considerazioni sul paradigma ad agenti

Il paridigma ad agenti si basa su alcuni concetti di intelligenza artificiale e teoria del linguaggio su sistemi distribuiti. E' importante osservare l'intrinseca architettura *peer-to-peer* dei sistemi di agenti: ogni agente è un peer che può iniziare una comunicazione potenzialmente con ogni altro agente. Il meccanismo che permette ad un agente di venire in contatto con altri può essere implementato seguendo principalmente due modelli:

peer-to-peer *puro*

e peer-to-peer *ibrido*

Il primo è un modello completamente decentralizzato e i peer (agenti) sono completamente autonomi. Sebbene questo modello offra interessanti prospettive, la complessità dei meccanismi per il mantenimento della coerenza della rete e per la scoperta dei peer è molto elevata, inoltre questi meccanismi prevedono un costante ed elevato scambio di informazioni che porta il rapido esaurirsi della banda a disposizione quando il numero dei peer aumenta. Un ultima importante considerazione riguarda la sicurezza di sistemi di questo tipo dove, mancano sistemi di controllo e dove ogni peer è libero di introdursi o disconnettersi da una rete. Le architetture ibride prevedono invece la presenza di un nodo speciale che offre principalmente il servizio di *pagine gialle*, ma che rende più semplice anche la gestione di meccanismi per il controllo ed il monitoraggio dell'attività degli agenti.

1.6 JADE

Al giorno d'oggi esistono numerosi progetti per la creazione di piattaforme di agenti, tutti con diversi propositi e con differenti gradi di sviluppo. La maggior parte di questi seguono le direttive FIPA. Allo scopo di scegliere la piattaforma da utilizzare per il progetto SEWASIE ne sono state prese in considerazione diverse, le quali sono state valutate sulla base di:

- supporto delle specifiche FIPA.

- Caratteristiche degli agenti: ad esempio la comunicazione intra-piattaforma, mobilità, persistenza e sicurezza.
- Disponibilità: la licenza sotto cui il prodotto viene rilasciato.
- Documentazione e
- applicazioni che sfruttano la piattaforma.

La piattaforma che ha soddisfetto al meglio il progetto è JADE.

1.7 Peculiarità piattaforma JADE

Come già detto JADE [6](Java Agent DEvelopment Framework) è un framework interamente scritto in java, sviluppato dal CSELT (Centro Studi E Laboratori Telecomunicazioni) congiuntamente all'Università di Parma, che funge da middleware per lo sviluppo di sistemi distribuiti multi-agente con struttura peer-to-peer, è conforme alle direttive FIPA e comprende tool per lo sviluppo, il debugging, la gestione e il monitoraggio del sistema. La struttura peer-to-peer è dovuta al fatto che un sistema basato sugli agenti è intrinsecamente peer-to-peer: ogni agente è infatti un nodo in grado di instaurare una comunicazione con ogni altro agente. Le API messe a disposizione da JADE sono indipendenti dalla versione di JAVA. Il meccanismo di trasporto è implementato usando RMI e IIOP. Sono inoltre presenti numerosi add-ons, in particolare JADE-S e JADE-LEAP, che integrano le funzionalità di JADE introducendo rispettivamente gestione della sicurezza ed una maggiore compatibilità con gli ambienti mobili. JADE offre un approccio flessibile ed efficiente per la comunicazione tra gli agenti.

1.8 Funzionamento di JADE

Il primo container che viene lanciato funziona obbligatoriamente come main-container, dunque saranno automaticamente lanciati anche gli agenti AMS e DF. Assieme al main-container viene lanciato anche l'*RMI registry* al quale verranno

aggiunti tutti gli altri container periferici che verranno aggregati per creare la piattaforma. E' importante sottolineare che ogni container lavora all'interno di una propria JVM (Java Virtual Machine).

Per lanciare il main-container si usa il comando:

```
java jade.Boot [options] [AgentSpecifier list]
```

Per lanciare un container periferico invece:

```
java jade.Boot -container [options] [AgentSpecifier list]
```

Le opzioni di maggiore interesse sono:

-container l'istanza di JADE che verrà creata è un container periferico che deve essere collegato ad un container principale.

-host specifica l'host name della macchina sulla quale è attivo il main-container presso cui il container periferico deve essere registrato.

-port specifica il numero di porta sulla quale lavora il main-container.

-local-host permette di specificare l'host name della macchina su cui si sta lanciando il container.

-local-port permette di specificare il numero di porta sulla quale il container potrà essere contattato.

-name permette di specificare *il nome simbolico con cui identificare la piattaforma*. Può essere usato quando si sta lanciando un main-container.

-container-name permette di specificare *il nome simbolico con cui identificare il container*.

-services consente di specificare i servizi dei quali si deve avvalere la piattaforma. I servizi desiderati devono essere attivati durante la fase di avvio.

-nomobility se viene specificata questa opzione verranno disabilitate le possibilità di clonazione e migrazione degli agenti.
Non si può utilizzare questa opzione quando si usa l'add-on JADE-LEAP.

-conf consente di specificare un file di configurazione contenente tutte le impostazioni di configurazione desiderate. Se non viene specificato il nome del file verrà avviato un'interfaccia grafica per la creazione di questo.

-gui se presente questa opzione verrà avviata RMA (Remote Monitoring Agent), una semplice interfaccia grafica che consente di monitorare la presenza e le azioni degli agenti sulla piattaforma. Da questa interfaccia sono inoltre avviabili gli altri tool per la gestione e per il monitoraggio della piattaforma.

La lista presetata non è esaustiva, ad esempio mancano le interessanti opzioni per la creazione e gestione delle repliche del main-container, utili per la creazione di piattaforme tolleranti ai guasti. Maggiori informazioni a riguardo possono essere reperite in [7].

Da linea di comando o da file di configurazione può essere introdotta una lista di agenti operanti sul container. Per poter avviare un agente bisogna specificare il nome della classe Java che implemente l'agente, il nome dell'agente e eventuali argomenti che si vogliono passare all'agente. Esemplificando:

```
Peter:myPackage.myAgent("My name is Peter", 1234)
```

Con questo esempio verrà creato sul container un agente di nome Peter, istanza di un oggetto della classe `myAgent` , appartenente al package myPackage ed al costruttore della classe verrà passato un array contenente i due valori My name is Peter e 1234.

1.8.1 JADE-S

JADE-S [1] è un pacchetto aggiuntivo che si appoggia su JADE e ne amplia le funzionalità introducendo la gestione della sicurezza; in particolare JADE-S permette la creazione di piattaforme multiutente gestendone l'autenticazione degli utenti e i permessi che questi hanno. JADE-S permette inoltre lo scambio di messaggi cifrati tra gli agenti appartenenti alla stessa piattaforma.

[1] qui si farà riferimento alla versione 2

1.8.2 JADE LEAP

La maggior parte dei dispositivi mobili, come palmari o cellulari, non supportano il Java Devolopment Kit (richiesto da JADE), ma ambienti più leggeri (come PersonalJava o MIDP), creati apposta per questo tipo di dispositivi, inoltre le differenti caratteristiche delle reti wireless rispetto alle reti fisse (alta latenza, banda minore, connettività intermittente...) e l'elevato consumo di memoria da parte di JADE, ne rendono impossibile l'utilizzo su questo tipo di ambienti. Per queste ragioni è stato creato JADE LEAP, un pacchetto aggiuntivo che sostituisce alcune classi del *core* di JADE in modo che la piattaforma possa funzionare anche su questo tipo di dispositivi. Per maggiori informazioni a riguardo si veda il paragrafo 4.2 del capitolo 4.

Capitolo 2

La Piattaforma SEWASIE

SEWASIE (Semantic Webs and AgentS in Integrated Economies) è un progetto di ricerca europeo (numero di contratto 34825, inerente alla chiamata IST-01-7-1A), della durata di 36 mesi, iniziato nel maggio 2002. Il progetto si pone come obiettivo l'implementazione di un motore di ricerca avanzato in grado di compiere ricerche intelligenti sul web sfruttando metodologie di indagine semantica. Sfruttando un'architettura in grado di organizzare le informazioni in base al loro valore semantico, il motore di ricerca SEWASIE sarà in grado di rintracciare i dati maggiormente significativi in base alle preferenze dell'utente che effettua la ricerca. SEWASIE integra inoltre meccanismi di ricerca e negoziazione ai quali si può accedere tramite l'utilizzo di opportune interfacce. Il sistema di ricerca viene effettuato da un complesso sistema distribuito di agenti i quali cooperano al fine del recupero delle informazioni. Il sito ufficiale del progetto è consultabile all'indirizzo riportato in [20].

Per meglio comprendere ciò di cui si parlerà nel resto del paragrafo, viene data ora una definizione generale di *ontologia*. Per ontologia in campo informatico si intende il *tentativo di formulare uno schema concettuale esaustivo e rigoroso nell'ambito di un dato dominio*; si tratta generalmente di una struttura dati gerarchica che contiene tutte le entità rilevanti, le relazioni esistenti fra di esse, le regole, gli assiomi, ed i vincoli specifici del dominio[25].

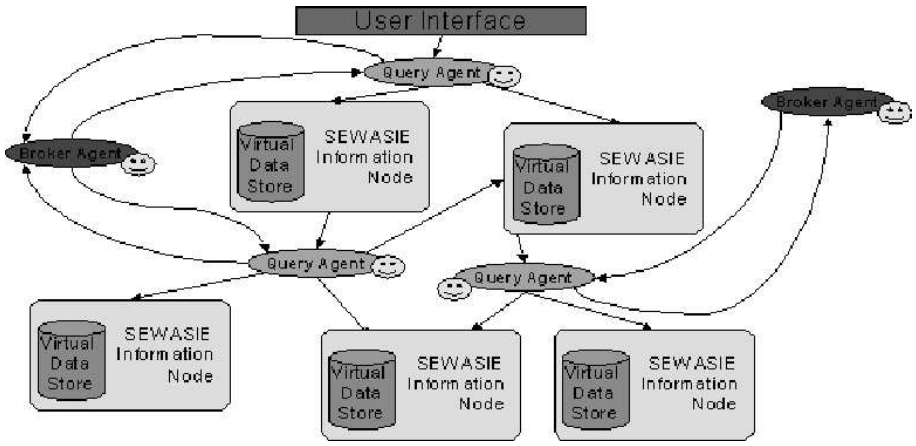

Figura 2.1: Architettura ad alto livello di SEWASIE

2.1 Struttura della piattaforma ad agenti del progetto SEWASIE

L'architettura di SEWASIE prevede che ogni agente appartenga ad uno specifico *tipo* il quale determina le azioni che l'agente può compiere[19]. I principali servizi sono:

- gestione delle strutture dati,
- scambio di messaggi con altri agenti,
- richiesta di servizi.

Le azioni che un tipo di agente può eseguire sono descritte, in linguaggio dichiarativo, in apposite *action policy*.

