

Alberto Cordioli - vr066075

Andrea Oboe - vr044794

uniBCM

Sistema di gestione esami sviluppato in Java RMI
per il corso di Programmazione Avanzata e di Rete



vr066075 Alberto Cordioli

vr044794 Andrea Oboe

Supervisore : Professor Massimo Merro

Facoltà di Scienze Matematiche, Fisiche e Naturali dell'Università di Verona
A.A. 2008/2009

Indice



Introduzione	2
Schema generale	3
Classi implementate	4
Interfacce implementate	5
Funzionamento del sistema	12
File di policy	16
Link utili	19

Introduzione

Il progetto che siamo andati a sviluppare consiste in un sistema Client-Server, basato su Java RMI/IIOP, rivolto alla gestione degli esami e dei corsi tenuti presso un particolare polo universitario. La scelta di realizzare questo determinato sistema ci ha portato ad applicare la maggior parte dei concetti studiati a lezione e di approfondire varie tematiche per meglio completare il progetto.

Come già accennato, uniBCM presenta una struttura tipicamente di rete, quale un'interazione tra Client e Server, che possiamo sintetizzare come:

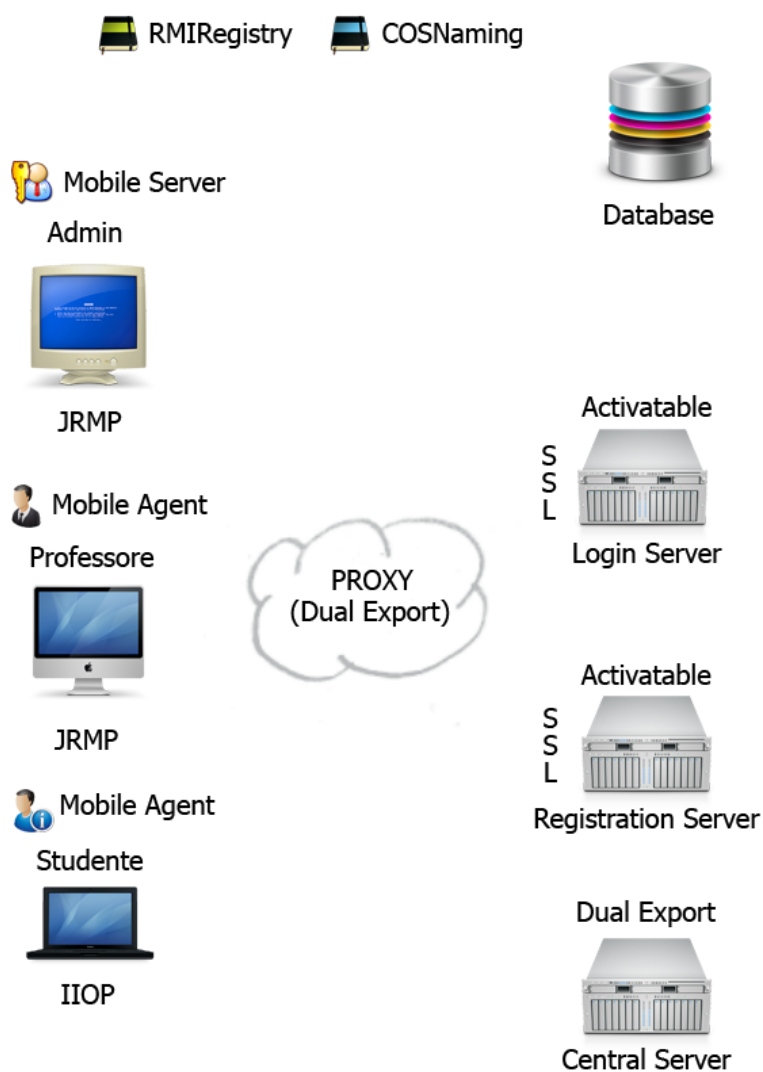
- Lato Server : vengono forniti vari servizi accessibili in modo esclusivo in base alla tipologia di Client. Il sistema presenta tre server (Login, Registration, Central) e un Proxy, i quali tratteremo esaustivamente successivamente;
- Lato Client : il sistema supporta due tipologie di client, ovvero JRMP e IIOP (questo per far vedere l'interoperabilità tra protocolli differenti), rispettivamente utilizzate da Amministratore/Professore e dallo Studente.

Il sistema è funzionante sia su protocollo file che http, previa scelta prima del lancio del sistema stesso.

Per eventuali chiarimenti sull'utilizzo del programma, si faccia riferimento agli script di lancio (denominati uniBCM sia lato server sia lato client).

Schema generale

Il sistema uniBCM presenta una struttura definita come nello schema sottostante.



Classi implementate

Andiamo a dare una breve descrizione delle classi che abbiamo realizzato e al loro generale funzionamento. Per semplicità suddividiamo la descrizione in base al package che le contiene.

