Angelo Antonio Salatino
Matr. 541532
Cdl in Ingegneria Informatica L3
Politecnico di Bari

Telegestione per le abitazioni

Progetto proposto per il sostenimento della prova pratica di Telematica II

Angelo Antonio Salatino

A.A. 2009/2010

Sommario

INTRODUZIONE

Leggendo velocemente la definizione di telegestione si apprende che:

"La telegestione consiste in una gestione totale di tutti gli aspetti riguardanti il funzionamento di un impianto di sicurezza (allarme, antincendio e televisivo a circuito chiuso) attraverso collegamenti telematici tra un Server ed un Client."

Nel nostro caso si opera su di una versione più estesa, nella quale è comunque possibile manipolare il sistema di allarme ma è anche possibile manipolare l'impianto di climatizzazione, impianto di irrigazione, tapparelle, impianto di illuminazione, forno e uso di scenari precedentemente creati.

Tutto ciò pone un enorme vantaggio per tutti coloro che sono spesso lontani dalle proprie abitazioni, i quali possono in remoto dal proprio ufficio o addirittura dal proprio palmare gestire con facilita gli impianti della propria dimora.

L'applicazione è stata sviluppata interamente in JAVA attraverso l'uso del software Eclipse e Net Beans, sfruttando il paradigma Client-Server per la comunicazione di più applicativi.

L'applicazione fa uso del protocollo SaReMH il quale attualmente non possiede nessuna RFC, ma sarà possibile apprezzarne i suoi contenuti qui di seguito.

Si procederà inizialmente con una spiegazione dettagliata di tale protocollo e conseguentemente verrà spiegato a grandi linee in cosa consiste il progetto.

Protocollo SaReMH

Il SaReMH, acronimo di Salatino's Remote Management for Homes, è un protocollo di livello 4(ovvero di livello applicativo) dello stack TCP/IP che si basa su una connessione TCP sulla porta 2781 permettendo cosi di interfacciare Client e Server.

Nel momento in cui la connessione è attiva, il client può dare inizio al dialogo con il server attraverso l'uso di opportuni comandi, esplicitati in seguito, al quale giungerà risposta non appena il server avrà elaborato la sua richiesta.

Comandi SaReMH

HELP	Comando relativo alla richiesta d'aiuto, fornisce all'utente le istruzioni necessarie
ALARM [flag]	Permette di gestire l'impianto di allarme
CLIME [flag] [num]	Attiva o disattiva i climatizzatori
IRR [flag] [num]	Gestisce impianto di irrigazione per il giardino
LAMP [flag] [num]	Gestisce l'impianto luce
OVEN [flag]	Gestisce l'accensione o lo spegnimento del forno
SCENE [flag] [num]	Attiva o disattiva scenari precedentemente creati
BLIND [flag] [num]	Alza o abbassa le tapparelle
STATE	Fornisce lo stato generale sull'abitazione
PASS [pwd]	Consente al client di loggarsi. Pwd è in MD5 codificata dal client.
QUIT	Consente di effettuare il logout
CLOSESRV	Disattiva il server per questioni di sicurezza

Risposte SaReMH

210	Comando accettato
213	Disconnesso
230	Password accettata
310	Comando ridondante:<motivazione>
400	Comando sconosciuto
410	Comando non accettato: <motivazione>
401	Sintassi del comando errata
430	Password non accettata
510	Errore del server, non è possibile gestire il comando
520	Errore sulla lettura dell'impianto

Comando HELP

HELP

Il comando HELP, qualora invocato, permette al client di ottenere una lista dove si specificano i comandi presenti all'interno del protocollo con i vari flag ed una breve descrizione. Questo comando risulta utile qualora uno o più comandi vengono dimenticati.

Esempio di risposta a tale comando:

```
Inserisci comando
HELP
SERVER>: HELP  :  Comando relativo alla richiesta d'aiuto, fornisce all'utente
le istruzioni necessarie
SERVER>: ALARM [ON,OFF,STATE] :     Permette di gestire l'impianto di allarme
SERVER>: CLIME [ACT,DEACT,STATE] [num]  :  Attiva o disattiva i climatizzatori
SERVER>: IRR [ON,OFF,STATE] [num]   :    Gestisce impianto di irrigazione per
il giardino
SERVER>: LAMP [ON,OFF,STATE] [num]   :    Gestisce l'impianto luce
SERVER>: OVEN [ON,OFF,STATE]  Gestisce l'accensione o lo spegnimento del forno
SERVER>: SCENE [SEE,STATE] [num]   :     Attiva e consulta gli scenari
precedentemente creati
SERVER>: BLIND [LOWER,RAISE,STATE] [num] :     Alza o abbassa le tapparelle
SERVER>: STATE  : Fornisce lo stato generale sull'abitazione
SERVER>: PASS [pwd]  :  Consente al client di loggarsi. Pwd è in MD5 codificata
dal client.
SERVER>: QUIT  :  Consente di effettuare il logout
SERVER>: CLOSESRV    :  Disattiva il server per questioni di sicurezza
```

Comando ALARM

ALARM [flag]

Il comando ALARM presenta tre flag:

- ON
- OFF
- STATE

Non appena il server si vede recapitare il comando ALARM ON esso attiverà l'allarme dell'abitazione. Nel caso in cui quest'ultimo fosse già attivo, il server specificherà che l'impianto era già attivo.