Ci sono quattro agenti base in SEWASIE: *Query Agent (QA), Brokering Agent (BA), Monitoring Agent (MA) e Communication Agent (CA).* In realtà anche il *SEWASIE Information Node (SINode)* dovrebbe essere considerato un agente, questo infatti collabora con agenti di altri tipi, in particolare SINode dipende da BA e fornisce servizi di rilevante importanza a QA. Anche l'interfaccia utente intergisce con gli agenti; il *query tool user interface* si occupa dell'inizializzazione di molte interazioni tra gli agenti ma non fornisce servizi. Il query tool istanzia,

in base alle richieste dell'utente, opportuni QA e poi, quando riceve i risultati dal QA crea, a seconda dell'esigenza, un MA o un CA. Il query tool interagisce anche con il BA per la creazione della query. Il query tool ricopre dunque un ruolo importante nell'architettura di SEWASIE, ma, dal punto di vista del SAE (SEWASIE Agent Environment), questo ricopre solo il ruolo di client e non di peer. Nel resto della trattazione il query tool verrà comunque descritto come un agente.

SINode

I SINode sono nodi virtuali (possono essere distribuiti su più computer) che gestiscono le sorgenti di informazioni e ne rendono disponibile una vista virtuale. Nel sistema possono essere presenti numerosi SINode ognuno dei quali può gestire diverse sorgenti. Gli SINode vengono usati per l'estrazione dei dati dalle sorgenti. L'*Ontology Builder* è l'entità (semi-automatica) che si occupa della creazione di una vista virtuale globale (Global Virtual View) sugli schemi delle sorgenti. L'importanza di questo elemento risiede nel fatto che le query elaborarate dai query agent del SINode sono costruite sulla vista globale fornita dell'Ontology Builder.

Brokering Agent

Si occupa di integrare le differenti GVV provenienti dai diversi SINode in una Brokering Agent Ontology (BA Ontology). Gli utenti formulano le loro query sulla base di questa ontologia che consente di guidare i Query Agent verso gli SINode che contengono le informazioni ricercate. Sulla rete di SEWASIE possono essere presenti più Brokering Agent ognuno dei quali integra le GVV provenienti da uno specifico dominio.

Query Agent

Il Query Agent riceve la query, espressa tramite un'ontologia, direttamente dall'interfaccia utente, quindi la traduce opportunamente secondo la GVV del BA e degli SINodes che andrà ad interrogare e infine sottomettere la query ai SINode. Il Query Agent si occupa anche della fusione dei risultati ottenuti dai differenti SINode.

Monitoring Agents

I monitoring Agent vengono utilizzati per memorizzare permanentemente i risultati di alcune query; operando in questo modo, ripetendo la stessa interrogazione potranno essere confrontati i dati ottenuti. Ogni monitor agent possiede una ontologia interna fissa, chiamata *domain-model* .

Communication Tool

Il Communication Tool utilizza i risultati ottenuti dalle interrogazioni, le ontologie dei differenti BA, oltre che specifiche ontologie di negoziazione, per la negoziazione di un contratto. Il Communication Tool può richiedere l'intervento di altri agenti che lo supportino durante i processi di decisione.

Capitolo 3

Considerazioni sulla Sicurezza in Sistemi Multiagente

Le applicazioni del mondo reale, specialmente quelle che lavorano in reti aperte, come Internet, devono essere progettate tenendo conto di problematiche di sicurezza. In particolare i sistemi distribuiti multiagente, a causa dell'autonomia e della mobilità di cui godono, richiedono un alto grado attenzione verso queste problematiche.

In questo capitolo, dopo una trattazione sommaria dei possibili rischi che coinvolgono i sistemi distribuiti, si entrerà più nel dettaglio esaminando come queste problematiche possano influenzare il comportamento dei MAS. Il fatto che gli esempi a seguire trattino unicamente di sistemi ad agenti, non significa affatto che le tipologie di attacco e di protezione qui presentate valgano unicamente per questo tipo di ambienti. Questo capitolo cerca di mostrare come le problematiche di sicurezza che coinvolgono i generici sistemi distribuiti interessino anche i sistemi ad agenti e come le peculiarità di questo tipo di sistemi aggiunga nuovi tipi di problematiche. Nel corso del capitolo verrano presi in considerazione sia MAS estesi su di un'unica piattaforma sia casi più complessi di MAS estesi su piattaforme differenti.

3.1 Minacce

Le minacce alla sicurezza possono essere divise in tre categorie principali:

- rilascio di informazioni non autorizzato,
- negazione di servizio (Denial Of Service) e
- corruzione delle informazioni.

Nel seguito le metodologie con cui le minacce di cui sopra possono essere attuate verranno classificate a seconda del componente che attua l'attacco.

I componenti di un sistema ad agenti che possono portare attacchi sono gli agenti e i container. Possono dunque essere identificate quattro categorie di minacce:

- attacco da parte di un agente verso la piattaforma
- attacco di un piattaforma verso un agente
- un agente attacca un altro agente all'interno della piattaforma
- un'entità esterna attacca il sistema ad agenti.

Una piattaforma ad agenti sicura dovrebbe proteggere tutte le interfacce esposte all'azione degli agenti e dei container, ciò significa proteggere sia il livello di infrastruttura che il livello agente, in modo che le azioni eseguite rispettino adeguate misure di sicurezza. Nel seguito vengono analizzate diverse metodologie di attacco ad un sistema di agenti.

3.1.1 Mascheramento

Il mascheramento si ha quando un agente, un container o un'entità esterna. si spacciano per un altro componente dello stesso tipo allo scopo di acquisire permessi o la fiducia di un'altra entità. Nel caso l'attacco abbia esito positivo, l'entità mascherata otterrà permessi per l'accesso a risorse alle quali non sarebbe autorizzato ad accedere, inoltre la colpa delle sue azioni ricadrebbe sopra l'agente di cui ha preso l'identità provocando, ad esempio, il danneggiamento della fiducia di

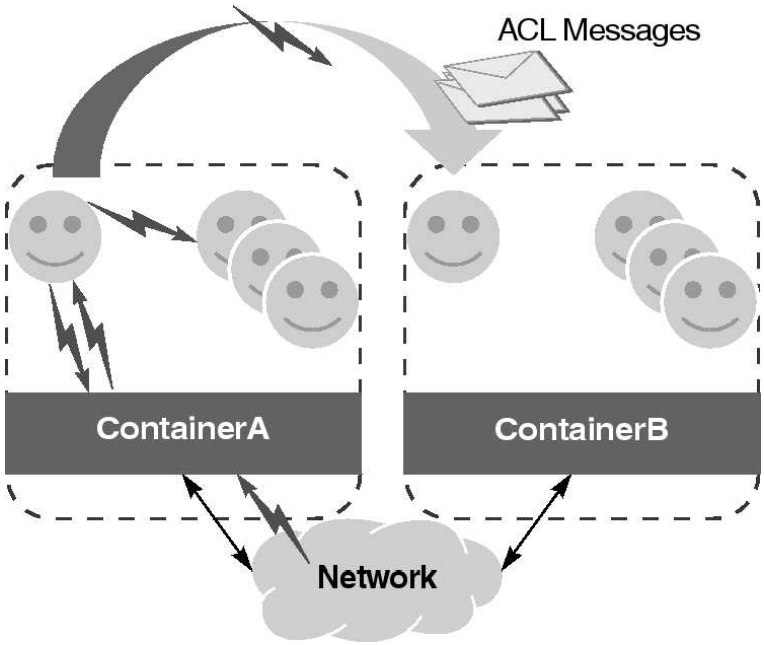

Figura 3.1: punti critici per la sicurezza

tutta la comunità di agenti della piattaforma. Non solo gli agenti possono essere interessati al mascheraramento della propria identità, ma anche una piattaforma può cercare di sembrare un'altra allo scopo di convincere un agente mobile di essere arrivato a destinazione e quindi ad un dominio sicuro.

3.1.2 Negazione di servizio

E' dovuto all'eccessivo consumarsi delle risorse computazionali della piataforma in seguito ad un'azione richiesta da un entità interna (agente o container) o esterna. Questo tipo di attacco può essere lanciato intenzionalemente, ad esempio da un agente, sfruttando una vulnerabilità del container, ma può anche essere dovuto ad errori di programmazione. Sfruttando le metodologie dell'ingegneria del software le vulnerabilità del container ed errori di programmazione si dovrebbero ridurre di numero. Oltre al caso di un attacco DoS verso un container, un agente può cercare di ottenere un DoS di un altro agente. I mezzi con cui cercare di ottenere questo sono numerosi, basti pensare ad un spamming-agent che inoltra un gran

numero di messaggi ad un altro agente. Il linguaggio di comunicazione degli agenti (ACL) e le policy che permettono o meno la comunicazione fra due agenti dovrebbero rendere questo genere di attacchi difficoltoso. Anche un container può provocare un DoS di un agente; quando un agente arriva in una piattaforma si aspetta che questa soddisfi le su richieste fedelmente, per cui una piattaforma maliziosa potrebbe deliberatemente ignorare le richieste dell'agente ad esempio introducendo un ritardo inaccettabile per l'attuazione di un compito di rilevante importanza (ad esempio l'acquisto di azioni), oppure, semplicemente non eseguire l'azione richiesta e terminare l'agente senza notifica.

3.1.3 Accesso non autorizzato

E' quel genere di attacco atto ad avere l'accesso ad informazioni o risorse di cui non si ha l'autorizzazione ad eseguire o conoscere. Può essere eseguito da un agente o da un'entità esterna nei confronti di un'altro agente o di un container. Se la piattaforma non prevede meccanismi di controllo a riguardo, è possibile che un agente attaccante cerchi di invocare direttamente i metodi pubblici di un'altro agente (ad esempio allo scopo di provocare un buffer overflow) o provare ad accedere ai dati e al codice dell'agente per modificarli.

3.1.4 Rifiuto

Si ha quando un agente che partecipa ad una transazione o ad una comunicazione chiede che questa sia interrotta. Il rifiuto può essere accidentale o meno, in ogni caso può portare a dispute di difficile risoluzione e alla perdita di dati. La piattaforma non è in grado di prevenire il rifiuto, ma può prevedere metodologie per la soluzione dei conflitti. A questo proposito è buona norma che gli agenti e la piattaforma coinvolti nella transazione mantengano record atti alla soluzioni di eventuali dispute.

3.1.5 Spionaggio

I container monitorano costantemente le comunicazioni e le istruzioni eseguite dagli agenti, è dunque facile per un container malizioso scoprire strategie di ne-

goziazione, algoritmi proprietari segreti, ecc. tramite l'analisi del codice degli agenti.

3.1.6 Alterazione

Quando un agente arriva su di una piattaforma per prima cosa espone ad essa il suo codice, il suo stato e i suoi dati i quali possono venire modificati da una piattaforma maliziosa. L'alterazione può essere scoperta da una piattaforma successivamente visitata dall'agente se questa possiede la firma digitale del codice originale.

3.1.7 Copia e replicazione

Muovendosi tra differenti piattaforma un agente aumenta la sua esposizione a rischi riguardanti la sicurezza. Un'entità estranea può infatti intercettare l'agente o un suo messaggio e dunque copiarlo ed eventualmente inoltrarlo dove desidera.