- `server.central` : è il server centrale del sistema che permette di restituire e modificare una serie di informazioni attraverso un'interrogazione diretta al database. Inoltre permette di tener traccia di tutte le connessioni effettuate presso il sistema uniBCM e restituendo in tempo reale tutte le informazioni relative agli utenti connessi;
- `server.login` : permette l'autenticazione al sistema mediante una serie di controlli incrociati oltre ai canonici controlli di username e password. E' inoltre presente un metodo che permette di contattare il mobile server dell'Amministratore una volta che quest'ultimo è stato esportato a destinazione;
- `server.proxy` : permette di reindirizzare le richieste di login e di registrazione da parte degli utenti presso i rispettivi servizi;
- `server.reg` : è il server a cui si collega esclusivamente l'Amministratore e che permette di compiere tutta una serie di operazioni volte alla gestione dell'intero sistema uniBCM;
- `mobile.magent` : questo package contiene le classi dei mobile agent del Professore e dello Studente che operano esclusivamente con il Central Server. Ovviamente per ciascun tipo di utente le operazioni possibili saranno differenti;
- `mobile.mserver` : è il mobile server che andrà in esecuzione sulla macchina dell'Amministratore. Attraverso questa classe è possibile interagire con tutta una serie di metodi provenienti dai server centrale, registrazione e login. Essendo un mobile server, esso verrà esportato a destinazione e al suo interno sarà presente un metodo *ping()* che permetterà al Login Server di invocarlo per ottenere tutta una serie di informazioni riguardanti l'host ospitante la console di amministrazione;
- `client` : questo package contiene le classi per le due tipologie di client sviluppate, ovvero JRMP e IIOP;
- `lib` : al suo interno è possibile individuare tutta una serie di librerie utilizzate dal sistema uniBCM. Per non dilungarci troppo sulle singole classi si rimanda alla documentazione reperibile nella directory doc.

Interfacce implementate

Riportiamo qui di seguito tutte le interfacce utilizzate da uniBCM, riportando anche dei commenti che descrivono le singole operazioni compiute da ogni metodo.

server.central

```
public interface Central extends Remote {

    /**
     * Ottiene le informazioni dell'utente user
     * @param user utente
     * @return Stringa di informazioni
     * @throws RemoteException
     */
    public String getInfo(String user) throws RemoteException;

    /**
     * Aggiunge un esame ad un utente.
     * Utente, nome dell'esame e voto vengono passati come parametri.
     * -ATTENZIONE-l'utente deve essere uno studente
     * @param user utente
     * @param examName nome dell'esame
     * @param value voto in trentesimi
     * @return stringa di conferma
     * @throws RemoteException
     */
    public String addExam(String user, String examName, int value)
    throws RemoteException;

    /**
     * Aggiunge un corso ad un utente
     * Utente, nome del corso e anno vengono passati come parametri
     * -ATTENZIONE-l'utente deve essere un professore
     * @param user utente
     * @param courseName nome del corso
     * @param year anno accademico
     * @return Stringa di conferma
     * @throws RemoteException
     */
    public String addCourse(String user, String courseName, String year)
    throws RemoteException;

    /**
     * Restituisce la lista di corsi tenuti dal docente user nei vari anni
     * accademici
     * @param user utente professore
     * @return array di corsi
     * @throws ClassNotFoundException
     * @throws IOException
     * @throws RemoteException
     */
    public Corso[] getCoursesList(String user)
    throws ClassNotFoundException, IOException, RemoteException;
```

```

/**
 * Restituisce la lista degli esami sostenuti dallo studente user
 * @param user utente studente
 * @return array di Esami
 * @throws ClassNotFoundException
 * @throws IOException
 * @throws RemoteException
 */
public Esame[] getExamsList(String user)
throws ClassNotFoundException, IOException, RemoteException;

/**
 * Aggiunge la referenza ad un utente attualmente loggato.
 * Questo server per tenere traccia degli utente attualmente loggati
 * Viene invocato quando un client si connette al sistema
 * @param ref referenza client
 * @throws RemoteException
 */
public void addRef(Reference ref) throws RemoteException;

/**
 * Rimuove la referenza ad un utente attualmente loggato.
 * Questo server per tenere traccia degli utente attualmente loggati
 * Viene invocato quando un client si esce dal sistema o quando il client
non e' piu'
 * raggiungibile(per eventuali crash di sistema)
 * @param ref referenza client
 * @throws RemoteException
 */
public void removeRef(Reference ref) throws RemoteException;

/**
 * Ottiene la referenza all'ultimo client loggato
 * @return Ip dell'ultimo client loggato
 * @throws RemoteException
 */
public String getLastRef() throws RemoteException;

/**
 * Restituisce il numero di client attualmente loggati al sistema uniBCM
 * @return intero rappresentante il numero di client loggati
 * @throws RemoteException
 */
public int getReferenceCount() throws RemoteException;
}

```

server.login

```

public interface Login extends Remote {

/**
 * Effettua il tentativo di connessione da parte dell'utente checkUser
passato come parametro.
 * Se l'utente e' registrato al sistema gli verra' restituito un oggetto
di tipo Mobile
 * (MobileServer se utente amministratore, MobileAgentProf se prof,
MobileAgentStud se studente)
 * @param checkUser utente da loggare
 * @return oggetto di tipo Mobile se l'utente e' registrato al sistema,
null altrimenti
 * @throws RemoteException
 */

```

```

    public Mobile login(User checkUser) throws RemoteException;

    /**
     * Setta la referenza del MobileServer allo stub passato come
     parametro.
     * Questo server per sapere dove si e' collegato l'amministratore
     poiche' esso possiede
     * un MobileServer
     * @param stubMS stub al mobile server dell'amministratore
     * @throws RemoteException
     */
    public void setMSRef(MobileServerItf stubMS) throws RemoteException;
}

