Estensione di un servizo di messaggistica per telefonia mobile (per una società di agenti TuCSoN)

User Guide

di
Mattia Bargellini

CAPITOLO 1

Getting Started

1.1 Contenuto del package e Installazione

Per installare il sistema occorre semplicemente scompattare il file `sw.zip`. Si otterrà una cartella il cui contenuto è quello mostrato in Fig.1.1.

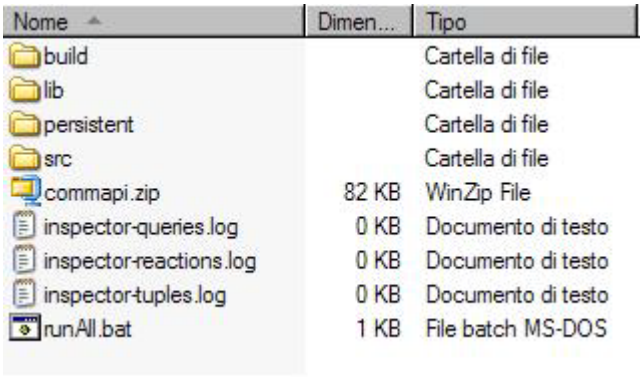

Fig.1.1: Contenuto del package.

Nella cartella `src` sono contenuti i seguenti file sorgenti:

- `SMS_Sender.java` : rappresenta l'agente che è incaricato di inviare gli sms;
- `AlarmClock.java` : rappresenta l'agente che simula la "sveglia";
- `StartService.java` e `StartAlarm.java` : classi che lanciano rispettivamente gli agenti `SMS_Sender` e `AlarmClock` e che contengono le reazioni che verranno settate nel centro di tuple;
- `Checker.java` : rappresenta l'agente che viene lanciato dalla reazione per il controllo dell'invio multiplo.

Il file `runAll.bat` serve per lanciare l'infrastruttura TuCSoN e gli agenti sopraelencati.

La cartella `lib` contiene i pacchetti `tucson.jar` (versione 1.4) e `jSMSEngine.jar` necessari rispettivamente per lanciare un nodo TuCSoN e per interfacciare il cellulare che funge da gateway con l'agente `SMS_Sender`.

La cartella `build` ha due sottodirectory:

- `alice\tucsonx\sms` in cui vi sono i bytecode dei sorgenti;
- `org\jsmsengine` in cui vi sono i sorgenti e i bytecode della libreria `jsmsengine`

La cartella `doc` contiene la documentazione nel formato javadoc.

Infine il file `commapi.zip` contiene le librerie per la comunicazione tra cellulare e pc sulla porta com. La guida all'installazione del pacchetto commapi[1] consiglia di:

- Copiare `win32com.dll` nella directory `<JDK>\bin`;
- Copiare `comm.jar` nella directory `<JDK>\lib`;
- Copiare `javax.comm.properties` nella directory `<JDK>\lib`;
- Aggiungere `comm.jar` al classpath;
- Copiare infine il file `activation.jar` nella cartella `<JDK>\lib`

[1] Per ulteriori informazioni si rimanda alla guida stessa e alla guida della libreria `jSMSEngine` sul sito **http://jsmsengine.sourceforge.net/**

1.2 Avvio del sistema

Per lanciare il sistema, se si è sotto ambiente Windows, è sufficiente cliccare sul file `runAll.bat`, altrimenti lo si può lanciare tramite shell scrivendo i seguenti comandi:

`java -cp lib\tucson.jar alice.tucson.runtime.Node` : per lanciare il nodo tucson

`java -cp lib\tucson.jar alice.tucson.ide.Inspector` : per lanciare il tool inspector che serve per controllare cosa accade nel centro di tuple

`java -cp lib\tucson.jar alice.tucson.ide.CLIAgent` : per lanciare un agente di supporto (agente che verrà usato in seguito per eseguire delle prove) [2]

`java –cp build;lib\tucson.jar;C:\j2sdk1.4.2_04\lib\comm.jar;C:\j2sdk1.4.2_04\lib\activation.jar alice.tucsonx.sms.StartService` : per lanciare l'agente SMS_Sender

`java -cp build;lib\tucson.jar alice.tucsonx.sms.StartAlarm` : per lanciare l'agente AlarmClock

[2] Per ulteriori chiarimenti si rimanda alla guida di TuCSoN scaricabile al sito **http://www-lia.deis.unibo.it/Research/TuCSoN/**

CAPITOLO 2

2.1 Guida all'uso

Una volta lanciato il sistema, seguendo la procedura descritta nel paragrafo 1.2, sarà possibile usare il sistema in due modi: manualmente, facendo delle prove tramite l'agente fittizio CLIAgent (fornito da TuCSoN) oppure costruendo un proprio agente che sfrutti il servizio, rispettando i protocolli di interazione descritti nel file *notes.pdf*.