3.2 Contromisure

Alcune tecniche convenzionali, comunemente utilizzate nell'ambito dei sistemi distribuiti, possono venire applicate anche per una gestione della sicurezza di un sistema ad agenti mobili, altre sono insufficienti e devono quindi essere modificate.

3.2.1 Protezione della piattaforma

Le tecniche più recenti per proteggere una piattaforma d'agenti sono:

Sandboxing: il codice insicuro viene eseguito all'interno di un unico spazio di indirizzamento virtuale ed i moduli vengono modificati in modo che gli accessi alla memoria siano confinati all'interno dei segmenti di codice e dei dati all'interno di un unico *dominio di fallimento*. Essendo gli ambienti ad agenti spesso implementati con linguaggi interpretati (ad esempio JAVA è

largamente utilizzato in applicazioni di questo genere), è abbastanza semplice per le *piattaforme* costruire ambienti isolati per la loro esecuzione. Come già avviene per le *Applet*, tramite il *Security Manager* di JAVA, si potrebbero porre importanti limitazioni al comportamento degli agenti.

Safe Code Interpretation: le tecnologie di sistemi ad agenti sono spesso sviluppate utilizzando linguaggi interpretati a cause dell'eterogeneità delle macchine sulle quali sono ospitati, questo consente una più facile identificazione dei comandi che possono essere considerati pericolosi e dunque consentirne l'esecuzione solo ad entità autorizzate.

Signed Code: tramite la firma digitale è possibile intuire l'integrità o meno dell'oggetto.

State Appraisal: permette di scoprire se lo stato dell'agente è stato alterato in qualche modo.

Path Histories: viene mantenuta traccia delle piattaforme precedenti sul quale l'agente è stato ospitato. Perché si riesca ad utilizzare questa tecnica ogni piattaforma deve aggiungere la sua firma nel record indicante il percorso compiuto dall'agente, e la firma della piattaforma successiva.

Proof Carrying Code: questo approccio obbliga l'autore dell'agente a provare che il codice dell'agente è conforme alle direttive di sicurezza stipulate con l'utente finale.

3.2.2 Protezione degli agenti

Le tecniche più recenti per proteggere una piattaforma ad agenti sono:

Partial Result Authentication Codes: tecnica che si propone di proteggere l'autenticità dello stato dell'agente o dei risultati (anche parziali) dovuti all'iterazione con il server. L'agente cifra queste informazioni usando le diverse chiavi che ha a disposizione. Ogni volta che l'agente migra su di un altro host il suo stato, o le altre informazioni critiche, vengono processati e viene generato un MAC (Menage Authentication Code). Questa tecnica permette di riscontrare eventuali manomissioni del codice.

Mutual Itinerary Recording: è una variazione del Path Histories con l'ausilio di uno o più agenti che cooperano alla ricostruzione dell'itinerario effettuato dell'agente. Quando un agente migra, trasmette informazioni riguardo l'attuale piattaforma dalla quale è ospitato, l'hop precedente e l'hop successivo, ai peer tramite un canale autenticato. Questi peer tengono traccia dell'itinerario ed effettuano operazioni appropriate quando si verificano delle inconsistenze. Questo approccio prevede che soltanto un minoranza delle piattaforme siano maliziose. Le difficoltà di questo approccio sono dovute alla instaurazione di un canale sicuro e al fatto che, in caso di morte dell'agente, non si riesce a determinare quale tra la piattaforma attuale e quella successiva a terminato il ciclo di vita dell'agente.

Itinerary Recording with Replication and Voting: tecnica che usa più copie dello stesso agente per effettuare la computazione. Una piattaforma maliziosa può corrompere più di un agente, ma se il numero di copie è abbastanza elevato la computazione dovrebbe andare a buon fine. Ad ogni fase vengono controllate le credenziali dell'agente e vengono inoltrati verso la fase successiva soltanto gli agenti replicati considerati validi (che avranno risultati equivalenti). Uno dei protocolli usati in questa tecnica richiede che agenti memorizzino anche le firme di tutti i nodi da cui sono stati ospitati. Questo metodo è simile al Path Histories, ma viene introdotta la tollerabilità al guasto. Uno svantaggio evidente è costituito dalle elevate risorse supplementari consumate dai cloni dell'agente che, rendono in molti casi impossibile l'applicazione di tale tecnologia.

Execution Tracing: tecnica che tramite la registrazione delle esecuzioni degli agenti determina modifiche non autorizzate. Ogni piattaforma deve dunque mantenere i log delle operazioni compiute da tutti gli agenti. Gli inconvenienti sono dati, oltre che dalla crescita potenzialmente vertiginosa dei file di log, anche dalle difficoltà che si incontrano nel decidere la politica con cui le azioni degli agenti vengono registrate.

Environmental Key Generation: permette ad un agente di utilizzare codice o informazioni cifrate. L'agente deve avere accesso ad alcuni servizi della

piattaforma per verificare che alcune particolari condizioni siano o meno verificate. In caso queste condizioni siano verificate l'agente può decifrare le informazioni critiche.

Computing with Encrypted Function: si cerca un metodo che funzioni autonomamente senza interazioni con la piattaforma domestica in grado di elaborare in maniera sicura elementi crittografici, ad esempio una firma digitale, in modo tale da poter eseguire il codice su di una piattaforma fidata. La piattaforma deve eseguire il programma comprendente una funzione cifrata senza potere discernere la funzione originale.

Obfuscated Code: strategia che consente di trasformare un'applicazione in un'altra ma con lo stesso comportamento. Ovviamente il codice dell'applicazione trasformata dovrà essere difficile da comprendere. La sicurezza di questa tecnica si basa dunque sulla sofisticatezza delle trasformazioni e dell'algoritmo di offuscamento. Al giorno d'oggi esistono numerosi programmi che attuano l'offuscamento (ma ne esistono altrettanti che operano il deffuscamento), del codice. I programmi di offuscamento più sofisticati operano trasformazioni preventive che rendono difficile l'impiego di tecniche di deoffuscamento note.

Anche se le tecniche di offuscamento del codice diventano sempre più sofisticate, non si ha ancora, e probabilmente non si avrà mai, lo stesso grado di sicurezza che si ha nel cifrare il codice nativo. Nonostante questo l'offuscamento del codice detiene alcuni vantaggi di notevole rilevanza rispetto a quest'ultimo: il codice offuscato a differenza del codice nativo cifrato mantiene l'indipendenza della piattaforma e non necessita di firma per evitarne le manomissioni. Un'ultimo fatto importante è che usando codice cifrato, la JVM non riesce ad effettuare la verifica del bytecode, e sarebbe dunque possibile l'esecuzione di operazioni illegali da parte dell'applicazione (o, in questo contesto dell'agente). La documentazione relativa alle tecniche di offuscamento del codice (JAVA) è molto numerosa, ma un'interessante report sulle tecniche e tool di offuscamento del codice JAVA lo si può trovare in [1].

Parte II

Capitolo 4

Gestione della Sicurezza in JADE

Come già osservato nel capitolo precedente, i componenti di una piattaforma ad agenti che possono dar luogo, deliberatamente o meno, a comportamente malevoli sono sia i container che gli agenti.

In letteratura con *principal* si indica un' *entità in grado di compiere azioni di cui è responsabile*, dunque, sia i container che gli agenti possono sicuramente essere considerati dei principal. Anche gli utenti, pur non compiendo azioni direttamente ma tramite i propri agenti o container, possono essere considerati tali, al contrario, le entità esterne non devono essere prese in considerazione in questo conteso.

Nel progettare un sistema di sicurezza bisognerebbe sempre prevedere l'obbligo per ogni principal di autenticarsi e solo dopo permettergli l'accesso alle risorse della piattaforma.

Solitamente nei sistemi complessi i permessi non sono attribuiti direttamente ai singoli principal, ma vengono piuttosto associati ai *ruoli* che questi ricoprono. I ruoli sono organizzati in *gruppi* ai quali un principal può essere aggiunto acquisendone così tutti i privilegi associati.

In un sistema ad agenti una gerarchia di principal potrebbe rappresentare ogni utente come un nodo padre per tutti i suoi agenti e container, o ancora, ogni container essere visto come un nodo padre per tutti gli agenti da lui ospitati.

Risorse e domini

Rendere sicura una piattaforma ad agenti significa:

- proteggere tutte le *risorse* ospitate da minacce esterne ed interne. Per risorse si intende non solo l'infrastruttura della piattaforma, ma anche le risorse del livello applicazione e dei livelli sottostanti.
- proteggere la piattaforma da entità esterne
- prevenire che gli agenti danneggino altri agenti e che abbiano accesso a risorse e/o servizi senza previa autorizzazione.

Gestire gruppi di risorse tramite un *domain administrator*, semplifica la gestione dei controlli di accesso.

I *permission* esprimono le azioni che i principal possono eseguire e le risorse alle quali possono accedere.

I permessi tipicamente sono composti da un *target* e da una lista di azioni che sono permesse ai principal con caratteristiche conformi al target. I target e le azioni dipendono dal dominio in questione. I permessi vengono spesso raggruppati in file di *policy*.

4.1 Panoramica sulla sicurezza in Java

Java è un linguaggio ad oggetti il cui *bytecode* viene eseguito all'interno di un ambiente protetto (la *Java Virtual Machine*) che ne permette l'indipendendenza dalla piattaforma ed una particolare predisposizione alla gestione delle problematiche inerenti la sicurezza. Tutti questi aspetti rendono Java particolarmente adatto allo sviluppo di sistemi distribuiti.

Allo scopo di generare codice sicuro la sintassi del linguaggio è molto rigida: il compilatore di bytecode obbliga infatti a definire i tipi dei dati, il numero e il tipo degli argomenti dei metodi ed il valore di ritorno di questi. Questo però potrebbe non bastare (si pensi al caso di bytecode generato da un compilatore Java non standard), per questo motivo la *Java Virtual Machine* compie un elevato numero di controlli sul bytecode prima di eseguirlo. Il *Class Loader*, che si occupa

del caricamento effettivo delle classi, ne effettua preventivamente un controllo dell'URL di provenienza e, quando incontra una classe presente sia in locale che in remoto, preferisce la prima ritenendola più sicura. Il *Security Manager* ha la funzione di gestire l'utilizzo delle librerie native e dunque di permetterne o meno l'utilizzo diretto o indiretto in base alle poliche di sicurezza contenute in appositi file di configurazione, chiamati security policy.

Struttura delle Policy in Java

Le autorizzazioni nei file di policy vengono epresse tramite istruzioni del tipo:

```
grant {
permission <classepermesso> [parametriopzionali];
};
```

I permessi, che appartengono a diverse categorie (domini), incluse nei diversi package del JDK, vengono concessi solo se esplicitamente dichiarati nel file di policy. Se nella stesura della policy non viene specificato il firmatario o l'origine delle classi i permessi si intendono riferiti a qualunque codice non firmato o firmato da chiunque e per qualsiasi origine.