```

server.reg

```

public interface Registration extends Remote {

    /**
     * Invia la richiesta pendente al server che la salverà su un file
     temporaneo fino alla
     * sua conferma o cancellazione.
     * @param data richiesta pendente marshalizzata
     * @return stringa di conferma o errore
     * @throws RemoteException
     */
    public String submitUser(MarshalledObject data) throws RemoteException;

    /**
     * Conferma o rifiuta la richiesta pendente dell'utente passato come
     parametro.
     * Al termine del metodo la richiesta pendente verra' eliminata oppure
     confermata a seconda
     * del parametro accept.
     * @param id identificativo dell'utente
     * @param accept operazione: true conferma richiesta, false elimina
     richiesta
     * @return stringa di conferma
     * @throws RemoteException
     */
    public String confUser(String id, boolean accept)
    throws RemoteException;

    /**
     * Conferma tutti gli utenti che hanno effettuato una richiesta
     pendente.
     * Gli utenti verranno quindi registrati al sistema.
     * Questa chiamata ha lo stesso effetto di equivale ad eseguire
     confUser(user[i],true)
     * per ogni utente che ha effettuato una richiesta pendente.
     * @see Registration#confUser(String, boolean)
     * @return stringa di conferma
     * @throws RemoteException
     */
    public String confAllUser() throws RemoteException;
}

```



```

/**
 * Elimina l'utente con identificativo passato come parametro dal
 sistema uniBCM
 * @param id utente da eliminare
 * @return stringa di conferma
 * @throws RemoteException
 */
public String delUser(String id) throws RemoteException;

/**
 * Elimina tutte le richieste pendenti. Tutti gli utenti che hanno
 effettuato una richiesta
 * di registrazione non ancora confermata verranno eliminati dal sistema
 * Questa chiamata ha lo stesso effetto di equivale ad eseguire
 confUser(user[i],false)
 * per ogni utente che ha effettuato una richiesta pendente.
 * @see Registration#confUser(String, false)
 * @return stringa di conferma
 * @throws RemoteException
 */
public String delAllPendUser() throws RemoteException;

/**
 * Restituisce la lista di tutti gli utente registrati al sistema
 * @return array di User
 * @throws RemoteException
 * @throws ClassNotFoundException
 * @throws IOException
 */
public User[] userList()
throws RemoteException, ClassNotFoundException, IOException;

/**
 * Restituisce la lista di tutti gli utenti che hanno effettuato una
 richiesta pendente
 * non ancora gestita(confermata/eliminata).
 * @return array di User
 * @throws RemoteException
 * @throws IOException
 * @throws ClassNotFoundException
 */
public User[] pendingUserList()
throws RemoteException, IOException, ClassNotFoundException;

/**
 * Setta lo stato dell'utente con id passato come parametro a
 sospeso/ attivo a seconda
 * del parametro suspend
 * @param id Id dell'utente
 * @param suspend stato (true sospeso, false attivo)
 * @return stringa di conferma
 * @throws RemoteException
 */
public String setUserState (String id, boolean suspend)
throws RemoteException;
}

```

server.proxy

```
public interface Proxy extends Remote{

    /**
     * Tenta di effettuare il login al sistema dell'utente user passato come
     * parametro
     * In realta' il Proxy invochera' il server di Login il quale
     * controllera' se l'utente e'
     * registrato e ritornera' un oggetto di tipo Mobile. Il Proxy
     * marshalizza l'oggetto e lo
     * restituisce al client
     * @param user utente da loggare
     * @return oggetto di tipo Mobile marshallizzato se l'utente e'
     * registrato al sistema,null altrimenti
     * @throws RemoteException
     */
    public MarshalledObject login(MarshalledObject user)
    throws RemoteException;

    /**
     * Crea una nuova richiesta pendente dell'utente user. In realta' il
     * Proxy invochera'
     * il server di Registrazione per effttuare l'operazione e restituirà
     * una stringa di conferma
     * dell'operazione
     * @param user utente
     * @return stringa di conferma
     * @throws RemoteException
     */
    public String submitUser(MarshalledObject user) throws RemoteException;

}
```

mobile

```
/**
 * Definisce i metodi fondamentali per un tipo di oggetto che vuole
 * implementare l'interfaccia
 * Mobile. N.B. Tutti gli oggetti di tipo MobileServer,MobileAgent DEVONO
 * dare un'implementazione
 * a questa interfaccia per poter essere restituiti correttamente al client
 * @author uniBCM team
 */
public interface Mobile{

    /**
     * Manda in esecuzione la sessione del Mobile
     * @throws ClassNotFoundException
     * @throws IOException
     * @throws RemoteException
     */
    public void runSession()
    throws ClassNotFoundException, IOException, RemoteException;

}
```

mobile.mserver

```
/**
 * Definisce i metodi per il tipo MobileServer. N.B. Tutti gli oggetti di
 tipo MobileServer DEVONO
 * dare un'implementazione a questa interfaccia per poter essere restituiti
 correttamente al client.
 * MobileServer rappresenta un raffinamento dell'interfaccia Mobile per il
 tipo Server
 * @author uniBCM team
 *
 */
public interface MobileServerItf extends Remote, Mobile{

    /**
     * Effettua il ping al Mobile Server corrente.
     * Il metodo restituisce l'indirizzo IP sul quale e' in esecuzione
     l'istanza corrente
     * del Mobile Server
     * @return indirizzo IP
     * @throws RemoteException
     */
    public String ping() throws RemoteException;

}
```