2.2 Esempi

Di seguito vengono proposti alcuni esempi, tramite l'uso di `CLIAgent`, per chiarire meglio il funzionamento del sistema.
Si suppone che la prima di fase di ogni esempio sia quello di richiesta del ticket al centro di tuple, come mostrato nelle figure sottostanti.

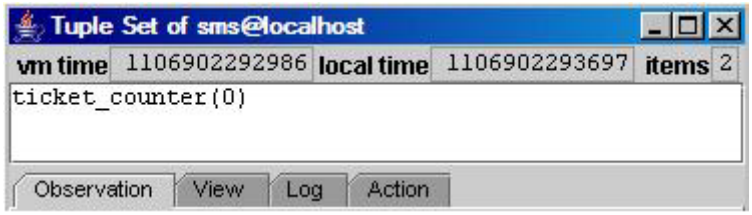

Fig.2.1: Contenuto del tuple centre sms all'avvio del sistema.

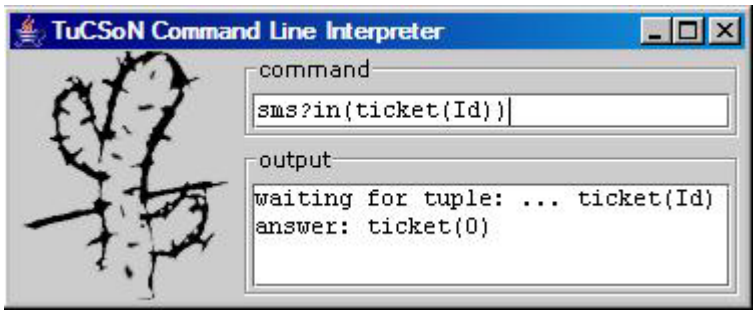

Fig.2.2: Richiesta del ticket da parte di CLIAgent.

Fig.2.3: Contenuto del tuple centre sms dopo la richiesta del ticket.

Esempio 1: invio diretto di un sms

L'esempio più semplice da cui partire è quello di richiesta diretta all'agente `SMS_Sender` tramite il template:

```
sms_to_send(Id,Recipient,Text)
```

dove `Id` è un numero intero, `Recipient` (numero di cellulare del destinatario) e `Text` (testo del messaggio) sono stringhe da inserire tra **“”**.
Nelle figure sono mostrate la tupla inserita nel centro di tuple da `CLIAgent`, la risposta di `SMS_Sender` e il contenuto del centro di tuple[3].

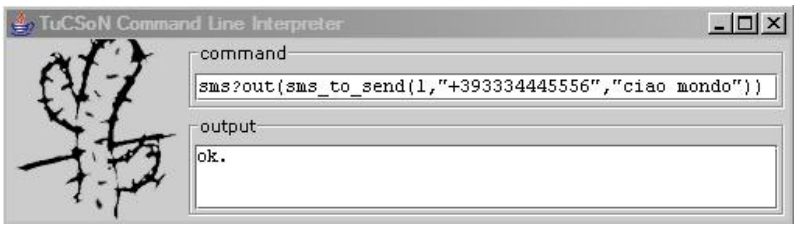

Fig.2.4: Richiesta invio “semplice”.

[3] Per la lista dei possibili status del servizio vedere l'Appendice.

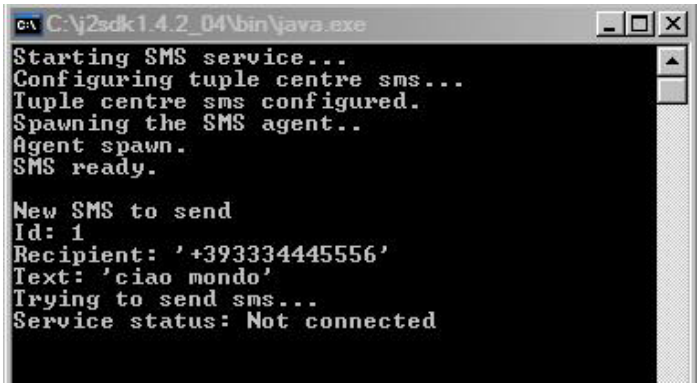

Fig.2.5: Output video di SMS_Sender.