Questa struttura degli elenchi dei permessi consente controlli molto particolareggiati per la gestione della sicurezza.

Il codice può inoltre venire firmato tramite crittografia a chiave pubblica.

Questo modello è dunque centrato sul codice e non consente meccanismi di *autenticazione e autorizzazione di un utente*, ma solo *autenticazione e autorizzazione di codice*. In altre parole i permessi vengono concessi o meno in base alla *provenienza* del codice, ma non in base a *chi* esegue il codice.

4.1.1 JAAS

Questo fatto può creare delle limitazioni nello sviluppo di alcune applicazione, ad esempio è possibile che si vogliano garantire a diversi utenti diversi privilegi, per queste ragioni è stato sviluppato JAAS (Java Authentication and Authorization Service), un insieme di Java packages per l'autenticazione e autorizzazione dell'utente. Utilizzando JAAS è possibile sviluppare applicazioni che utilizzino una

combinazione dei due modelli di controllo d'acceso: *code-centric* e *user-centric access control*.[16]

L'*autenticazione* è il meccanismo che consente di determinare univocamente chi è l'utente che vuole eseguire il codice, mentre il meccanismo di *autorizzazione* provvede ad assicurarsi che il richiedente abbia effettivamnte il permesso per effettuare l'azione.

L'autenticazione in JAAS viene compiuta in modalità *pluggable*. Questo permette all' applicazione Java di rimanere indipendente dalle tecnologie sottostanti per l'autenticazione. L'implementazione di questa non è altro che un'estensione dello schema d'autorizzazione standard del J2SE basato sui file di policy.

L'autenticazione consiste nel creare un istanza di un oggetto LoginContext, il quale necessita del nome di una *Configurazione* per determinare quale LoginModule verrà creato. Il LoginModule definisce la tecnologia per l'autenticazione, e utilizza username e password per effettuare l'autenticazione. Per autenticare un soggetto sono seguiti i seguenti passi:

- l'applicazione istanzia un LoginContext,
- il LoginContext consulta una Configurazione per caricare il LoginModule configurato per l'applicazione,
- l'applicazione invoca il metodo login del LoginContext
- il metodo login invoca il LoginModule caricato. Il LoginModule cerca di autenticare il soggetto. Se l'autenticazione ha successo, il LoginModule associa al soggetto l'username e la password forniti,
- il LoginContext ritorna lo stato dell'autenticazione all'applicazione,
- se l'autenticazione ha avuto successo si ha l'introduzione al sistema,
- L'applicazione può recuperare il soggetto autenticato dal LoginContext,
- il LoginContext consulta una Configurazione per determinare il LoginModule configurato per una particolare applicazione.

Di conseguenza, è possibile utilizzare differenti LoginModule, senza cambiare l'applicazione stessa.

4.2 Peculiarità di JADE-S

In questa sezione verrà descritto l'approccio utilizzato da **JADE-s v2** per la gestione della sicurezza, in particolare verranno descritti:

- la catena di delegazione per il controllo d'accesso,
- la firma e la cifratura dei messaggi.

Maggiori informazioni a riguardo possono essere tratte dall'articolo [18] riportato in bibliografia.

4.2.1 Catena di Delegazione dei Diritti di Accesso

La gestione tradizionale del controllo di accesso si basa sulle *ACL* (Access Control List); queste, nella loro accezione più semplice, non sono altro che insiemi di permessi ordinati per nome e, quando è richiesta un'azione viene prima controllato che il richiedente sia riconosciuto come un utente fidato, poi, nel caso l'autenticazione abbia dato esito positivo, vengono esaminati i permessi associati all'utente e, se è autorizzato a compiere l'azione richiesta, questa viene eseguita.

E' facile capire che questo approccio centralizzato non è adatto per un sistema distribuito e dinamico quale è una piattaforma ad agenti. Quando il numero di utenti diventa molto elevato le ACL diventano molto numerose e difficili da gestire, ma soprattutto è difficile modellare un sistema basato su ACL che debba far fronte a relazioni effimere che possono cambiare rapidamente e senza che i componenti abbiano una reciproca conoscenza.

Un metodo differente consiste nel firmare insiemi di permessi garantiti da un principal autenticato. Questo modello evita l'utilizzo di grandi liste centralizzate e consente una maggiore libertà per quel che riguarda la delegazione dei permessi, oltre che una maggiore adattabilità ai cambiamenti di un ambiente dinamico.

Se un principal ha determinati permessi, può decidere di delegarli ad altri agenti tramite un *certificato di delegazione* contenente, oltre alla lista dei permessi che intende delegare, anche un periodo di validità che regoli l'entrata in vigore e il termine del permesso. L'agente delegante deve anche apporre la sua firma digitale al certificato.

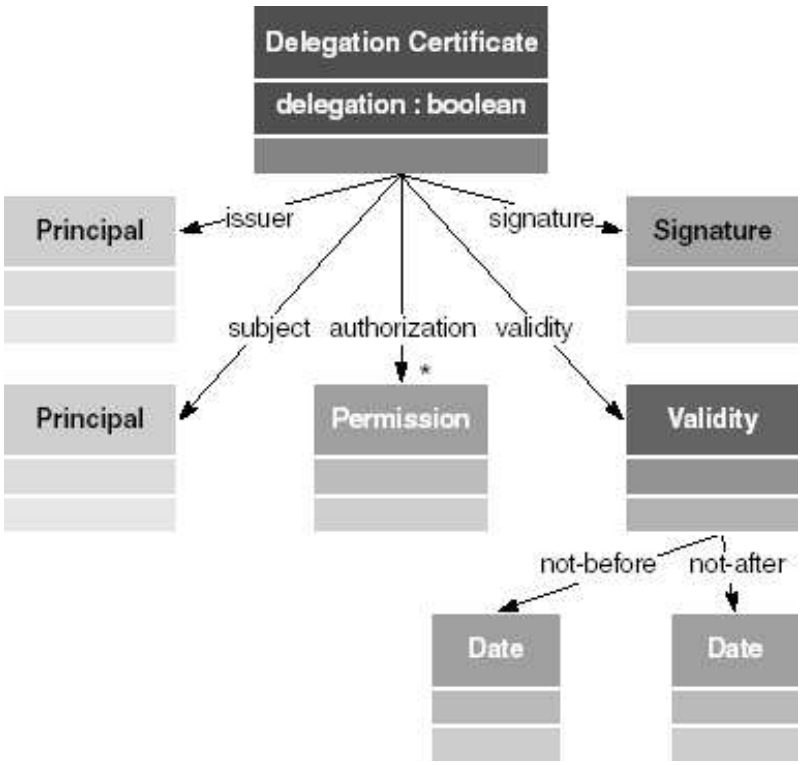

Figura 4.1: struttura certificato di delegazione

L'agente ricevente può decidere a sua volta di delegare i permessi acquisiti ad un'ulteriore entità, apponendo la sua firma e dando così vita ad una catena, o addirittura una rete, di delegazione. Per una trattazione più ampia delle reti di delegazione è possibile fare riferimento a [23] e [22].

Queste tecniche di delegazione dei certificati vengono largamente utilizzate anche in altri contesti: ad esempio, nell'ambito dei database, dove un utente che crea una tabella, o qualunque altro oggetto del database, ne e' il proprietario e gli vengono automaticamente garantiti tutti i privilegi applicabili a tale oggetto, con la possibilita' di impostare anche ad altri utenti tali privilegi (privilegio di concessione). Un utente che abbia un privilegio ed abbia in oltre su di esso il privilegio di concessione puo' assegnare tale privilegio ad un altro utente e passare ad esso anche il privilegio di concessione. I privilegi sono concessi da chi ne abbia il permesso (cioe' dal proprietario dell'oggetto e da chi abbia il privilegio di concessione) mediante il comando GRANT e revocati mediante il comando

REVOKE. Maggiori informazioni a riguardo possono essere reperite in [4].

Se si descrive il sistema come un insieme di peer, è facile osservare che ognuno di questi può svolgere il ruolo sia di *controller* che di *requester* per una determinata risorsa. Ogni componente è un'autorità che deve proteggere le sue risorse e che si assume la responsabilità di iniziare una catena di delegazione. Questo metodologia consente la definizione di policy a differenti livelli: piattaforma, container e agente.

Controllo di Accesso Basato su SPKI

Esitono principalmente due metodologie per realizzare il controllo d'accesso tramite certificati:

- controllo d'accesso basato sull'identità
- controllo d'accesso basato sulla chiave.

La prima, collega il certificato emesso da un principal, contenente il suo nome, con un altro certificato emesso da un'autorità riconosciuta, la seconda metodologia invece, sfruttando SPKI (Simple Public Key Infrastructure), utilizza un solo certificato emesso direttamente dal principal.

4.2.2 SPKI

Lo svantaggio principale nell'utilizzare il controllo basato sull'identità consiste nel fatto che questo necessita della presenza di un'entità di certificazione che firmi il certificato di identità, in questo modo ci sono due entità emettittrici di certificati che possono potenzialmente essere attaccate.

Nel secondo caso bisogna proteggere una sola chiave e l'unico requisito è che il principal possieda una coppia di chiavi crittografiche asimmetriche.

Rappresentare il principal con la sua chiave pubblica consente una delegazione sicura senza la presenza di una terza parte esterna. I certificati SPKI, nella loro implementazione più semplice, assicurano direttamente i permessi alla chiave e quindi il nome non è direttamente coinvoltro nel meccanismo consentendo quindi anche un certo livello di privacy al momento dell'accesso al servizio. Una più

ampia discussione sulle problematiche di autenticazione tramite chiava pubblica e sulle *web of trust* la si può trovare in [5].

Verifica della catena di certificati

Al momento della richiesta di un servizio il principal deve allegare alla sua domanda l'intera catena di certificati. Il gestore della risorsa, prima di concederla al richiedente, deve controllare ogni certificato e, se un certificato è scaduto, viene immediatamente scartato. Il periodo di validità dell'intera catena è dato dall'intersezione dei periodi di validità definiti per tutti i certificati. Se per accedere ad una determinata risorsa sono necessari più permessi, allora alla richiesta possono venir allegate più catene e ovviamente la risorsa viene concessa se tutte le catene sono valide.

4.2.3 Firma e Cifratura dei Messaggi

In JADE la comunicazione tra i container di una stessa piattaforma può venire cifrata tramite TLS (Transport Layer Security); questo però non basta, in quanto anche la comunicazione tra gli agenti risiedenti anche su piattaforme differenti dovrebbe poter avvenire in maniera sicura. La FIPA ha emanato un'apposito standard il quale prevede che la cifratura dei messaggi ACL avvenga sopra il livello di payload, in modo da proteggere anche gli slot del messaggio e per consentire una più semplice verifica da parte del ricevente.

In conclusione JADE-S permette il controllo d'accesso, un sofisticato meccanismo di delegazione dei permessi, la definizione di policy a diversi livelli e l'instaurazione di canali di comunicazione sicuri all'interno della piattaforma.