mobile.magent

```
/**
 * Definisce i metodi per il tipo MobileAgent. N.B. Tutti gli oggetti di tipo
 MobileAgent DEVONO
 * dare un'implementazione a questa interfaccia per poter essere restituiti
 correttamente al client.
 * MobileAgent rappresenta un raffinamento dell'interfaccia Mobile per il
 tipo Agent
 * @author uniBCM team
 *
 */
public interface MobileAgentItf extends Mobile{

    /**
     * @see Mobile#runSession()
     */
    public void runSession() throws ClassNotFoundException, IOException,
 RemoteException;

}
```

client

```
/**
 * Definisce i metodi comuni a tutti i tipi di client, siano essi studenti,
 * professori
 * o amministratori
 * @author uniBCM Team
 */
public interface Client extends Runnable, Serializable {

    /**
     * Setta il server verso il quale questo client effettuerà la
     * connessione
     * @param server indirizzo ip del server
     */
    public void setServer(String server);

    /**
     * Setta il protocollo sul quale funzionerà il client di uniBCM team
     * @param protocol protocollo
     */
    public void setProtocol(String protocol);

    /**
     * Setta l'indirizzo IP del client.
     * N.B. Serve per IIOP non potendo usare la classe InetAddress per
     * recuperare
     * il proprio indirizzo IP
     * @param address
     */
    public void setIPAddress(String address);
}
```

Funzionamento del sistema

Prima di addentrarci nel dettaglio nel funzionamento del sistema, andiamo a definire le caratteristiche delle singole componenti che permettono a uniBCM di operare.

Lato Server abbiamo:

- Proxy, trasparente ai client che usufruiscono del sistema, è dual export e si occupa di gestire le richieste rivolte al server di Login e di Registration, occupandosi inoltre di marshallizzare il mobile server e il mobile agent restituito rispettivamente all'Amministratore e al Professore/Studente. Si ricordi inoltre che esso è l'unico server registrato presso RMIRegistry (porta 2222) e COSNaming (porta 5555);
- Login, di tipo Activatable, accetta connessioni SSL provenienti dal Proxy e attraverso un controllo delle credenziali fornite permette di restituire il mobile server/agent in caso di successo. Abbiamo implementato il sistema di autenticazione in modo tale che il client IIOP non possa mai ricevere il mobile server relativo all'amministratore (anche in caso di corretta autenticazione) per una questione di maggior sicurezza e, analogamente, il client JRMP non potrà mai loggarsi con delle credenziali di tipo studente. Quindi riassunto ciascuna tipologia di utente è vincolata ad utilizzare il tipo di client e protocollo a cui esso è assegnato;
- Registration, sempre di tipo Activatable, accetta anch'esso connessioni via SSL e oltre ad interagire con il mobile server dell'Amministratore, permette ad un nuovo utente del sistema di potersi registrare creando una nuova registrazione pendente che verrà autorizzata solamente dall'Amministratore di uniBCM;
- Central, dual export, è il cuore del sistema che oltre a restituire tutta una serie di informazioni relative agli utenti di uniBCM e ad aggiungere esami e corsi rispettivamente a studenti e professori, permette di tener traccia di tutti gli utenti che interagiscono con il sistema stesso.

Tutto il sistema che abbiamo creato e sostenuto da un database che ci permette di gestire più facilmente i vari utenti e tutti i dati relativi.

La scelta di realizzare un'interfaccia grafica è stata essenziale per permettere una maggiore usabilità all'utente finale.

Andiamo ora a definire la fase di start-up del sistema, per meglio comprendere le varie scelte effettuate (si faccia riferimento a Setup.java).

Innanzitutto viene settato il Security Manager e viene creata una nuova istanza di Central Server che, essendo dual export, rende necessario l'invocazione dei metodi