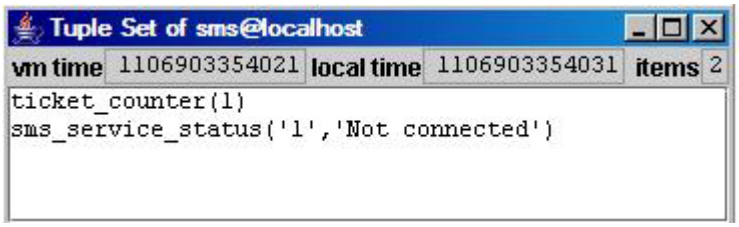

Fig.2.6: Contenuto del centro di tuple.

Esempio 2: uso di AlarmClock

L'agente `AlarmClock` può essere usato anche indipendentemente dall'invio di un messaggio, poiché fornisce un servizio di sveglia basandosi solo su data e ora. Per accedere direttamente ad `AlarmClock` occorre usare il di protocollo:

```
when(Id,Date,Time)
```

dove `Id` è un numero intero, `Date` è la data di invio nel formato "dd.mm.yyyy" (es: "20.01.2005") e `Time` è l'orario nel formato "hh.mm" 24 ore (es: "17.05").
All'ora X AlarmClock emetterà un segnale di sveglia tramite la tupla:

```
wake_up(Id)
```

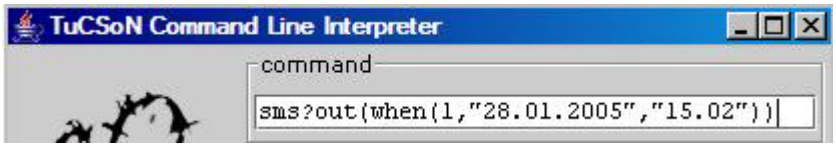

Fig.2.7: Richiesta di "sveglia".

Fig.2.8: Output video di AlarmClock

Fig.2.9: Contenuto centro di tuple

Esempio 3: invio multiplo

Questo esempio chiarisce il funzionamento del sistema nel caso in cui venga richiesto un servizio di invio multiplo a più destinatari e/o con testo maggiore della lunghezza consentita[4]. Si hanno i seguenti casi possibili:

1. Inserimento di una tupla contenente N destinatari e un testo minore di 8 caratteri: verranno prodotte N tuple con lo stesso messaggio nel formato dell'esempio 1;

[4] Per semplicità è stato impostato un valore massimo per il testo uguale a 8 invece dei 160 caratteri canonici.

2. Inserimento di una tupla contenente un unico destinatario e un testo maggiore di 8 caratteri: verranno prodotte M tuple nel formato dell'esempio 1 (dove M = lunghezza testo/8) ognuna contenente l'unico destinatario e M-esimo blocco di testo;
3. Inserimento di una tupla contenente N destinatari e un testo maggiore di 8 caratteri: verranno prodotte N*M tuple nel formato dell'esempio 1, ognuna contenente N-esimo destinatario e M-esimo blocco di testo.

Il template da usare è:

```
sms(Id,[Recipient],Text)
```

dove `[Recipient]` è una lista di numeri nel formato internazionale(ad esempio ["+393331112224","+394445556667"]).
Di seguito sono mostrati il tipo di tupla inserita nel centro di tuple da `CLIAgent`, la reazione all'evento, lo splitting della tupla e il contenuto del centro di tuple.

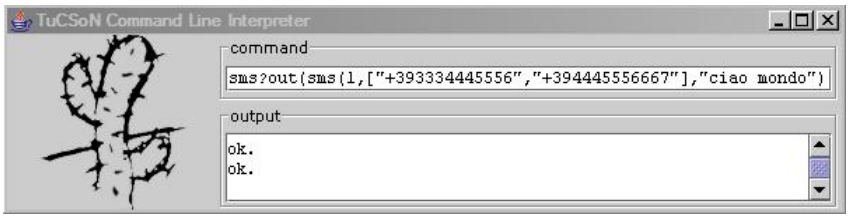

Fig.2.10: Richiesta di invio multiplo.

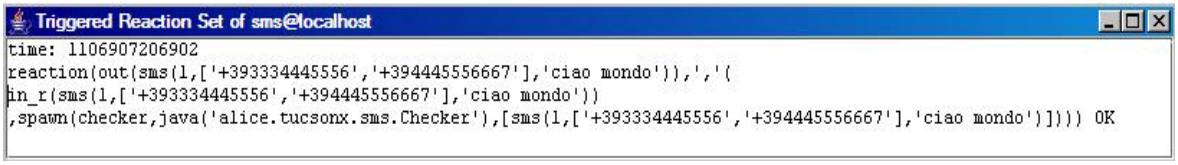

Fig.2.11: Reazione.