Es:

```
Inserisci comando
ALARM ON
SERVER>: 210 Comando accettato.
Inserisci comando
ALARM ON
SERVER>: 410 Comando non accettato :Sistema di allarme già attivo.
Inserisci comando
```

Per ALARM OFF accade il contrario di quanto specificato in ALARM ON, esso disattiva l'impianto di allarme e qualora fosse già inattivo il server si preoccuperà solo di avvisare che l'impianto è stato precedentemente disattivato non compiendo nessun'altra azione.

Quando viene invocato il comando ALARM STATE il client si vedrà recapitare un messaggio che fornisce lo stato circa l'impianto di allarme.

Es. di ALARM OFF e ALARM STATE:

```
Inserisci comando
ALARM OFF
SERVER>: 210 Comando accettato.
Inserisci comando
ALARM OFF
SERVER>: 410 Comando non accettato :Sistema di allarme già disattivato.
Inserisci comando
ALARM STATE
SERVER>:    Stato allarme: Disattivato
Inserisci comando
```

Comando CLIME

CLIME [flag] [num]

Come accade per il comando ALARM anche il comando CLIME presenta tre flag:

- ACT
- DEACT
- STATE

Il comando CLIME ACT consente di attivare il sistema di climatizzazione all'interno di una specifica stanza.

Il CLIME DEACT esegue l'operazione contraria al comando precedente, ovvero disattiva il climatizzatore o più climatizzatori precedentemente attivati nella stanza specificata, mentre il CLIME STATE fornisce al client una situazione dettagliata su tutto l'impianto di climatizzazione.

Il campo num è un identificativo della stanza a cui far riferimento ovvero in quale stanza attivare o disattivare il sistema di climatizzazione. Di conseguenza il campo num deve essere necessariamente specificato, altrimenti il comando non potrà essere accettato dal server perché a sua volta non saprebbe a quale stanza far riferimento. Qualora si invoca CLIME STATE non ha senso specificare alcun num.

Nel caso in cui non si conosce l'identificativo della stanza viene consigliato l'uso del comando CLIME STATE, perche quest'ultimo oltre a fornire lo stato attuale sull'impianto di climatizzazione fornisce informazioni circa l'associazione presente tra identificativo e stanza.

Es. di mini sessione:

```
Inserisci comando
CLIME STATE
SERVER>:    2 Cucina: Spento
```

```
SERVER>:    3 Bedroom: Spento
Inserisci comando
CLIME ACT 1
SERVER>: 410 Comando non accettato: Climatizzatore non presente.
Inserisci comando
CLIME ACT
SERVER>: 401 Sintassi del comando errata.
Inserisci comando
CLIME ACT 2
SERVER>: 210 Comando accettato.
Inserisci comando
CLIME STATE
SERVER>:    2 Cucina: Acceso
SERVER>:    3 Bedroom: Spento
Inserisci comando
CLIME DEACT 2
SERVER>: 210 Comando accettato.
Inserisci comando
CLIME DEACT 2
SERVER>: 310 Comando Ridondante: Climatizzatore già spento.
Inserisci comando
```

N.B.: Nella stanza 1 non essendoci installato alcun climatizzatore, nella lista ricevuta da CLIME STATE non verrà specificato nulla circa la stanza 1.

Comando IRR

IRR [flag] [num]

Il comando IRR permette con opportuni flag, di poter manipolare gli impianti di irrigazione attivando una zona, disattivandone un'altra e cosi via, consentendo di avere sempre verde il proprio giardino.

I flag presenti per questo comando sono:

- ON
- OFF
- STATE

In questo caso num non fa riferimento ad una stanza ma è utilizzato come identificativo di una determinata zona esterna(es: viale, porticato, retro, laterale etc) dove è previsto un impianto di irrigazione proprio e separato da altre zone in modo tale da poter gestire ogni zona a proprio piacere. Risulta ovvio che le zone possono essere gestite a proprio piacere includendo ad esempio tutte le zone in un unico impianto, oppure tanti impianti quante sono le zone. Il valore num va specificato sempre tranne il caso in cui si invoca IRR STATE.

Il comando IRR ON permette di attivare l'impianto previsto per la zona, avente identificativo specificato in coda al comando stesso, contrariamente si ottiene invocando il comando IRR OFF con lo stesso identificativo. IRR STATE invece permette di ottenere uno stato generale su tutto gli impianti di irrigazione mostrando quelli attivi e quelli disattivi, oltre a rendere nota l'associazione presente tra identificativo e zone presenti.

Es. di mini sessione:

```
Inserisci comando
IRR STATE
SERVER>:    1 Viale: Spento
SERVER>:    2 Porticato: Spento
Inserisci comando
IRR ON 2
SERVER>: 210 Comando accettato.
Inserisci comando
IRR ON 1
SERVER>: 210 Comando accettato.
Inserisci comando
IRR STATE
SERVER>:    1 Viale: Acceso
SERVER>:    2 Porticato: Acceso
Inserisci comando
IRR OFF 1
SERVER>: 210 Comando accettato.
Inserisci comando
IRR STATE
SERVER>:    1 Viale: Spento
SERVER>:    2 Porticato: Acceso
Inserisci comando
```

Comando LAMP

LAMP [flag] [num]

Il comando LAMP presenta, anch'esso, tre flag:

- ON
- OFF
- STATE

Qualora invocato, seguito da uno dei precedenti tre flag e da un num, che a sua volta funge da identificativo per la stanza, permette di accendere la luce in una determinata stanza(LAMP ON), spegnerla(LAMP OFF) o di conoscere lo stato generale su tutti gli impianti di illuminazione presenti nella varie stanze(LAMP STATE). E' necessario specificare il num della stanza altrimenti il server non saprebbe a quale stanza fare riferimento, tranne quando si invoca il comando LAMP STATE.