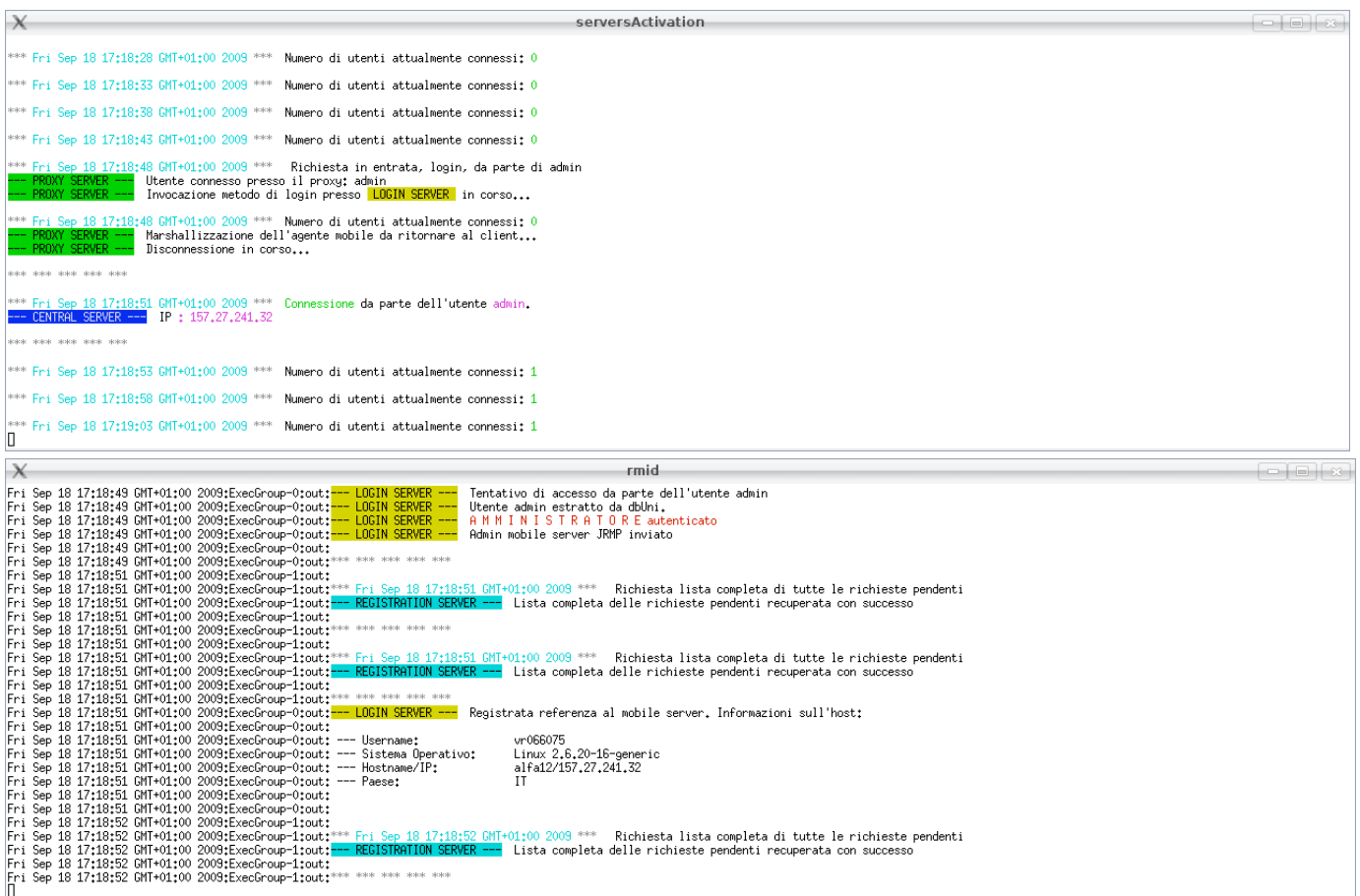
Alberto Cordioli - vr066075

Andrea Oboe - vr044794

UnicastRemoteObject.exportObject() e *PortableRemoteObject.exportObject()*. Dopo alcune inizializzazioni, quali creazione dell'utente Admin, vengono settate tutte le properties per i nostri due server Activatable, che comprendono definizione del codebase, file di policy, valore del LeaseValue per la DGC (settato di default a 2 minuti) e i parametri necessari per le connessioni SSL.

Una volta create tali properties, vengono creati due gruppi diversi di attivazione, rispettivamente per i due server Activatable; questa scelta è stata vincolata alla necessità di mantenere il più possibile un alto grado di sicurezza, in modo da celare alcune informazioni che risulterebbero altrimenti accessibili se entrambi i server risiedessero nello stesso gruppo. Inoltre si ricordi che Registration Server è accessibile esclusivamente dall'Amministratore e una scelta come quella appena descritta permette una buona sicurezza in fase gestionale del sistema.