4.3 Sicurezza in JADE-S

Autenticazione

L'*autenticazione* è il primo passo per l'accesso ai servizi del sistema. L'autenticazione garantisce che l'utente che cerca di avviare un componente, sia esso

un agente o un container, sia considerato affidabile dall'host ospitante il main-container della piattaforma. L'autenticazione avviene per mezzo una coppia username - password che identifica univocamente l'utente e si basa su JAAS e dunque possono venire utilizzati diversi meccanismi per il login; l'attuale versione di JADE-S supporta i meccanismi Unix, NT e Kerberos e simple. Mentre i moduli per l'autenticazione di tipo Unix e NT dipendono dalla piattaforma utilizzata, Kerberos è indipendente dal sistema, ma prevede una configurazione specifica. Il quarto meccanismo, simple, si basa sulle informazoni contenute in un semplice file di testo contenente le coppie username-password. Questo meccanismo è stato utilizzato per la realizzazione dei test sulla piattaforma ad agenti del progetto SEWASIE. Il meccanismo di callback avviene invece o tramite command line, o tramite file di testo o per mezzo di una dialog box invocata al momento dell'avvio del container.

Permessi e Policy

Grazie al meccanismo di autenticazione si ha la garanzia che solo determinati utenti possano avere accesso alla piattaforma. Il passo successivo è quello di associare ad ogni componente appropriati permessi. Questi permessi sono descritti in opportuni file di policy secondo la sintassi Java/JAAS. L'architettura di JADE prevede la distinzione di due file di policy che garantiscono agli agenti

- permessi sull'intera piattaforma o
- permessi sui container periferici.

I permessi sulla piattaforma vengono caricati al momento dell'avvio del main-container; quando un agente gode di un certo permesso sulla piattaforma, questo permesso varrà anche su ogni altro container. Ad ogni container periferico è associato un ulteriore file di policy che vale solo sul container considerato.

Il supporto per la sicurezza è implementato come un insieme di JADE Services, i più importanti dei quali sono:

- jade.core.security.SecurityService: gestisce meccanismo di autenticazione, oltre che importanti funzionalità di crittografia e gestione della coppie di chiavi.

- jade.core.security.permission.PermissionService: è il servizio incaricato di verificare che l'agente che sta eseguendo una determinata azione (invio di un messaggio, richiesta all'AMS di creare un'altro agente, etc.) abbia le opportune credenziali per quella determinata azione.
- jade.core.security.signature.SignatureService: è il servizio che si occupa della firma dei messaggi e della verifica della validità di questa.
- jade.core.security.encryption.EncryptionService: questo servizio consente la cifratura e decifratura dei messaggi.

Quando si vuole che la piattaforma utilizzi i servizi per la sicurezza offerti da JADE-S è necessario specificarlo al momento del lancio.

Integrità dei messaggi e confidenzialità

La firma dei messaggi e la loro cifratura garantiscono un certo livello di sicurezza all'interno però della sola piattaforma. Il messaggio ACL è composto di due parti: *envelope* che contiene le informazioni indispensabili per il trasporto, e *payload* che contiene i dati e le informazioni che vogliono essere trasmessi. Spesso l'intera payload viene cifrata per impedire entità non autorizzate di carpirne il contenuto; in questo caso l'envelope conterrà anche la firma, l'algoritmo di cifratura o la chiave.

LIMITAZIONI

Al momento le piattaforme JADE che fanno uso del security add-on, non permettono la mobilità degli agenti, perché questa non è supportata da JADE-S, inoltre i messaggi scambiati tra gli agenti, sono sì trasmessi su un canale sicuro, ma non sono firmati (quest'ultimo problema è stato risolto con l'introduzione della versione 3 di JADE-S).

4.4 JADE e i firewall

La piattaforma JADE è interessata da un problema di rilevante importanza: una piattaforma distribuita su più host non funziona se questi sono protetti da firewall.

Per poter risolvere questo drammatico malfunzionamento è necessario studiare i meccanismi che permettono la mobilità e lo scambio di messaggi in JADE: questi meccanismi sono implementati sfruttando il protocollo *Java Remote Method Protocol (JRMP)*.

4.4.1 RMI

JRMP funziona sopra il protocollo TCP/IP e permette ad un programma in esecuzione su una JVM di effettuare chiamate a metodi presenti su un'altra JVM, anche se questa si trova su una macchina remota. Un client può invocare metodi di oggetti remoti con la stessa sintassi utilizzata per invocare i metodi locali. L'API RMI fornisce classi e metodi che gestiscono tutte le comunicazioni ed i parametri sottostanti, occupandosi degli accessi ai metodi di oggetti remoti. Nel Java 2 SDK versione 1.3, al JRMP è stata aggiunta una versione di RMI denominata RMI-IIOP che si appoggia al protocollo Internet Inter-ORB Protocol (IIOP) definito dall'Object Management Group (OMG). L OMG è il gruppo creatore dell'architettura distribuita *object-oriented* indipendente dalla piattaforma e multilinguaggio chiamata CORBA (Common Object Request Broker Architecture). Il protocollo IIOP è utilizzato dai client e dai server *CORBA Object Request Broker* (ORB) per le loro comunicazioni. RMI-IIOP permette a Java l'interazione con l'architettura CORBA e quindi con le applicazioni distribuite multipiattaforma sviluppabili in vari linguaggi, permesse da tale importante architettura. Riassumiamo brevemente i concetti fondamentali:

- RMI, come dice il suo stesso nome, permette l'invocazione di metodi su un oggetto remoto, che cioè risiede in un'altra Virtual Machine, eventualmente su un altro nodo della rete,
- Il meccanismo funziona a partire dalla definizione di un'interfaccia che descrive quali sono i metodi remoti invocabili,
- Il programmatore deve scrivere l'implementazione dell'interfaccia, che è poi l'oggetto remoto vero e proprio, mentre un compilatore apposito, RMIC, genera automaticamente due oggetti che gestiscono le due estremità della comunicazione (lo STUB e lo SKELETON).

- Infine un registro apposito (RMIREGISTRY) permette di associare oggetti remoti ai loro nomi, e di ottenere una reference remota di un oggetto a partire dal suo nome.

Dunque RMI funziona basandosi sul paradigma client/server: un server istanzia e registra gli oggetti remoti che vengono messi a disposizione di uno o più client. È un meccanismo chiaramente asimmetrico, dal momento che l'iniziativa parte sempre dal client. Se però volessimo che su due nodi della rete ci siano oggetti che possono invocarsi reciprocamente saremmo costretti, ad esempio, a lanciare il demone rmiregistry e registrare gli oggetti remoti su entrambi gli host: a questo punto avremmo configurato due server, ed ognuna delle due macchine può essere il client dell'altra. Questa soluzione è piuttosto complessa.

Fortunatamente, RMI mette a disposizione una modalità peer-to-peer che, contrapponendosi alla client/server, permette l'invocazione reciproca tra due oggetti remoti richiedendo che *solo uno* di essi (il server) si registri con il rmiregistry. Il suo funzionamento è veramente molto semplice: se un client ha già effettuato con successo il collegamento con il server remoto, il client stesso diventa un oggetto remoto invocabile dal server.

E' facile capire che lo sviluppo di applicazioni ad agenti mobili, trae indubbi benefici, in fase di realizzazione, dall'utilizzo di RMI: la modalità peer-to-peer in particolare è di grande aiuto nello sviluppo di questo tipo di applicazioni.

RMI e firewall

In una tipica applicazione del mondo reale, la rete viene protetta da possibili aggressori esterni attraverso un firewall.

Il firewall viene configurato per garantire l'accesso solo a quei servizi che l'amministratore di sistema ha classificato come leciti o non pericolosi, e sbarra la strada a tutto il restante traffico. Quasi sicuramente un firewall non è configurato per il protocollo JRMI in quanto poco utilizzato.

Il protocollo JRMP prevede che il server utilizzi due porte: la prima, solitamente la 1099, è fissa e viene utilizzata per l'azione di ascolto, la seconda invece è casuale e viene utilizzata per la trasmissione dei dati. L'utilizzo di questa porta

casuale impedisce alle applicazioni che sfruttano RMI di lavorare correttamente in presenza di firewall.

Per poter operare attraverso un firewall, RMI mette a disposizione due meccanismi che mascherano il traffico JRMI in modo da farlo apparire come traffico HTTP (supponendo ovviamente che esso possa attraversare il firewall). Entrambi i meccanismi sono implementati a livello di RMI transport e quindi sono praticamente trasparenti al programmatore.

Se il meccanismo di trasporto RMI non riesce a contattare il server direttamente, esso tenta una connessione sulla porta 80 (quella del protocollo HTTP), impacchettando, la richiesta RMI in una HTTP POST (il tipico messaggio che viene generato da una form HTML). I server RMI si accorgono della situazione, ed estraggono automaticamente la richiesta RMI dal corpo della POST; al momento di mandare indietro una risposta al client, la reimpacchettano in un messaggio HTTP per permettere di nuovo l'attraversamento del firewall. Tutta quest'operazione viene chiamata HTTP tunnelling.

Non è richiesta nessuna modifica al codice Java, ma il programmatore deve solo accertarsi della configurazione del firewall ed agire di conseguenza:

1. il firewall potrebbe essere stato configurato per redirigere le richieste HTTP su una porta arbitraria; in tal caso è sufficiente lanciare il server RMI per rispondere su quella porta. Tuttavia questo approccio impedirebbe l'uso in contemporanea di un server WWW tradizionale,

2. alternativamente è possibile installare un apposito script CGI-BIN su un server WWW pre-esistente per gestire la situazione. Questo script deve rispondere alla URL /cgi-bin/java-rmi, ed un esempio, chiamato java-rmi.cgi, è incluso nel JDK.

Tutto questo meccanismo è controllato dai metodi createSocket() e createServerSocket() della classe java.rmi.server.rmiSocketFactory, che un programmatore esperto può ridefinire nel caso voglia implementare comportamenti non standard. Come si diceva prima, possono essere richieste alcune operazioni di configurazione sul client e sul server:

1. Se il client non si trova nel dominio del server, è necessario che i loro nomi siano specificati in forma completa (fully qualified). Questo può essere un

problema per il server: in funzione della piattaforma dove è installata e della sua configurazione, non sempre la Java Virtual Machine è in grado di ottenere automaticamente il nome del dominio in cui si trova. Per esempio, se il nome del server è myserver.domain.com, può capitare che la JVM non sia in grado di ottenere la parte domain.com, definendo quindi l'host locale come myserver. In questi casi, il nome completo del server deve essere specificato nella proprietà di sistema java.rmi.server.hostname.

2. Visto che il client tenta l'HTTP tunnelling automaticamente, nel caso di errori sulla rete, o se semplicemente il server RMI non è stato attivato o se il suo indirizzo è sbagliato, passa un po' di tempo prima che il runtime Java si accorga del problema e segnali l'errore. Se si è sicuri che non si vuole tentare l'HTTP tunnelling, questa modalità si può disabilitare dal client impostando la proprietà di sistema java.rmi.server.disableHttp con il valore true.