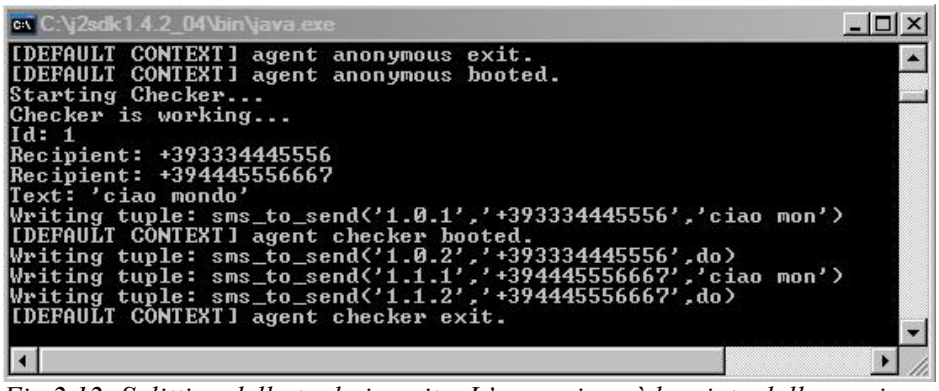

Fig.2.12: Splitting della tupla inserita. L'operazione è lanciata dalla reazione.

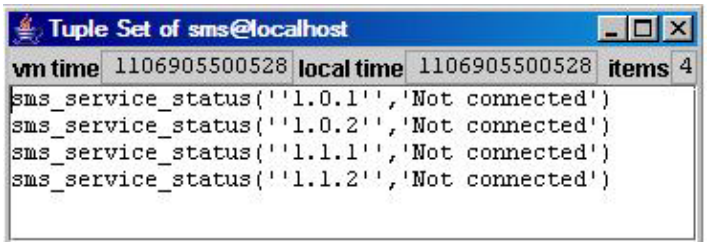

Fig.2.13: Contenuto del centro di tuple dopo l'elaborazione di SMS_Sender.

Esempio 4: invio differito

Questo esempio chiarisce il funzionamento del sistema nel caso in cui venga richiesto un servizio di invio differito. Il template da usare è:

```
sms(Id,[Recipient],Text,Date,Time)
```

dove `[Recipient]` è una lista di numeri come nell'esempio 3 (in questo esempio verrà usato un solo destinatario), `Date` è la data di invio e `Time` è l'orario nel formato descritto nell'esempio 2.
Una reazione estrae le informazioni di data e ora per l'agente `AlarmClock`, il quale allo scattare dell'ora X inserisce un tupla di `wake_up(Id)` nel tuple centre. A questo punto `SMS_Sender` cerca di inviare il messaggio.

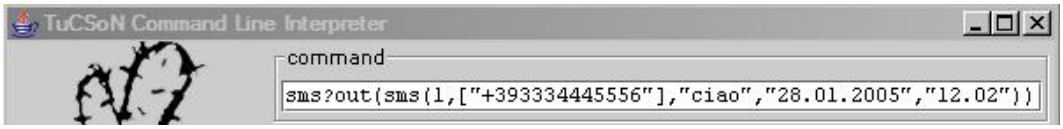

Fig.2.14: Richiesta di invio differito.

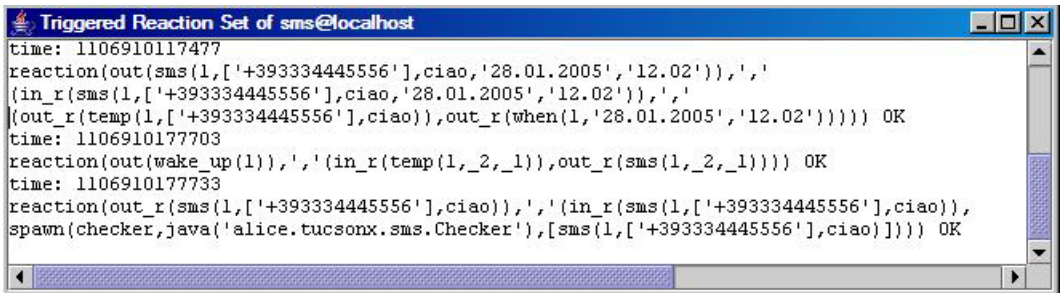

Fig.2.15: Reazioni.

Fig.2.16: Contenuto finale del centro di tuple.

Esempio 5: invio multiplo e differito

Questo esempio è l'insieme degli esempi 3 e 4 visti sopra..