Una volta conclusa anche questa fase, viene creata un'istanza di Proxy, che verrà esportata su entrambi i protocolli JRMP e IIOP (essendo dual export) e registrata sia su COSNaming che su RMIRegistry (si ricordi che solamente il Proxy è registrato presso i due servizi di naming). Da questo momento in poi il Proxy è accessibile dai due tipo di client previa lookup sul corrispondente registro.



```
*** Fri Sep 18 17:18:28 GMT+01:00 2009 *** Numero di utenti attualmente connessi: 0
*** Fri Sep 18 17:18:33 GMT+01:00 2009 *** Numero di utenti attualmente connessi: 0
*** Fri Sep 18 17:18:38 GMT+01:00 2009 *** Numero di utenti attualmente connessi: 0
*** Fri Sep 18 17:18:43 GMT+01:00 2009 *** Numero di utenti attualmente connessi: 0
*** Fri Sep 18 17:18:48 GMT+01:00 2009 *** Richiesta in entrata, login, da parte di admin
--- PROXY SERVER --- Utente connesso presso il proxy: admin
--- PROXY SERVER --- Invocazione metodo di login presso LOGIN SERVER in corso...
*** Fri Sep 18 17:18:48 GMT+01:00 2009 *** Numero di utenti attualmente connessi: 0
--- PROXY SERVER --- Marshalling dell'agente mobile da ritornare al client...
--- PROXY SERVER --- Disconnessione in corso...
*** **
*** Fri Sep 18 17:18:51 GMT+01:00 2009 *** Connessione da parte dell'utente admin.
--- CENTRAL SERVER --- IP : 157.27.241.32
*** **
*** Fri Sep 18 17:18:53 GMT+01:00 2009 *** Numero di utenti attualmente connessi: 1
*** Fri Sep 18 17:18:58 GMT+01:00 2009 *** Numero di utenti attualmente connessi: 1
*** Fri Sep 18 17:19:03 GMT+01:00 2009 *** Numero di utenti attualmente connessi: 1

Fri Sep 18 17:18:49 GMT+01:00 2009:ExecGroup-0:out:--- LOGIN SERVER --- Tentativo di accesso da parte dell'utente admin
Fri Sep 18 17:18:49 GMT+01:00 2009:ExecGroup-0:out:--- LOGIN SERVER --- Utente admin estratto da dbUnit.
Fri Sep 18 17:18:49 GMT+01:00 2009:ExecGroup-0:out:--- LOGIN SERVER --- A N N I S T R A T O R E autenticato
Fri Sep 18 17:18:49 GMT+01:00 2009:ExecGroup-0:out:--- LOGIN SERVER --- Admin mobile server JRMP inviato
Fri Sep 18 17:18:49 GMT+01:00 2009:ExecGroup-0:out:*** **
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-1:out:--- REGISTRATION SERVER --- Richiesta lista completa di tutte le richieste pendenti
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-1:out:--- REGISTRATION SERVER --- Lista completa delle richieste pendenti recuperata con successo
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-1:out:*** **
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-1:out:--- REGISTRATION SERVER --- Richiesta lista completa di tutte le richieste pendenti
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-1:out:--- REGISTRATION SERVER --- Lista completa delle richieste pendenti recuperata con successo
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-1:out:--- LOGIN SERVER --- Registrata referenza al mobile server. Informazioni sull'host:
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-0:out:--- Username: vr066075
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-0:out:--- Sistema Operativo: Linux 2.6.20-16-generic
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-0:out:--- Hostname/IP: alfa12/157.27.241.32
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-0:out:--- Paese: IT
Fri Sep 18 17:18:51 GMT+01:00 2009:ExecGroup-0:out:
Fri Sep 18 17:18:52 GMT+01:00 2009:ExecGroup-1:out:--- REGISTRATION SERVER --- Richiesta lista completa di tutte le richieste pendenti
Fri Sep 18 17:18:52 GMT+01:00 2009:ExecGroup-1:out:--- REGISTRATION SERVER --- Lista completa delle richieste pendenti recuperata con successo
Fri Sep 18 17:18:52 GMT+01:00 2009:ExecGroup-1:out:*** **
```

Abbiamo perciò il sistema uniBCM online e pronto a ricevere le richieste da parte dei client, che ora andiamo a descrivere.



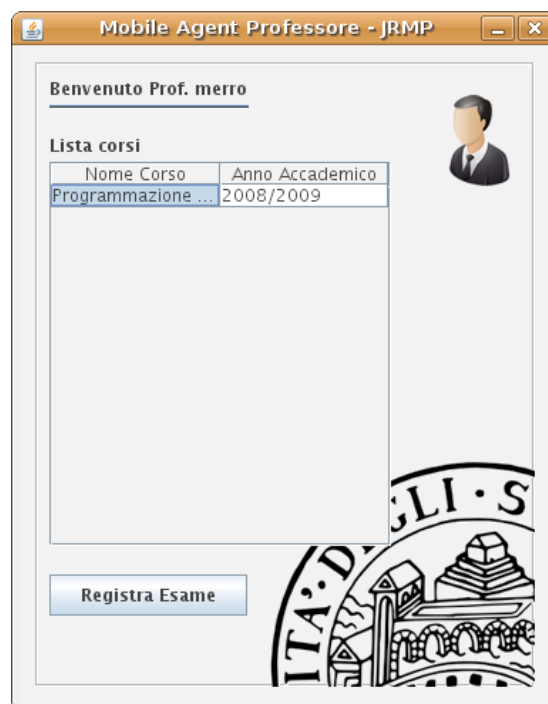
Abbiamo implementato due tipologie di client, ovvero JRMP che copre l'utente Amministratore/Professore e IIOP che si occupa in modo esclusivo degli account di tipo studente. Dopo il settaggio delle varie properties per le

connessioni SSL, i client eseguono una lookup sul corrispondente registro in base al protocollo utilizzato, ottenendo la referenza remota al Proxy, per poter invocare in modo completamente trasparente i metodi di *login()* o *submitUser()*. Concretamente i client non invocano direttamente i due metodi appena menzionati presso i rispettivi server di Login e Registration, applicando così correttamente l'idea del proxy.

Nel caso della richiesta di registrazione per un nuovo utente, dopo aver compilato il form in tutti i suoi campi, viene creata presso il server uniBCM una richiesta pendente, che una volta resa attiva, permetterà all'utente di poter autenticarsi con le credenziali specificate.