Es. di mini sessione:

```
Inserisci comando
LAMP STATE
SERVER>:    1 Bagno: Spento
SERVER>:    2 Cucina: Spento
SERVER>:    3 Bedroom: Spento
Inserisci comando
LAMP ON 1
SERVER>: 210 Comando accettato.
Inserisci comando
LAMP ON 3
SERVER>: 210 Comando accettato.
Inserisci comando
```

```
LAMP STATE
SERVER>:    1 Bagno: Acceso
SERVER>:    2 Cucina: Spento
SERVER>:    3 Bedroom: Acceso
Inserisci comando
LAMP OFF 2
SERVER>: 310 Comando Ridondante: Luce già spenta.
Inserisci comando
LAMP OFF 3
SERVER>: 210 Comando accettato.
Inserisci comando
LAMP STATE
SERVER>:    1 Bagno: Acceso
SERVER>:    2 Cucina: Spento
SERVER>:    3 Bedroom: Spento
Inserisci comando
```

Comando OVEN

OVEN [flag]

Il flag per questo comando sono:

- ON
- OFF
- STATE

Questo comando permette di manipolare il forno della propria abitazione, nel caso in cui risulta utile trovare al proprio rientro un cibo pronto.

Per accendere il forno ad una temperatura già prefissata si deve richiamare il comando OVEN ON, per spegnerlo OVEN OFF, mentre per controllare il suo attuale stato OVEN STATE.

Es. di mini sessione:

```
Inserisci comando
OVEN STATE
SERVER>:    Stato del forno: Spento
Inserisci comando
OVEN ON
SERVER>: 210 Comando accettato.
Inserisci comando
OVEN STATE
SERVER>:    Stato del forno: Acceso
Inserisci comando
OVEN OFF
SERVER>: 210 Comando accettato.
Inserisci comando
```

Comando SCENE

SCENE [flag] [num]

Il comando SCENE permette di attivare gli scenari precedentemente caricati, si spiegherà in seguito cosa si intende con tale affermazione. Uno scenario, nel nostro caso, è un insieme di comandi appartenenti al medesimo protocollo, che permettono di ottenere un determinato effetto a seguito dell'utilizzo di un unico comando. Ad esempio si potrebbe creare lo scenario "Esci" il quale si preoccupa di spegnere tutte le luci, abbassare le tapparelle e attivare il sistema di allarme, in molti casi risulta vantaggioso l'utilizzo di questi scenari.

I flag concessi per questo comando sono:

- null ->""
- SEE
- STATE

Num è l'identificativo dello scenario utile a tracciarlo univocamente.

Per attivare uno scenario si deve semplicemente invocare il comando SCENE seguito da un identificativo, es SCENE 1. Per controllare la lista degli scenari si può, invece, invocare il comando SCENE STATE.

Tuttavia è possibile al contempo controllare o quanto meno visionare il contenuto di uno scenario ovvero tutta la serie di comandi appartenenti, allo stesso protocollo, salvati all'interno dello scenario stesso invocando il comando SCENE SEE seguito da un num identificativo.

Es. di una mini sessione:

```
Inserisci comando
SCENE STATE
SERVER>:    1 Esci
SERVER>:    2 Irrigazione
Inserisci comando
SCENE SEE 1
SERVER>: ALARM ON
SERVER>: LAMP OFF 1
SERVER>: LAMP OFF 2
SERVER>: LAMP OFF 3
SERVER>: Termine lista.
Inserisci comando
SCENE 1
SERVER>: 210 Comando accettato.
SERVER>: 310 Comando Ridondante: Luce già spenta.
SERVER>: 210 Comando accettato.
SERVER>: 210 Comando accettato.
SERVER>: Scenario attivato.
Inserisci comando
SCENE SEE 2
SERVER>: IRR ON 1
SERVER>: IRR ON 2
SERVER>: Termine lista.
Inserisci comando
```

Comando BLIND

BLIND [flag] [num]

Il comando BLIND permette, se presenti, di abbassare o alzare le tapparelle della propria abitazione.

I flag sono:

- LOWER
- RAISE
- STATE

Quando si invoca il comando BLIND LOWER seguito da un num, che identifica la stanza, il sistema abbassa le tapparelle presenti nella stessa, con BLIND RAISE sempre seguito dal num, comunque necessario in tutti i casi, alza le tapparelle, con BLIND STATE non specificando il num si può controllare lo stato di tutte le tapparelle e conoscere l'associazione presente tra num e stanza.

Qualora nel risultato del comando BLIND STATE non viene stampato qualche identificativo allora si ha che per tale stanza non esiste una tapparella gestibile dal sistema.