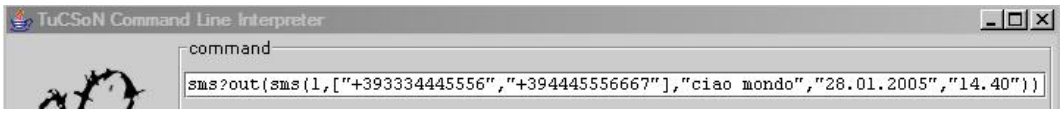

Fig.2.17: Richiesta di invio multiplo e differito.

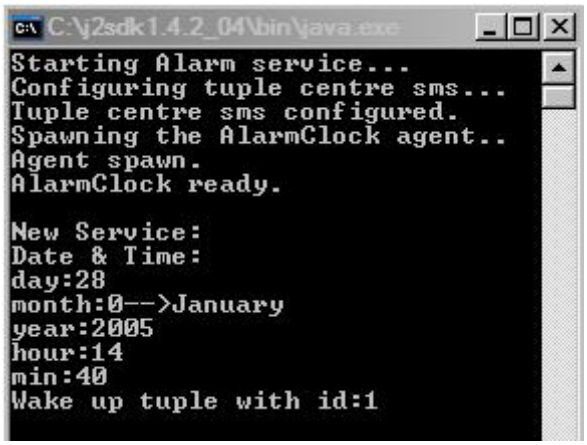

Fig.2.18: Output video di AlarmClock.

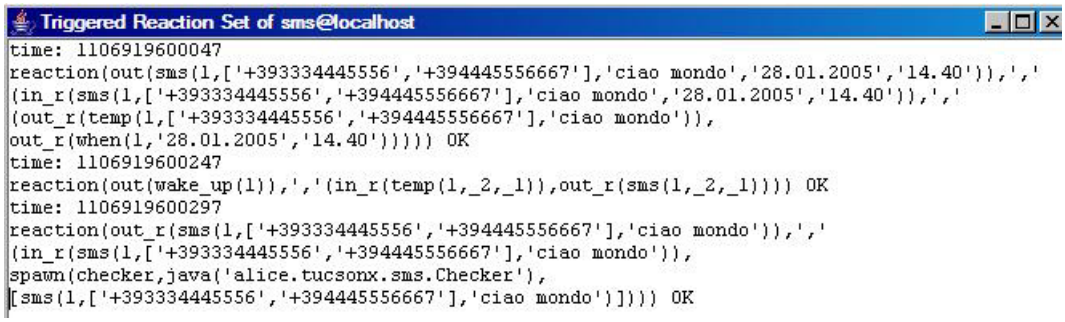

Fig.2.19: Reazioni.

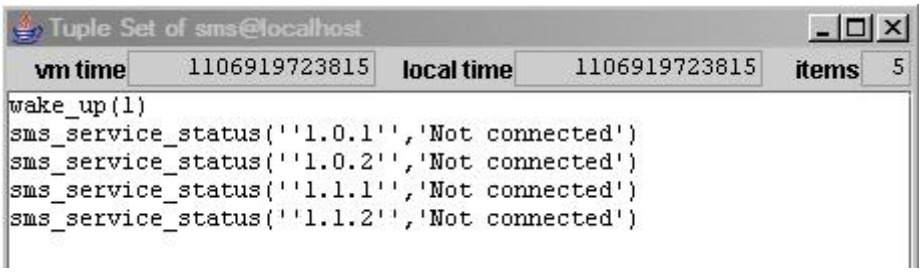

Fig.2.20: Contenuto finale del centro di tuple.

Appendice A

Elenco errori forniti da `SMS_Sender`:

Errori riportati dalla libreria jSMSEngine		Stato riportato da SMS_Sender
ERR_CHARSET_HEX_NOT_SUPPORTED	-21	"Charset hex not supported"
ERR_COMM_NOT_SUPPORTED	-20	"Device does not support ASCII or PDU mode"
ERR_GENERIC_ERROR	-1	"Generic Error"
ERR_INVALID_DIR	-100	"Invalid dir"
ERR_MESSAGE_NOT_FOUND	-30	"Message not found"
ERR_NO_CACHE	-101	"No cache"
ERR_NOT_CONNECTED	-11	"Not connected"
ERR_NOT_INITIALIZED	-10	"Not initialized"
ERR_NOT_SUPPORTED	-9999	"The specified operation is not supported by jSMSEngine API."
ERR_OK	0	"OK"
ERR_SEND_FAILED	-40	"Send-message operation failed"
ERR_SIM_PIN_ERROR	-102	"the PIN given is invalid"