Nel caso invece di autenticazione avvenuta per un particolare utente ci troviamo in due casi distinti:

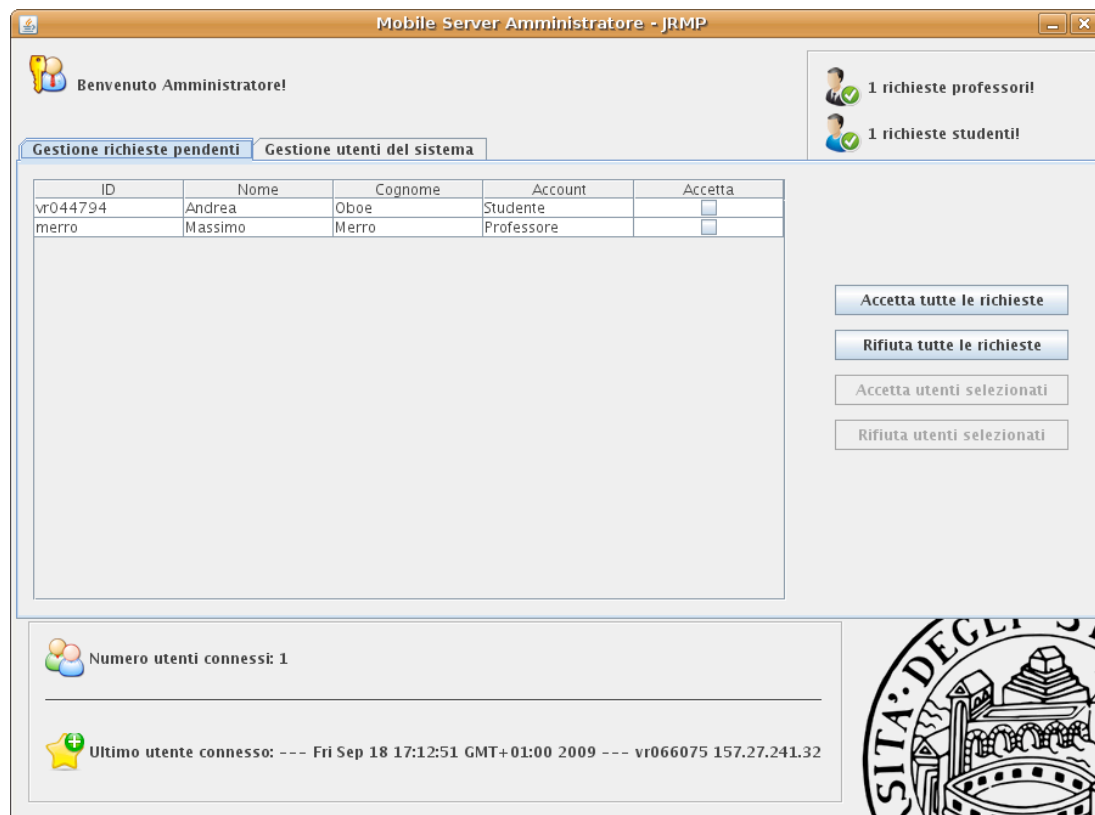
- autenticazione Professore/Studente: viene restituito un mobile agent corrispondente al tipo di utente appena loggato, che verrà lanciato non appena giunto a destinazione. Tale agente mobile avrà una referenza a Central Server e sempre in base al livello dell'utente, sarà possibile compiere una serie di operazioni che lasciamo al lettore scoprire.
- autenticazione Amministratore: una volta autenticato (si ricordi che per loggarsi come Amministratore sono necessarie le seguenti credenziali: user: admin, password: 1234 e tale login è permesso esclusivamente su client JRMP) il client riceve un mobile server che verrà esportato e lanciato non appena giunge a destinazione (si noti che tale mobile server non è esportato all'origine). L'avvenuta



Alberto Cordioli - vr066075

Andrea Oboe - vr044794

esportazione viene confermata dalla restituzione al sistema uniBCM delle caratteristiche del sistema ospitante il mobile server, come ad esempio il tipo di sistema operativo, l'ip, il paese di provenienza. Queste informazioni permettono di riconoscere in modo chiaro chi ha i privilegi di amministratore in quel dato momento, garantendo una tracciabilità in caso di eventuali accessi non autorizzati. Inoltre in questo caso la connessione presso Registration Server è criptata per mantenere un elevato grado di sicurezza.



Si noti come l'utente finale non sia a conoscenza della connessione al Proxy, in quanto avrà l'impressione di accedere direttamente ai rispettivi servizi invocati.

File di policy

Per rendere portabile al massimo il nostro sistema e per rendere una maggiore restrizione su cosa un client possa fare relativa ad una particolare macchina server-side, abbiamo deciso di creare due strutture base per i relativi file di policy per i client JRMP e IIOP, che prima del lancio del corrispondente client di Bootstrap, vengono modificate in accordo con la situazione correntemente in atto.