Es. di mini sessione:

```
Inserisci comando
BLIND STATE
SERVER>:    2 Cucina: Abbassata
SERVER>:    3 Bedroom: Abbassata
Inserisci comando
BLIND RAISE 1
SERVER>: 410 Comando non accettato: Tapparella non presente.
Inserisci comando
BLIND RAISE 2
SERVER>: 210 Comando accettato.
Inserisci comando
BLIND STATE
SERVER>:    2 Cucina: Alzata
SERVER>:    3 Bedroom: Abbassata
Inserisci comando
BLIND LOWER 2
SERVER>: 210 Comando accettato.
Inserisci comando
BLIND STATE
SERVER>:    2 Cucina: Abbassata
SERVER>:    3 Bedroom: Abbassata
Inserisci comando
```

Comando STATE

STATE

Il comando STATE permette di ottenere uno stato generale su tutta l'abitazione. Il server quando riceve questo comando interroga tutti gli impianti e fornisce informazioni circa gli stati di tutti gli impianti.

Es:

```
Inserisci comando
STATE
SERVER>: Allarme:
SERVER>:    Stato allarme: Disattivato
```

```
SERVER>: Climatizzatori:
SERVER>:    2 Cucina: Spento
SERVER>:    3 Bedroom: Spento
SERVER>: Irrigazione:
SERVER>:    1 Viale: Spento
SERVER>:    2 Porticato: Acceso
SERVER>: Tapparelle:
SERVER>:    2 Cucina: Abbassata
SERVER>:    3 Bedroom: Abbassata
SERVER>: Lampade:
SERVER>:    1 Bagno: Acceso
SERVER>:    2 Cucina: Spento
SERVER>:    3 Bedroom: Spento
SERVER>: Forno:
SERVER>:    Stato del forno: Spento
SERVER>: Scenari:
SERVER>:    Scene 1: Attivo
Inserisci comando
```

Comando PASS

PASS [pwd]

Il comando PASS è il primo comando che il client invia al server. Non contiene flag, mentre il campo pwd è formato da una serie di caratteri che formano la password di sistema, codificata attraverso l'algoritmo MD5. Questo comando serve al server per capire se il client con cui ha appena effettuato la connessione ha il permesso per accedere alla gestione del sistema. La codifica in MD5 serve ai fini della sicurezza , cosi facendo si cerca di impedire che qualcuno possa entrare in possesso dei permessi per accedere al sistema.

Tuttavia è possibile inserire la password incorretta per sole tre volte prima che la connessione tra client e server venga chiusa.

Comando QUIT

QUIT

Il comando QUIT quando viene invocato permette al client di effettuare il log out e di chiudere cosi la connessione. Tuttavia il server resta attivo in modo tale che altri client possono connettersi al server in istanti successivi alla ricezione del comando.

Comando CLOSESRV

CLOSESRV

Il comando CLOSESRV è un comando di sicurezza, utile quando in determinate situazioni si ha la necessità di spegnere il server ed impedirgli cosi di accettare nuove connessioni.

Si potrebbe pensare a delle anomalie sul sistema, modifiche apportate al sistema da terzi indesiderati e quindi procedere con la chiusura del server. Sarà possibile ripristinarlo in loco.

Esempio di sessione SaReMH

Lato Client:

```
Programma di Telegestione per le Abitazioni.
Inserisci indirizzo ip, oppure l'hostname del server.
localhost
Connessione stabilita con: localhost/127.0.0.1:2718
Inserisci la password per connettersi al server:
CIAO
SERVER>: 230 Password accettata
Login Effettuato
Inserisci comando
ALARM ON
SERVER>: 410 Comando non accettato :Sistema di allarme già attivo.
Inserisci comando
ALARM OFF
SERVER>: 210 Comando accettato.
Inserisci comando
ALARM STATE
SERVER>:    Stato allarme: Disattivato
Inserisci comando
QUIT
SERVER>: Logout effettauto.
Logout Effettuato
Connessione chiusa
```

Lato Server

```
Server in ascolto sulla porta: 2718
/*************************************************************/
Connessione stabilita con: /127.0.0.1:3808
Nuovo client accettato.
Attivo il thread che si occuppa della connessione stabilita con:
/127.0.0.1:3808
Questo thread è identificato come: /127.0.0.1:2718
/*************************************************************/
CLIENT /127.0.0.1:3808>: PWD accettata, quindi è attivo il dialogo.
Connessione in chiusura con /127.0.0.1:2718
```

Remote Management for Homes

Panoramica

Il progetto nel complesso che prende il nome di Remote Management for Homes comprende ben tre sottopregetti ed è sviluppato interamente in JAVA. Tuttavia presenta al suo interno ben 4 package e 17 classi organizzate attraverso una struttura ad albero:

Sottoprogetti	Package	Classi	Sviluppato in
RemoteManagementforHomes	package 1	Alarm.java Blind.java Clime.java Help.java Irrigate.java Lamp.java MD5.java Oven.java RemoteManagementforHomes.java Scene.java ServeServer.java StateServer.java genericMethod.java	J2SE
	package 2	contentFile.java setup.java	
ClientRMFH	package3	ClientRMFH.java	J2SE
PalmClient	package4	PalmClient.java	J2ME

I primi due progetti sono stati sviluppati con il software Eclipse, contrariamente dal terzo dove si è usato Net Beans per via delle sue funzionalità avanzate nella programmazione J2ME.

Il package 1 nel suo insieme contiene tutte le classi utili al server per servire i comandi inviati dal client, ogni comando ha una propria classe come ad esempio la classe Alarm, Blind, Clime etc.

RemoteManagementforHomes è il server vero è proprio, il quale si mette in ascolto sulla porta 2718 ed accetta le connessioni in entrata.

ServeServer è una classe che implementa l'interfaccia Runnable in modo tale da offrire un server multithread.

MD5 è un server secondario che permette di effettuare le codifiche di password in md5, utili ai client J2ME che non contengono librerie a supporto di tale codifica.