Se si deve passare attraverso un firewall, solo il server RMI può esportare un oggetto remoto, e quindi la modalità peer-to-peer descritta precedentemente non funziona.

Come ultima considerazione, va purtroppo tenuto presente che, in caso di HTTP tunneling, RMI diventa almeno dieci volte più lento (ovviamente il valore esatto dipende dalle tante variabili in gioco: velocità del firewall, del server WWW, eccetera).

Per ovviare questi problemi sono state prese in considerazione più strade: le due vie più accreditate, per la loro semplicità di realizzazione e per il non decadimento delle performance, erano la modifica della classe RMISocketFactory in modo da eliminare l'aleatorietà della porta e l'introduzione di JADE-LEAP. E' stata scelta la seconda strada in quanto la più semplice.

4.4.2 JADE LEAP

JADE-LEAP è nato con lo scopo di sopperire alle limitazioni che subentravano in un contesto di reti wireless per quel che riguarda i meccanismi di mobilità e comunicazione di JADE. In questo scenario infatti firewall e indirizzi IP dinamici

rendono impossibile, o molto complesso, il funzionamento di RMI. Il nuovo protocollo introdotto da JADE-LEAP, per qeal che riguarda la comunicazione intra-piattaforma, è JICP (JADE Inter Container Protocol). In JADE-LEAP sono state sviluppate nuove librerie che sostituiscono alcuni componenti di JADE, in modo tale che la piattaforma appaia come una normale piattaforma di JADE e le API rimangano le stesse. Come già detto, tutte le comunicazioni tra container JADE sono rese possibili da RMI il quale però non è supportato dall'edizione di java per i dispositivi mobili, così si è deciso di reimplementare i meccanismi di mobilità. All'inizio si pensava di adottare SOAP (Single Object Access Protocoll) che è uno dei protocolli per l'implementazione di servizi Web più analizzati degli ultimi tempi, basato su XML consente a due applicazioni di comunicare tra loro, indipendentemente dall'hardware e software e dai linguaggi di programmazione utilizzati per svilupparle. Il problema è che l'uso di XML per le comunicazioni wireless è troppo oneroso. Ogni container in JADE-LEAP è visto come un *JICP Peer*. I messaggi degli agenti ed ogni altra informazione che deve esser trasmessa viene opportunamente convertita e quindi passata al JICP Peer il quale la incapsula in un opportuno frame e la inoltra, tramite socket TCP/IP, verso il container remoto. Dopo aver spedito il frame il JICP Peer rimane in attesa di un frame di risposta (di solito un acknowledge).

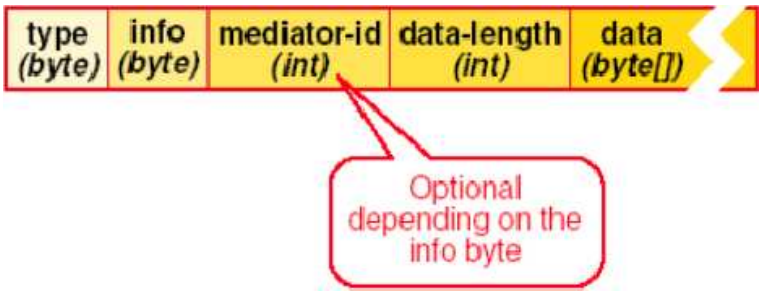

Figura 4.2: frame JIPC

Per implementare il meccanismo di comunicazione JADE-LEAP utilizza per default la socket messe a disposizione da java, ma è possibile per l'amministratore del sistema d'agenti, decidere di utilizzare altri protocolli che meglio si addicono all'ambiente preso in esame. L'utilizzo delle socket consente di impostare

una porta fissa e consente dunque l'esecuzione dell'applicazione anche dietro un firewall.

Capitolo 5

Configurazione della Piattaforma

Tra i requisiti dell'architettura della piattaforma di agenti del progetto SEWASIE vi sono sicuramente *scalabilità* e *sicurezza* ma non l'interoperabilità con altre piattaforme.

La piattaforma di agenti non si pone dunque l'obiettivo di una completa conformità agli standard FIPA.

In questa parte si vogliono analizzare metodologie per garantire

- l' identificazione degli utenti,
- il controllo degli agenti,
- la protezione dei dati.

L'add-on JADE-S consente di soddisfare tutti i maggiori requisiti richiesti dal documento Guidelines of general security and policy for the development of system modules [11] inerenti alla gestione della sicurezza.

5.0.3 Scenario di test

Il primo passo perché la piattaforma sia abilitata al supporto per la sicurezza e perché possa essere sfruttato il nuovo kernel di JADE introdotto con JADE-LEAP è l'installazione degli appositi add-on: JADE-S e JADE-LEAP. Per quanto riguarda JADE-S è sufficiente scaricare il package con già presente il bytecode. L'add-on JADE-LEAP è presente solo in formato sorgente ed è quindi necessario generare

il bytecode JAVA. Per la generazione del bytecode è necessario l'uso di *ant*, utile tool che automatizza la compilazione di sorgenti java. Al momento della generazione del bytecode è da considerare il tipo di sistema sul quale la piattaforma JADE-LEAP dovrà lavorare; sono infatti previste tre modalità di generazione del bytecode, ognuna delle quali con requisiti diversi:

- j2se: è la modalità da scegliere se JADE-LEAP andrà eseguito su un PC.
- pJAVA: serve per eseguire JADE-LEAP su dispositivi palmari
- midp: per eseguire JADE-LEAP su dispositivi che supportano MIDP1.0 (la maggior parte dei cellulari che supportano JAVA)

La piattaforma ad agenti di SEWASIE viene chiaramente eseguita su di un server connesso ad una rete fissa, è quindi la prima opzione quella che verrà utilizzata. Se fosse stato richiesto l'utilizzo di una delle altre due modalità sarebbe stato necessario possedere i pacchetti JAVA j2me-wtk-home (nel caso di modalità pjava) o il Sun J2ME Wireless ToolKit (nel caso di JADE-LEAP per midp).

Configurazione della piattaforma multi-utente

Per lanciare una piattaforma multiutente sicura il primo passo da compiere è decidere la scelta del tipo di autenticazione da utilizzare. Per i test è stata utilizzato un semplice file di testo contenente le associazioni username-password.

Listing 5.1: password.txt

```
alice    alice
bob      bob
```

Passo successivo è stato quello di creare il file di configurazione per il main-container. In questo file (main.conf) sono contenute le informazioni indispensabili per l'instaurazione di *join* tra il main-container e i container periferici, come il nome e la porta da utilizzare. La parte successiva evidenzia i servizi, oltre a quelli di default, da attivare sulla piattaforma. Con l'opzione gui settata su false viene impedita la creazione delll'RMA, l'interfaccia grafica di JADE; questa verrà creata da l'opportuna classe JAVA rappresentante lo scenario di test. Con il parametro java.security.policy si indica il percorso dove trovare il file di policy contenente le

informazioni sui permessi. Risulta evidente che questo è un parametro di JAVA, non di JADE. Nella sezione relativa all'autenticazione va specificato innanzitutto il tipo di login; in questo caso è stato scelto il tipo Dialog il quale si occupa di far apparire una dialog box, durante il caricamento della piattaforma, nella quale andranno introdotta una coppia username e password. In caso di discordanza tra i dati introdotti nella dialog box e quelli contenuti nei file password.txt (listing 5.1) il caricamento verrà interrotto, in caso contrario verrà consultato il file policy.txt (listing 5.4) e, se l'utente ha i permessi adeguati, sarà infine avviata la piattaforma. Per evitare di introdurre ad ogni avvio della piattaforma username e password, è possibile memorizzare questi nel file di configurazione e scegliendo la modalità Cmdline. Infine viene espilicitato il percorso dove trovare il file jaas.conf contenente la configurazione di JAAS; tale configurazione è descritta nel (listing 5.3)

Listing 5.2: main.conf

```
#----JADE configuration----

#------Platform------
name=apollo
port=2000
#------Services  ------
services=\
jade.core.security.SecurityService;\
jade.core.security.permission.PermissionService;\
jade.core.security.signature.SignatureService;\
jade.core.security.encryption.EncryptionService;\
jade.core.event.NotificationService

#------Agents  ------
gui=false

#------Transport------
nomtp=true

#------Security configuration------

#----Permission----
# Permission Policy file
java.security.policy=confSec/policy.txt

#----Authentication----
```

```
# - Type of Prompt - can be: {Cmdline, Text, Dialog}
#('Text' does not work well with 'ant')
jade.security.authentication.logincallback=Dialog

# - if Cmdline, use this user/pass -
#owner=alice:alice

# - Auth module - can be:{Simple,Unix,NT,Kerberos}
jade.security.authentication.loginmodule=Simple

# - if Simple, use this password file
#(passwords.txt is the default value)
jade.security.authentication.loginsimplecredfile=\
confSec/passwords.txt

# - JAAS configuration file -
java.security.auth.login.config=confSec/jaas.conf

#---- end JADE configuration ----
```

Essendo JAAS basato su PAM (Pluggable Authentication Mode), in fase di configurazione oltre a specificare quali LoginModule devono essere usati, è necessario esplicitare come l'esito positivo di questi modifichi o meno la sequenza di login. Questo è possibile tramite l'utilizzo di particolari attributi chiamati *JAAS Control Flag*.

I valori che i *JAAS Control Flag* possono assumere sono i seguenti:

- REQUIRED: l'utente deve passare il test di autenticazione e ulteriori *Authentication provider* sono sempre chiamati
- SUFFICIENT: se l'utente passa il test di autenticazione non devono essere chiamati ulteriori *Authentication provider*
- REQUISITE: se l'utente passa questo test vengono invocati ulteriori *Authentication provider*, ma questi possono fallire.
- OPTIONAL: all'utente è permesso passare o fallire questo controllo.

Per la configurazione della nostra applicazione è stato sempre utilizzato il *JAAS Control Flag* required.

Listing 5.3: jaas.conf

```
/**
** JAAS configuration file
**/
Simple {
jade.core.security.authentication.SimpleLoginModule required;
};

Unix {
com.sun.security.auth.module.UnixLoginModule required;
};

NT {
com.sun.security.auth.module.NTLoginModule required;
};

Kerberos {
com.sun.security.auth.module.Krb5LoginModule required;
};
```

Assegnazione dei permessi

Dopo aver opportunamente configurato la piattaforma è possibile pasare all'assegnazione dei permessi. Si vuole ottenere un sistema in cui ogni container possiede un ben preciso proprietario. Nei test effettuati esistono solo due container: il main-container ed un container periferico. L'utente che detiene il permesso di avviare e gestire il main-container è stato chiamato alice, mentre l'utente responsabile del container periferico bob. Sono inoltre configurabili, grazie ai *target constrain* impostazioni che consentono:

- ad alcuni tipi di agente di concedere servizi solo ad altri tipi di agenti,
- ad alcuni tipi di agente di concedere servizi solo ad agenti di un determinato proprietario,
- alle sorgenti di rilasciare informazioni solo a certi agenti di certi proprietari.