Riportiamo di seguito la struttura realizzata:

Policy JRMP

```
grant codeBase "_CLIENTPROT_CLIENTPATH" {
    // _COMMENT permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:_PORT", "connect, resolve";
    // _FILE permission java.io.FilePermission "_SERVERHOME/public_html/uniBCM/-", "read";
    // _HTTP permission java.io.FilePermission "http://_UNIBCMSEVER._DOMAIN:_PORT/-", "read";

    // per completezza: non necessari al fine del corretto funzionamento
    permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
    permission java.lang.RuntimePermission "exitVM";
    permission java.lang.RuntimePermission "accessClassInPackage.sun.swing";

    permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:1098", "connect, resolve";
    permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:2222", "connect, resolve";
    permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:10000-", "connect, resolve";
    permission java.net.SocketPermission "_UNIBCMCLIENT._DOMAIN:36000", "resolve";

    permission java.util.PropertyPermission "user.name", "read";
    permission java.util.PropertyPermission "user.country", "read";
    permission java.util.PropertyPermission "os.name", "read";
    permission java.util.PropertyPermission "os.version", "read";

    // permission per SSL
    permission java.util.PropertyPermission "javax.rmi.ssl.client.enabledCipherSuites", "read";
    permission java.util.PropertyPermission "javax.rmi.ssl.client.enabledCipherSuites", "read";
    permission java.util.PropertyPermission "javax.rmi.ssl.client.enabledProtocols", "read";
};

grant codeBase "_PROTOCOL_PATH/-" {
    // _COMMENT permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:_PORT", "connect, resolve";
    // _HTTP permission java.io.FilePermission "http://_UNIBCMSEVER._DOMAIN:_PORT/-", "read";

    // per completezza: non necessari al fine del corretto funzionamento
    permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
    permission java.lang.RuntimePermission "exitVM";
```

```

permission java.lang.RuntimePermission "accessClassInPackage.sun.swing";

permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:1098", "connect, resolve";
permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:2222", "connect, resolve";
permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:10000-", "connect, resolve";
permission java.net.SocketPermission "_UNIBCMCLIENT._DOMAIN:36000", "resolve";
permission java.net.SocketPermission "_UNIBCMSEVER.sci.univr.it:10000-", "accept";

permission java.util.PropertyPermission "user.name", "read";
permission java.util.PropertyPermission "user.country", "read";
permission java.util.PropertyPermission "os.name", "read";
permission java.util.PropertyPermission "os.version", "read";
//permission per SSL
permission java.util.PropertyPermission "javax.rmi.ssl.client.enabledCipherSuites", "read";
permission java.util.PropertyPermission "javax.rmi.ssl.client.enabledCipherSuites", "read";
permission java.util.PropertyPermission "javax.rmi.ssl.client.enabledProtocols", "read";
};

```

Policy IIOP

```

grant codeBase "_CLIENTPROT_CLIENTPATH" {
    //_COMMENT permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:_PORT", "connect, resolve";
    //_FILE permission java.io.FilePermission "_SERVERHOME/public_html/uniBCM/-", "read";
    //_HTTP permission java.io.FilePermission "http://_UNIBCMSEVER._DOMAIN:_PORT/-", "read";

    // per completezza: non necessari al fine del corretto funzionamento
    permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
    permission java.lang.RuntimePermission "exitVM";
    permission java.lang.RuntimePermission "accessClassInPackage.sun.swing";

    permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:5555", "connect, resolve";
    permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:10000-", "connect, resolve";
};

grant codeBase "_PROTOCOL_PATH/-" {
    //_COMMENT permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:_PORT", "connect, resolve";

    // per completezza: non necessari al fine del corretto funzionamento
    permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
    permission java.lang.RuntimePermission "exitVM";
    permission java.lang.RuntimePermission "accessClassInPackage.sun.swing";

    permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:5555", "connect, resolve";
    permission java.net.SocketPermission "_UNIBCMSEVER._DOMAIN:10000-", "connect, resolve";
};

```

Come è facile intuire, abbiamo parametrizzato i punti cruciali per avere un accurato file di policy per ciascun protocollo, limitando il più possibile ciò che ciascun client possa effettuare. Tale modifica al file di policy prima del lancio del client viene effettuata mediante

una funzione che può essere facilmente individuata nel file uniBCM lato client chiaramente. Oltre ai classici permessi studiati in modo approfondito a lezione, abbiamo avuto la necessità di inserire dei permessi riguardanti la gestione dello schermo, ovvero gli AWTPermission, in quanto il nostro progetto presenta un'interfacciamento grafico.

E' interessante notare come la struttura del file di policy per IIOP sia notevolmente ridotta in termini di permessi, in quanto ad uno studente viene limitato l'accesso in funzione a ciò che effettivamente può fare.

Un ulteriore vantaggio nel definire una parametrizzazione dei file di policy ci permette di utilizzare in modo semplice e diretto sia il protocollo file che il protocollo http, evitando così all'utente di dovere settare manualmente il protocollo su cui uniBCM opera.

Link utili

Riportiamo alcuni link e consultazioni che abbiamo ritenuto essenziali per la realizzazione del sistema uniBCM.

Lucidi ed esercizi per approfondire Java RMI - <http://profs.sci.univr.it/~merro/par.html>

Tutorial della Sun su Java RMI - <http://java.sun.com/docs/books/tutorial/rmi/index.html>

Forum della Sun specifico per RMI - <http://forums.sun.com/forum.jspa?forumID=58&start=0>

Java RMI con SSL - <http://www.j2ee.mc/j2se/1.5.0/docs/guide/rmi/socketfactory/SSLInfo.html>

E' inoltre vivamente consigliata la consultazione di questo libro:

Java RMI

William Grosso

Publisher: O'Reilly

First Edition October 2001

ISBN: 1-56592-452-5

Per eventuali chiarimenti specifici al progetto:

Alberto Cordioli : alberto.cordioli@gmail.com

Andrea Oboe : oboe.andrea@gmail.com