La classe genericMethod funge da jolly per il package 1, perche contiene una sola copia di tutti quei metodi che sarebbero ripetuti nelle varie classi, si chiarirà meglio in seguito questo concetto.

Il package 2 è un package di servizio il quale contiene una classe di setup atta alla configurazione del server, esempio l'impianto di allarme, l'impianto luci etc. Contiene, anche, una classe contentFile che a sua volta contiene i modi e i metodi atti all'utilizzo della maggior parte dei file che utilizza il software. Per quanto riguarda i file in seguito verrà fatta un'ampia panoramica sui contenuti.

Il secondo progetto è il client che meglio si interfaccia all'applicazione del primo progetto, formato da un solo package e da una sola classe.

Il terzo progetto è un client per dispositivi mobili, programmato attraverso lo standard J2ME per dispositivi palmari e cellulari.

Package1

RemoteManagementeforHomes.java

Questa classe è definita la classe madre del server in quanto appena si esegue l'applicazione il metodo main instanzia un oggetto di questa classe, lo inizializza e ne chiama il metodo start() il quale a sua volta mette in ascolto il server sulla porta 2718 ed avvia un nuovo thread per ogni connessione accettata.

Il metodo start(), tuttavia, attiva e mette in ascolto un ulteriore server sulla porta 7777 utile ai dispositivi mobili per ottenere l'md5 della propria password. L'uso di questo server secondario verrà chiarito in seguito.

ServeServer.java

Questa classe implementa l'interfaccia Runnable permettendo cosi di implementare il sistema multithread.

Presenta un metodo run(), che si occupa della connessione con il client. All'interno della stessa classe vi sono dei metodi che permettono di effettuare il login(login() e code()) e dei metodi che permettono il dialogo tra il client e server(serverun() e commands()).

Ci sono anche dei metodi di servizio come logout(), shutdown() che rispettivamente permettono all'utente di sloggarsi e spegnere il server per questioni di sicurezza.

E' bene chiarire che l'operazione di login viene effettuata con la massima sicurezza in quanto il comando PWD ricevuto dal client contiene una stringa in md5. La password, quindi, nella rete viaggia crittografata.

Quando il server la riceve, effettua un confronto con l'md5 della password del sistema, se esse sono uguali allora si può dare il consenso al client circa l'utilizzo del server.

Alarm.java

Questa classe si preoccupa di servire l'utente qualora viene invocato un comando relativo all'impianto di allarme. Viene instanziato un oggetto di questa classe nella ServeServer ed in occasione del evento descritto viene chiamato il metodo serve() della classe Alarm che si preoccupa di in interpretare il comando e di distribuire il compito di esecuzione ai vari metodi secondari:

- alarmOn()
- alarmOff()
- alarmState()

Blind.java

Come quanto accade per la classe Alarm si ha per la classe Blind, viene instanziato un oggetto di questa classe in ServeServer e di conseguenza viene chiamato il metodo serve() dell'oggetto appena creato nel momento in cui il metodo commands() della ServeServer, intercetta un comando relativo all'impianto delle tapparelle. Il metodo serve() interpreta il comando e delega l'esecuzione ai metodi:

- blindR()
- blindL()
- blindState()

Clime.java

La classe clime si comporta allo stesso modo delle precedenti, con l'unica differenza che serve i comandi relativi al sistema di climatizzazione. Quando viene chiamata la serve() esso interpreta il comando ricevuto e delega i seguenti metodi ad espletare quanto richiesto:

- climeAct()
- climeDeact()
- climeState()

Help.java

La classe Help, dal nome stesso, fa intendere che il suo uso e relativo al comando HELP. Quando il metodo commands() della ServeServer, legge in ingesso tale comando chiama il metodo serve() dell'oggetto di questa classe precedentemente instanziato. Questo metodo non fa altro che leggere da un file di testo(HELP.txt) l'help dei comandi e replicarlo sullo stream di uscita.

Irrigate.java

La classe irrigate si occupa dell'impianto di irrigazione e contiene quattro metodi:

- serve()
- irrOn()
- irrOff()
- irrState()

dove il primo si occupa di servire il comando e delegare alle tre successive il compito di espletare la richiesta.

Lamp.java

Questa classe si occupa dell'impianto luci. Qualora sullo stream di ingresso venisse letto un comando con radice LAMP viene chiamato subito il metodo serve() dell'oggetto Lamp precedentemente instanziato che successivamente delegherà il compito di completare l'operazione ad uno dei tre seguenti metodi:

- lampOn()
- lampOff()
- lampState()

MD5.java

Questa è una classe speciale, essa estende la classe Thread. In questo modo si riesce ad ottenere un altro server in ascolto su un'altra porta indipendente da quello principale. Cosi facendo si ha la possibilità che questo server secondario accetti connessioni in ingresso ed effettui la codifica in md5 del contenuto presente sull'input stream riversandolo poi sull'output stream.

Oven.java

La classe Oven permette di gestire l'accensione o lo spegnimento di un forno. Con un oggetto di tipo Oven già istanziato nella classe ServeServer, verrà chiamato il metodo serve() di tale oggetto nel caso in cui il server si vedrà recapitare un comando la cui radice è OVEN.