Per l'assegnazione dei permessi si usa la sintassi utilizzata da JAAS per i file policy. I permessi che possono essere assegnati agli utenti per permettergli o meno l'accesso ai servizi della piattaforma e dei suoi elementi sono:

- jade.security.AgentPermission,

- jade.security.MessagePermission,
- jade.security.PlatformPermission,
- jade.security.ContainerPermission.

Ad ognuno dei permessi descritti possono essere associate determinate *azioni* e *target constraint* secondo il modello:

```
grant principal jade.security.Name "<principalName> {
  permission <permissionClass> "<targetConstraints>", "<actions>";
};
```

Le *azioni* ed i *target constrains* che possono essere associati alle categorie di permessi sono rappresentate nella seguente tabella:

Permission	actions	Target Constrains
AgentPermission	create kill	agent-owner agent-name container-owner agent-class
AgentPermission	suspend resume	agent-owner agent-name agent-class
MessagePermission	send-to	agent-owner agent-name
PlatformPermission	create kill	
ContainerPermission	create kill	container-owner

Sarà dunque possibile fare in modo che su di un container solo gli agenti con un determinato nome e/o di uno specifico proprietario possano essere autorizzati a spedire messaggi, o sospendere un altro agente o ogni altra azione consentita ad un agente. Come si può osservare combinando insieme i vari *target constrains* è possibile creare configurazioni per l'assegnazione dei permessi molto selettive.

Perchè il codice di JADE possa venire eseguito è necessario che gli vengano assegnati gli opportuni permessi JAVA. Senza questa operazione JAVA non permetterà l'utilizzo del codice di JADE, con l'ovvia conseguenza dell'impossibilità di avviare la piattaforma ad agenti. Per questa ragione la prima cosa da fare è garantire a tutte le classi del progetto opportuni permessi JAVA. Per la fase di test sono stati garantiti tutti i permessi previsti da JAVA a tutto il codice. Passo successivo è la creazione di opportune policy per ogni utente della piattaforma ad agenti. In conformità con quanto visto precedentemente alla piattaforma ad agenti possono accedere soltanto due utenti: alice e bob.

```
// JADE code is allowed all permissions.
grant codebase "file:../lib/add-ons/security/lib/jadeSecurity.jar" {
        permission java.security.AllPermission; };
grant codebase "file:../lib/jade.jar" {
        permission java.security.AllPermission; };
grant codebase "file:../lib/jadeTools.jar" {
        permission java.security.AllPermission; };
grant codebase "file:../lib/add-ons/leap/j2se/lib/JadeLeap.jar" {
        permission java.security.AllPermission; };
grant codebase "file:../lib/xercesImpl.jar" {
        permission java.security.AllPermission; };
grant codebase "file:../lib/xmlParserAPIs.jar" {
        permission java.security.AllPermission; };
grant codebase "file:../lib/xml-apis.jar" {
              permission java.security.AllPermission; };
grant codebase "file:../lib/momis.jar" {
        permission java.security.AllPermission; };
grant codebase "file:../lib/Base64.jar" {
      permission java.security.AllPermission; };
grant codebase "file:../lib/http.jar" {
      permission java.security.AllPermission; };
grant codebase "file:../lib/iiop.jar" {
      permission java.security.AllPermission; };
grant codebase "file:../lib/mssqlserver.jar" {
        permission java.security.AllPermission; };
grant codebase "file:../lib/msbase.jar" {
      permission java.security.AllPermission; };
grant codebase "file:../lib/msutil.jar" {
      permission java.security.AllPermission; };

grant principal jade.security.Name "alice" {
permission jade.security.PlatformPermission "", "create,kill";
permission jade.security.ContainerPermission "", "create,kill";
permission jade.security.AgentPermission "", "create,kill";
permission jade.security.AgentPermission "", "suspend,resume";
permission jade.security.AMSPermission   "", "register,deregister,modify";
```

```
permission jade.security.MessagePermission "", "send-to";
};

grant principal jade.security.Name "bob" {
permission jade.security.ContainerPermission "container-owner=bob", "create";
permission jade.security.ContainerPermission "container-owner=bob", "kill";
permission jade.security.AgentPermission "agent-owner=bob" , "create";
permission jade.security.AgentPermission "agent-name=bob-*", "create";
permission jade.security.AgentPermission "agent-owner=bob", "kill,suspend";
permission jade.security.AgentPermission "agent-owner=bob", "resume";
permission jade.security.AMSPermission   "agent-owner=bob", "register";
permission jade.security.AMSPermission   "agent-owner=bob", "deregister";
permission jade.security.AMSPermission   "agent-owner=bob", "modify";
permission jade.security.MessagePermission "", "send-to";
};
```

Listing 5.4: policy.txt

Alice è l'utente di maggiore importanza in quanto è l'unica che può avviare il main container, aggiungervi o eliminare agenti, oltre che sospenderli o risvegliarli. L'utente chiamato bob può effettuare le stesse operazioni ma solo su di un container periferico, o su agenti, di sua proprietà. A questo punto la configurazione del main-container è terminata ed è possibile lanciarlo. Configurazione analoga va effettuata anche per il container periferico che verrà collegato al main-container. In questo caso il file di configurazione è quasi identico a quello del main-container:

```
#----JADE configuration----
container=true
port=1098
#------ Services ------
services=\
jade.core.security.SecurityService;\
jade.core.security.permission.PermissionService;\
jade.core.security.signature.SignatureService;\
jade.core.security.encryption.EncryptionService;\
jade.core.event.NotificationService

#------ Security configuration------

#---- Permission----
# Permission Policy file
java.security.policy=confSec/cpolicy.txt

#---- Authentication----
# - Type of Prompt -
#can be: {Cmdline, Text, Dialog}
```

```
#('Text' does not work well with 'ant')
jade.security.authentication.logincallback=Cmdline

# - if Cmdline, use this user/pass -
owner=bob:bob
```

Listing 5.5: cont-1.conf

La differenza principale è la presenza del parametro container il quale evidenzia il fatto che il container in questione è un container periferico il quale va collegato ad un main container. Anche per questo container va creato un opportuno file di policy, le quali però sono valide solo in questo ambiente.

```
// JADE code is allowed all permissions.
grant codebase "file:../lib/add-ons/security/lib/jadeSecurity.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/jade.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/jadeTools.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/add-ons/leap/j2se/lib/JadeLeap.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/xercesImpl.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/xmlParserAPIs.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/xml-apis.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/momis.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/Base64.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/http.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/iiop.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/mssqlserver.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/msbase.jar" {
      permission java.security.AllPermission;  };
grant codebase "file:../lib/msutil.jar" {
      permission java.security.AllPermission;  };
// --- Startup example---
// --- Policy on the remote container---


grant principal jade.security.Name "bob" {
        permission jade.security.AgentPermission "", "create,kill";
        permission jade.security.AgentPermission "", "suspend,resume";
```

```
        permission jade.security.AMSPermission "", "register,deregister";
        permission jade.security.AMSPermission "", "modify";
};

grant principal jade.security.Name "*" {
        permission jade.security.MessagePermission "", "send-to";
};
```

Listing 5.6: cpolicy.txt

Avvio della piattaforma di test

Per avviare la piattaforma esistono principalmente due metodologie:

- tramite linea di comando
- tramite classe JAVA che si occupa di creare gli opportuni container e agenti.

All'interno del progetto SEWASIE era stata adottata la seconda opzione, si è scelto dunque di continuare nella stessa direzione.

```
/**
org.sewasie.test.ScenarioSec.java

Avvia una piattaforma con il controllo per la sicurezza
fornito dall'add-on jadeSecurity.jar.
Per lanciarlo ha bisogno di tre parametri:
  1) file di configurazione della piattaforma
  2) file di configurazione per il FactoryAgent
  3) porta sulla quale il container (non main) lavora
*/

package org.sewasie.test;

import jade.core.Runtime;
import jade.core.Profile;
import jade.core.ProfileImpl;
import jade.wrapper.*;
import jade.core.AID;

public class ScenarioSec{

/**
* The scenario main port
*/
private static int _scenarioPortNumber = 1099;
//private static int _scenarioPortNumber = 1101;
```

```
/**
* Getter for the _scenarioPortNumber field */
public static int getScenarioPortNumber(){
        return(_scenarioPortNumber);
}
 /**
 * Setter for the _scenarioPortNumber field */
public static void setScenarioPortNumber(int v ){
        _scenarioPortNumber = v;
}

public static void main(String args[]) {
  try {
    // Get a hold on JADE runtime
    Runtime rt = Runtime.instance();
    //Exit the JVM when there are no more containers around
    rt.setCloseVM(true);

    //Create a Profile object initialized with the settings
    //specified in a given property file
    Profile pMain = new ProfileImpl(args[0].toString());

    System.out.println("Launching a whole in-process
        platform..."+pMain);
    AgentContainer mc = rt.createMainContainer(pMain);

    //Create a Profile object initialized with the settings
    //specified in a given property file
    AgentContainer container1 = rt.createAgentContainer(
        new ProfileImpl(args[1]));
    AgentController rma = mc.createNewAgent("rma",
        "jade.tools.rma.rma", new Object[0]);
    rma.start();

    AgentController fa = container1.createNewAgent(
        "FA1",
        "org.sewasie.agents.factory.FactoryAgent",
        newObject[]{args[1]}
    );
    fa.start();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

Listing 5.7: ScenarioSec.java

Come specificato nei commenti iniziali, la classe ScenarioSec.java accetta tre parametri: i due file di configurazione relativi al main-container ed al container periferico, il numero della porta sulla quale lavorerà.

Con

```
Runtime rt = Runtime.instance();
```

viene creata una nuova JVM sulla quale viene avviata un'istanza di JADE. Successivamente viene creato un profilo, ovvero un'instanza della classe ProfileImpl, contenente tutte le impostazioni di configurazione necessarie all'avvio del main-container. Le impostazioni del profilo vengono ricavata dal primo argomento passato all'applicazione, che non è altro che il file main.conf descritto nel listing 5.2. Successivamente viene creato il main container

```
AgentContainer mc = rt.createMainContainer(pMain);
```

passandogli in fase di creazione il riferimento al profilo creato sulla base del file main.conf. Operazioni analoghe vanno effettuate per il container periferico. Il secondo file di configurazione, oltre ad alcuni parametri utili per la creazione del container, contiene anche parametri di configurazione per l'agente FactoryAgent che verrà creato nelle righe successive con

```
AgentController fa = container1.createNewAgent("FA1",
        "org.sewasie.agents.factory.FactoryAgent",
        new Object[]{args[1]});
```

Viene creata anche un'istanza dell'RMA per il monitoraggio degli agenti.

```
AgentController rma = mc.createNewAgent("rma", "jade.tools.rma.rma",
        new Object[0]);
rma.start();
```

Per l'avvio dello scenario di test è stato creato un'apposito script che si preoccupa di lanciare la classe ScenarioSec.

```
@echo off
set CLASSPATH=.;.\commons-launcher.jar;.\lib\launcher;\
.\lib\launcher\ant-launcher.jar;.\lib\launcher\ant.jar
rem echo %CLASSPATH%
rem echo .
start JAVA LauncherBootstrap -verbose runScenarioSec confSec\main.conf\
confSec\cont-1.conf 1097 %4 %5 %6 %7 %8 %9
```

Lo script invoca il LauncherBootstrap, un'applicazione che si occupa del reperimento e del lancio della JVM e della classe richiesta. Per poter permettere al LauncherBootstrap di trovare ScenarioSec è stato modificato il file built.xml aggiungendo il percorso dove trovare il bytecode della classe desiderata:

```
<target name="runScenarioSec" description="Starts LEAP S platform" >
  <launch classname="${project.package}.test.ScenarioSec">
    <classpath refid="classpath"/>
  </launch>
</target>
```

A questo punto è possibile avviare lo scenario di test.

Capitolo 6

Conclusioni e Sviluppi Futuri

Per la realizzazione dell'elaborato presentato è stato effettuato uno studio sui modelli architetturali dei sistemi distribuiti, approfondendo in modo particolare il paradigma ad agenti, soprattutto per quel che concerne la gestione della sicurezza in ambienti aperti. I concetti acquisiti sono stati quindi utilizzati per un'analisi delle caratteristiche della piattaforma ad agenti JADE; tale analisi, oltre a consolidare le conoscenze teoriche sui sistemi ad agenti, ha anche portato ad una maggiore comprensione del funzionamento della JVM e dei meccanismi per l'autenticazione utilizzati in JAVA e dunque dei meccanismi per la gestione dei permessi utilizzati in JADE in quanto estensione di quanto proposto da JAVA. Infine le nozioni apprese sono state applicate per introdurre la gestione della multiutenza sulla piattaforma ad agenti del progetto SEWASIE e per consentire a questa di poter operare correttamente anche su host protetti da firewall.

Per quanto riguarda gli sviluppi futuri della piattaforma ad agenti SEWASIE la prima cosa che si potrebbe realizzare è l'effettiva distribuzione dei container periferici su host differenti, in modo tale da distribuire il carico di lavoro che altrimenti obererebbe una singola macchina. Un'altra interessante possibilità sarebbe quella di sfruttare il servizio di replicazione del main container previsto da JADE per la realizzazione di una piattaforma tollerante ad eventuali cadute del container principale.

Tra le prerogative del progetto SEWASIE non vi era l'interoperabilità con altre piattaforme; sono state brevemente illustrate alcune problematiche di sicurezza a cui sono sottoposti i sistemi distribuiti in generale, evidenziando inoltre come queste possano venire sfruttate in un contesto di sistema ad agenti. Come già evidenziato nella sezione di competenza, la tecnologia attuale non è ancora in grado di fornire risposte adeguate alle problematiche di sicurezza che coinvolgono gli ambienti di agenti mobili tra piattaforme

eterogenee. Alcune proposte sono elevata complessità, mentre altre non sono ancora in uno stadio di maturità.

Appendice A

Secure Platform Designer

Obiettivo della tesi era anche quello di realizzare un tool grafico che consentisse di configurare i permessi distribuiti di jade. Il tool è una semplice interfaccia grafica dove, tramite una tabella, è possibile introdurre i permessi introdotti da JADE-S. Nelle prime tre colonne è possibile usufruire di una lista per la scelta del tipo di permesso, del tipo di target constrain e del tipo di azione da compiere. Nelle ultime due colonne vanno invece inseriti il nome del *principal* a cui si sta facendo riferimento ed eventuali stringhe da introdurre nel target constrain. Il tool è stato scritto in linguaggio JAVA, come ogni elemento di JADE, in modo da essere indipendente dalla piattaforma. Per la realizzazione dell'interfaccia grafica mi sono avvalso dei package inclusi nella Java Standard Edition, awt e swing. Nel seguito viene mostrata l'interfaccia grafica realizzata.

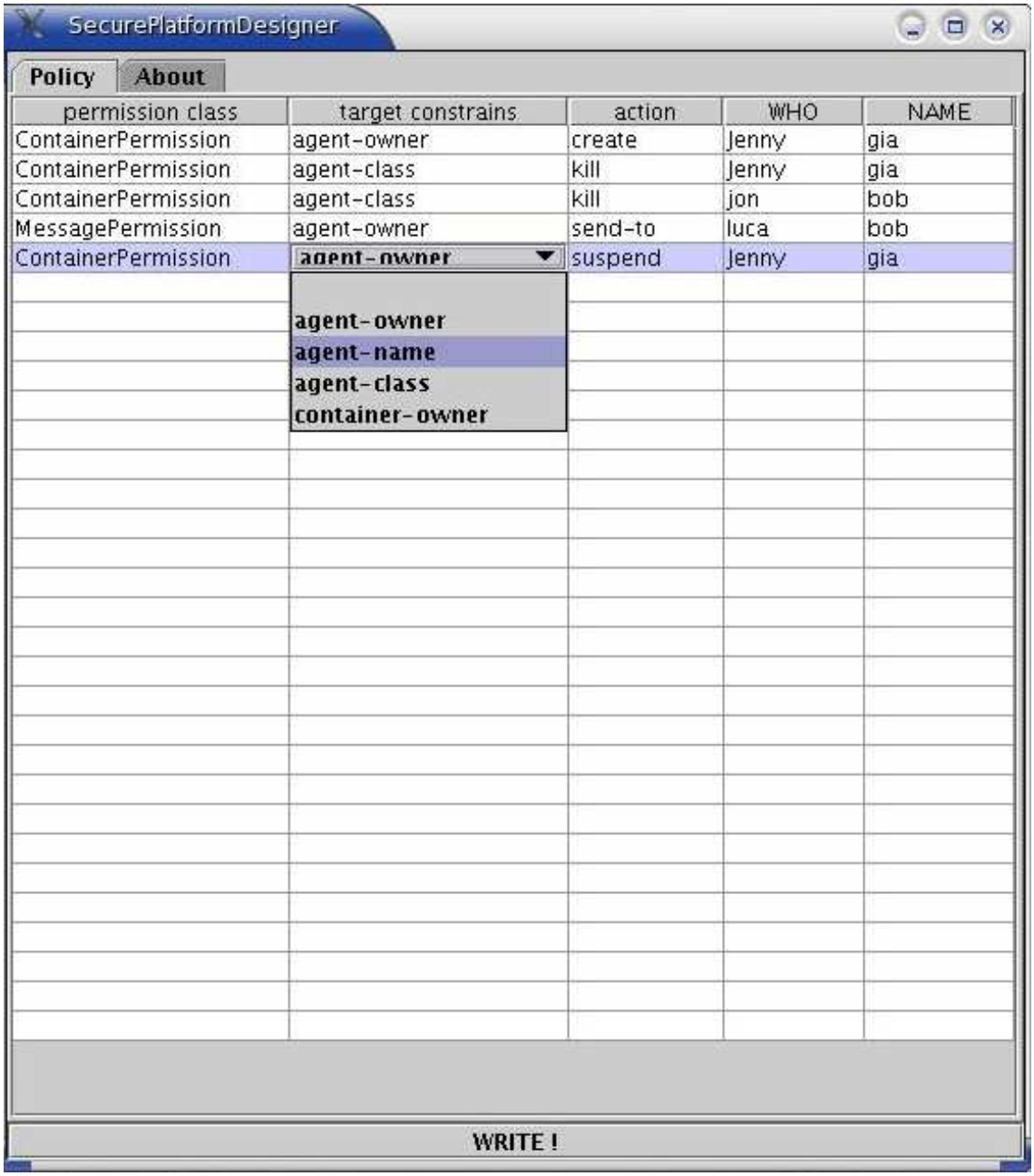

Figura A.1: security gui

Bibliografia

[1] Kilpatrick D. Matt B. Reisse A. Van Vleck T. Badger L., D'Anna L. Self-protecting mobile agents obfuscation techniques evaluation report. Technical report, NAI Labs, 2001.

[2] G. Caire. *LEAP USER GUIDE*. TILAB S.p.A., 2005.

[3] Larry Koved Anthony Nadalin Charlie Lai, Li Gong and Roland Schemers. User authentication and authorization in the java(tm) platform. *Proceedings of the 15th Annual Computer Security Applications Conference, Phoenix, AZ,*, December 1999.

[4] C.J. Date. *A Guide to The SQL Standard*.

[5] Lampson B. Rivest R. Thomas B. Ylonen T. Ellison C., Frantz B. Spki certificate theory. Technical report, IETF RFC 2693, 1999.

[6] A. Poggi G. Rimassa F. Bellifemine, G. Caire. Jade a white paper. *exp*, 2003.

[7] T. Trucco F. Bellifemine, G. Caire. *JADE ADMINISTRATOR'S GUIDE*. TILAB S.p.A., 2003.

[8] Foundation for Intelligent Physical Agent. *FIPA Agent Management Specification*, 2002.

[9] G. Rimassa G. Caire, N. Lhullier. A communication protocol for agents on handheld devices. *exp*, 2002.

[10] Fabrizio Giudici. *Programmazione avanzata in RMI*. http://mokabyte.it/.

[11] Daniele Montanari Guido Vetere. Guidelines of generale security and policy for the development of system modules. Technical report, SEWASIE consortium, 2002.

[12] JADE-Board. *JADE SECURITY GUIDE*. TILAB S.p.A., 2005.

[13] jGuru. *Foundamentals of RMI*. http://developer.java.sun.com.

[14] Bradshaw J.M. *Software Agents*. 1997.

[15] K. Sycara. *Multi-agent Infrastructure agent Discovery, Middle Agents for Web Services and Interoperation*.

[16] Qusay H. Mahmoud. Java authentication and authorization service (jaas) in java 2, standard edition (j2se) 1.4. *http://java.sun.com/developer/technicalArticles/Security/jaasv2/index.html*, September 2, 2003.

[17] Paolo Malacarne. *Java Servlet*. 2000.

[18] G. Vitaglione N. Lhullier, M. Tomaiuolo. Security in multi-agent systems: Jade-s goes distributed. *exp*, 2003.

[19] E. Franconi S. Tessaris S. Bergamaschi P. Fillottrani, J. Dix. Sewasie architecture agentization. Technical report, SEWASIE consortium, 2004.

[20] SEWASIE. http://www.sewasie.org/.

[21] SUN Microsystem, Inc. *RMI Specification*. http://java.sun.com/j2se/1.4/docs/guide/rmi/spec/rmiTOC.html.

[22] Aura T. On the structure of delegation networks. In *Proc. 11th IEEE Computer Security Foundations Workshop (Rockport MA)*, 1998.

[23] Aura T. Distribuited access-right management with delegation certificates. Technical report, 1999.

[24] Tom Karygiannis Wayne Jansen. Mobile agent security. Technical report, National Institute of Standard and Technology, Computer Security Division, 2000.

[25] wikipedia. http://it.wikipedia.org/.