Scene.java

La classe Scene è una classe particolare perche all'interno di essa presenta comunque quattro metodi come tutte le altre ma eseguono operazioni differenti. I metodi sono:

- serve()
- sceneAct()
- sceneSee()
- sceneState()

Nelle altre classi al posto della sceneSee() avremmo trovato sceneDeact() un metodo che permette di disattivare uno scenario. Tuttavia, non avrebbe senso disattivare un scenario e ritornare cosi alla configurazione precedente all'attivazione dello scenario stesso. Se proprio risulta necessaria questa operazione è più conveniente creare un'ulteriore scenario complementare a quello che si è creato in precedenza.

Es:

scenario: Esci

```
ALARM ON
LAMP OFF 1
LAMP OFF 2
LAMP OFF 3
```

Qualora si voglia disattivare questo scenario i comandi invocati sarebbero:

```
ALARM OFF
LAMP ON 1
LAMP ON 2
LAMP ON 3
```

Si apprende subito che il primo comando avrebbe senso, ma i successivi tre non avrebbero senso in quanto risulta inutile riaccendere le tre luci.

Il metodo sceneAct() serve ad attivare lo scenario, mentre sceneSee() serve a visualizzare il contenuto dello scenario, mentre lo sceneState() serve a visualizzare gli scenari salavati.

StateServer.java

La classe StateServer è utile qualora venisse rilevato il comando STATE, fornendo al client lo stato generale dell'intero sistema. Essa consta di un solo metodo(serve()) il quale chiama i vari metodi state() delle altre classi dopo averne instanzito gli oggetti.

genericMethod.java

La classe genericMethod contiene tutti i metodi static, comuni e soprattutto utili alle classi del package 1. Si sta parlando di metodi che effettuano scritture su file, letture da file, verificano l'esistenza di un intero all'interno di una stringa, restituiscono l'intero inserito in una stringa.

Package2

contentFile.java

E' definita la classe chiave del progetto, perche è usata come classe all'interno degli arraylist.

Questa classe contiene tre variabili:

- **private** String room

- **private int** num
- **private int** value

con le quale si tiene traccia di tutto il sistema all'interno dei file. Si vedrà in seguito come queste tre varibili vengono gestite all'interno dei file e che significato hanno.

Tuttavia, questa classe possiede tanti metodi get e tanti metodi set quante sono le variabili, in modo da poter accedere alla singola variabile dall'esterno.

setup.java

La classe setup è una sorta di procedura, essa deve essere avviata separatamente dal server, permettendo cosi di eseguire il setup del sistema, ovvero configurare le stanza, impianto luci, tapparelle, allarme, forno e molto altro. Permette anche di creare scenari, cancellarli, cambiare la password del sistema e molto altro.

Package3

ClientRMFH.java

Questa è la classe adibita al client. Nel momento in cui lo si avvia egli chiede l'indirizzo a cui connettersi e quando la connessione si è stabilità viene richiesta la password(comunque impostata nel setup) per poter effettuare il login.

Quando il client è abilitato al dialogo, si è in grado di gestire la propria abitazione digitando tutti i comandi previsti dal protocollo.

Package4

PalmClient.java

La classe PalmClient è stata progettata per consentire anche ai cellulari/palmari di svolgere le stesse funzioni che svolge il client ClientRMFH. Questa classe tuttavia è stata progetta in J2ME ovvero una tecnologia che consente di sviluppare software o utilities per cellulari. E' formato essenzialmente da una schermata principale nella quale viene chiesto l'indirizzo e la password e se questi ultimi sono corretti si ottiene velocemente il dialogo con il server. Successivamente appare una schermata nella quale vi è una textbox dove vi è possibile inserire il comando, inviarlo e ottenere risposta.

File

Pwd.txt

Pwd.txt come da estensione è un file di testo nella quale è contenuta in chiaro la password che un client deve inserire per accedere al server. L'autenticazione avviene qualora la password letto dall'input stream è uguale a quella conservata in questo file.

Questo file può essere acceduto in scrittura dalla classe setup qualora si voglia impostare o modificare la password mentre può essere acceduto in lettura dal server stesso quando deve verificare la password ricevuta dal client.

Logfile.txt

Il Logfile è un file sequenziale nella quale vengono conservate tutte le azioni dei vari client verso il server, comandi ricevuti, risposte inviate, etc in modo da tener conto, cosi, di tutte le operazioni effettuate. Tutto ciò risulta vantaggioso se si pensa all'insorgere di un problema nel server, per via di qualche anomalia nel sistema, il che semplificherebbe all'amministratore la ricerca, l'identificazione e quindi la risoluzione del problema.

Tuttavia questo file potrebbe essere reso disponibile per produzione di statistiche sui comandi più usati e quindi migliorare la qualità del progetto.

Es:

```
Connessione stabilita con: /127.0.0.1:4326 -> Fri Aug 27 16:29:42 CEST 2010
        CLIENT /127.0.0.1:4326>: Comando ricevuto: ALARM STATE
        CLIENT /127.0.0.1:4326>: Risposta inviata: --Stato allarme: Attivato
        CLIENT /127.0.0.1:4326>: Comando ricevuto: ALARM OFF
        CLIENT /127.0.0.1:4326>: Risposta inviata: 210 Comando accettato.
        CLIENT /127.0.0.1:4326>: Comando ricevuto: ALARM STATE
        CLIENT /127.0.0.1:4326>: Risposta inviata: --Stato allarme: Disattivato
        CLIENT /127.0.0.1:4326>: Comando ricevuto: QUIT
Disconnesso da: /127.0.0.1:4326 -> Fri Aug 27 16:30:15 CEST 2010
```

Help.txt

Il file Help è un file non soggetto a scritture ma solo a letture. Esso contiene una serie di righe, le quali vengono lette dal server ed inviate al client qualora esso stesso avesse invocato il comando HELP.

Contenuto del file:

HELP: Comando relativo alla richiesta d'aiuto, fornisce all'utente le istruzioni necessarie
ALARM [ON,OFF,STATE]: Permette di gestire l'impianto di allarme
CLIME [ACT,DEACT,STATE] [num]: Attiva o disattiva i climatizzatori
IRR [ON,OFF,STATE] [num]: Gestisce impianto di irrigazione per il giardino
LAMP [ON,OFF,STATE] [num]: Gestisce l'impianto luce
OVEN [ON,OFF,STATE]: Gestisce l'accensione o lo spegnimento del forno
SCENE [SEE,STATE] [num]: Attiva o disattiva scenari precedentemente creati
BLIND [LOWER,RAISE,STATE] [num]: Alza o abbassa le tapparelle
STATE: Fornisce lo stato generale sull'abitazione
QUIT: Consente di effettuare il logout
CLOSESRV: Disattiva il server per questioni di sicurezza

Alarm.dat

Come da estensione rappresenta un file binario, esso contiene una sola variabile intera. Quest'ultima può assumere tre valori:

- -1: Impianto non gestibile
- 0: Impianto gestibile ma attualmente disattivato
- 1: Impianto gestibile e attivo.

Condizionatori.dat

Il file condizionatori contiene un arraylist di dimensione pari al numero delle stanze presenti nel sistema. L'arraylist a sua volta è di tipo contentFile:

- **private** String room
- **private int** num
- **private int** value

In definitiva il contenuto del file è verosimilmente il seguente:

room1	num1	value1
room2	num2	value2
.	.	.
.	.	.
roomn	numn	Valuen

In questo arraylist la variabile room assume il nome della stanza(es: Bagno, Cucina) il num indentifica il numero di impianto mentre il value indica lo stato dell'impianto(0->spento, 1->acceso, -1 climatizzatore inesistente). Il valore di num oltre a rappresentare un identificativo per la stanza esso è il numero da usare nel comando quando si vuole accedere a tale impianto.

Es: CLIME ACT 1, dove si accede all'impianto di climatizzazione della room1.

Irrigazione.dat

Il file Irrigazione contiene all'interno di esso lo stesso arraylist ma di dimensioni diverse. Questa differenza è dovuta essenzialmente al fatto che la variabile room tiene traccia del nome dell'impianto(es. Frontale, Laterale), il num indentifica il numero di impianto mentre il value indica lo stato dell'impianto(0->spento, 1->acceso).

Luci.dat

Il file Luci contiene lo stesso arraylist del file Climatizzatori con l'unica differenza che nel file climatizzatori è possibile ritrovare alcune delle n variabili value=-1 perche è possibile che non vi siano climatizzatori, mentre per le luci non accade. E' impensabile che vi sia una stanza senza luce. Di default viene settato il valore value=0 e scritto sul file. Per quanto riguarda la variabile room e la variabile num i due file sono uguali, in definitiva si differenziano per l'unica colonna della variabile value perche essa presenta contenuti informativi differenti.

Oven.dat

Il file Oven assomiglia molto al file Alarm, nel quale è possibile trovare una sola variabile intera che può assumere tre valori:

- -1: Impianto non gestibile
- 0: Impianto gestibile ma attualmente disattivato
- 1: Impianto gestibile e attivo.

Tapparelle.dat

Il file Tapparelle contiene anch'esso un arraylist molto simile a quello contenuto nel file Luci e nel file Climatizzatori. Se si rappresenta il contenuto del file attraverso una tabella simile a quella vista in precedenza abbiamo:

room1	num1	value1
room2	num2	value2
. .	. .	. .
roomn	numn	Valuen

il file tapparelle con i gli altri due file su citati si differenzia per la sola colonna della variabile value.

Le prime due colonne conservano lo stesso contenuto informativo ovvero:

- room->Stanza
- num-> id della stanza

mentre la variabile value può assumere tre valori:

- -1: Impianto non gestibile
- 0: Impianto gestibile ma attualmente disattivato
- 1: Impianto gestibile e attivo.

Stanze.dat

Il file stanze è la matrice di tutto il sistema, esso è il primo file che viene compilato esso conserva al suo interno l'associazione:

Stanza->id stanza

tutto questo grazie ad un arraylist simile a quello usato nel file Luci etc. nella quale pero vengono impiegate solo le prime due colonne, la variabile value non contiene alcun significato informativo e viene impostato a 0 di default.

Es:

room1	1	0
room2	2	0
. .	. .	. .
roomn	n	0

In definitiva tutti gli altri file all'interno di essi hanno una copia del file Stanze. Esso funge anche da prova dell'esistenza di un setup. Quando si esegue l'applicazione del setup del sistema ed esiste già un file stanze il software chiede se si vuole riusare il vecchio file ed eseguire quindi un setup condizionato anziché un setup obbligatorio.

List.dat

Il file List serve a tener traccia degli scenari presenti nel sistema. All'interno di esso è presente una lista ci oggetti contentFile, come accade per il resto dei file visti fin'ora.

La variabile room indica il nome dello scenario(es: Esci), la variabile num funge da identificativo per lo scenario, esso è immutabile nel tempo fino all'eliminazione dello scenario.

La variabile value è una variabile di stato e può assumere due valore significativi:

- -1: inesistenza dello scenario perche è stato cancellato

- 0: esistenza dello scenario

Non esiste un valore che individua lo stato di attivo perche non serve tener traccia di uno scenario attivo in quanto poi non sarà possibile disattivarlo.

Scene[num]

Il file Scene serve a rappresentare uno scenario, il valore num serve ad indicare a quale dei tanti scenari fa riferimento.

All'interno di ognuno di essi è possibile trovare un arraylist di String contenente a sua volta un listato di comandi da far eseguire al server nel momento in cui viene attivato lo scenario.

Principali problematiche riscontrate

Le conoscenze acquisite durante il corso sono state più che sufficienti a mettere in piedi questo progetto nella parte relativa al J2SE in quanto all'interno di quest'ultima sono stati utilizzati gli stessi metodi visti durante le lezioni ed esercitazioni. Per quanto riguarda la parte del J2ME sono nate alcune complicazioni, dovute principalmente all'interfacciamento con un linguaggio di programmazione simile ma con metodi differenti. Un esempio banale potrebbe essere quello dell'apertura di un socket:

J2SE: Socket link=new Socket(servAddress,port);

J2ME: SocketConnection sc = (SocketConnection)Connector.open("socket://"+hostname+":"+PORT);

Un ulteriore complicazione è stata riscontrata all'apertura degli stream e della trasmissione di dati sugli stessi. In J2SE esistono readLine() e la println() che ci permettono di leggere e scrivere delle linee o stringhe sullo stream, la stessa cosa non vale per il J2ME in quanto sugli stream previsti da questo standard ci si possono trovare solo dei byte[]. Con tutta una serie di prove si è riusciti a trovare un compromesso tra le trasmissioni e le ricezioni tra i vari client e il server. Con esattezza dagli stream si leggono degli interi con succesivo casting a char e messi in uno stringbuffer, mentre le scritture avvengono sia per byte[] che per stringhe perche esse sono del tutto compatibili.

Un ulteriore problema si è manifestato quando si è tentati di instaurare un dialogo tra un client J2ME e un server J2SE in quanto sono sorti due problemi ovvero "il valore null restituito da una readLine()" e il "mancato rilascio del controllo da parte di una read()". In teoria per lo standard J2ME nel momento in cui si esegue una scrittura sullo stream, lo si deve chiudere altrimenti chi ascolta dall'altra parte, sul server, non rilascia il controllo e impedisce il regolare svolgimento delle azioni. Lo stesso vale per la fase di lettura in quanto la read() è sempre in ascolto finche non rileva lo stream chiuso. Arrivati a questo punto non si poteva permettere che venisse trasmesso un solo messaggio e di conseguenza fosse chiuso lo stream in quanto il progetto si basava su una connessione persistente e su un molteplice scambio di dati tra client e server.

Le prime prove consistevano nel evitare di chiudere gli stream e cercare qualche metodo che rilasciasse il controllo per permettere cosi al server di elaborare quanto ricevuto, i tantativi fallivano in quanto tutti i metodi esistenti rilasciavano il controllo non appena fosse chiuso lo stream.

Di li si è pensato di lasciare aperti gli stream sul server mentre gli stream sul client venivano chiusi e successivamente riaperti qualora si volesse inserire un ulteriore comando, ma le letture successive alle prime da parte del server restituivano un null. C'era qualche problema di sincronizzazione.

Successivamente si è pensato di chiudere gli stream anche sul server e riaprirli successivamente alla chiusura, ma la situazione non cambiava.

Il tentativo che risolvesse tutti i problemi in un colpo solo è stato quello di usare delle lettere chiave come # e * in quanto posti al termine del dato da trasmettere con opportuni controlli permettevano di terminare le trasmissioni evitando di chiudere gli stream.

Conclusioni

In conclusione ci troviamo davanti ad un ottimo progetto che ha come scopo principale la simulazione di quanto avviene in un piccolo impianto di domotica, l'apprendimento del linguaggio di programmazione e l'uso del paradigma Client-Server. Inoltre si ha che il software è facilmente estendibile nel caso in cui il protocollo venisse aggiornato con nuovi comandi. Tenendo conto che ogni tipo di comando possiede una propria classe, quindi per estendere il software è necessario aggiungere altre classi che siano in grado di servire i nuovi comandi.

Si potrebbe pensare allo sviluppo di questo software e renderlo disponibile sul mercato se al posto dei file venissero usate delle primitive per gestire delle FPGA oppure dei Relè USB in modo tale da inserirli in veri e propri impianti civili o industriali cosi da rendere disponibile un effetto fisico a seguito dell'uso di un comando.

Ringraziamenti

Particolari ringraziamenti vanno al mio amico Michele che mi ha fornito la giusta ispirazione. Altrettanti vanno agli utenti del forum "Oracle & Sun" i quali hanno saputo instradarmi alla soluzione dei problemi previsti con l'interfacciamento tra un server J2SE e un client J2ME.

Inoltre ringrazio anticipatamente tutti coloro che intendono contribuire all'opera, migliorandola.

Sitografia

Network Programming with J2ME Wireless Devices:

http://www.wirelessdevnet.com/channels/java/features/j2me_http.phtml

J2ME Tutorial:

http://www.roseindia.net/j2me/

J2ME tutorials, J2ME basics, J2ME samples and example code:

http://www.java-samples.com/j2me/