

2017

# ConvergentApps

ELABORATO DI APPLICAZIONI TELEMATICHE  
MARIANO PASCARELLA - M63 000098



# INTRODUZIONE

L'elaborato prodotto ha l'obiettivo di mostrare l'uso delle Sip Servlets, con una particolare attenzione rivolta alla composizione di applicazioni e alla convergenza. Quest'ultimi due punti rappresentano le principali caratteristiche che differenziano le Sip Servlets dalle Servlets Http.

L'elaborato fornisce anche un Tutorial su come impostare la macchina per realizzare ed eseguire applicazioni convergenti, in particolare verrà mostrato come configurare Apache Tomcat in modo da gestire lo stack protocollare Sip. Per questo obiettivo si farà uso di RestComm Sip Servlet, un framework che estende le caratteristiche di Tomcat, permettendo a quest'ultimo di gestire e contenere le Sip Servlets, che combinato allo stack Http permette la realizzazione di applicazioni convergenti.

Per mostrare le nuove caratteristiche introdotte dall'uso delle Sip Servlet, sono state realizzate tre applicazioni, ognuna delle quali implementa un determinato servizio finito ed indipendente. Allo stesso tempo, sfruttando il pattern composition, le applicazioni possono essere combinate per formare delle Pipe. La tecnica di composizione permette da un lato il riuso del software, dall'altra la realizzazione di catene di applicazioni, fondamentale in tutti quegli scenari dove bisogna adoperare dei filtri prima di procedere nell'elaborazione.

## RESTCOMM

RestComm Sip Servlet è un progetto gestito da TeleStax e rappresenta una piattaforma open con la quale è possibile sviluppare e mandare in esecuzione servizi Sip e applicazioni convergenti.



Fig 1: RestComm

RestComm Sip Servlet può essere considerata come un'estensione da integrare all'interno di un Application Server container, nel nostro caso Apache Tomcat, che non rappresenta un vero e proprio Application Server, ma si tratta di un semplice contenitore di Servlet.

La piattaforma RestComm aggiunge funzionalità al contenitore permettendo a quest'ultimo di comunicare con il mondo esterno anche attraverso il protocollo SIP oltre che il classico Http. Tomcat da solo non è capace di gestire le Sip Servlets, tantomeno le applicazioni convergenti, mentre attraverso il contenitore fornito da Restcomm, al cui interno è presente lo stack protocollare Sip,

Tomcat può gestire i messaggi Sip e la convergenza(Http/Sip). RestComm oltre al contenitore fornisce un'implementazione delle specifiche Sip Servlet v1.1 dello standard JSR 289.

## SIP SERVLET

Le Sip Servlet API 1.1 sono definite nel JSR 289 e descrivono uno standard per implementare servizi e applicazioni SIP. RestComm Sip Servlet fornisce un'implementazione di questa specifica che può essere comparata alle Servlet Http. Le Servlet Sip importano la maggior parte delle specifiche Http, ma aggiungono anche nuovi metodi per poter gestire i dialoghi SIP che sono molto più evoluti della controparte Http.

## ARCHITETTURA

Verrà descritto di seguito la nostra architettura e le applicazioni realizzate.

### Il Sip Container

Al centro della nostra architettura, troviamo il server Tomcat al cui interno è presente un Container capace di contenere sia le Servlets Sip che le Servlets Http.

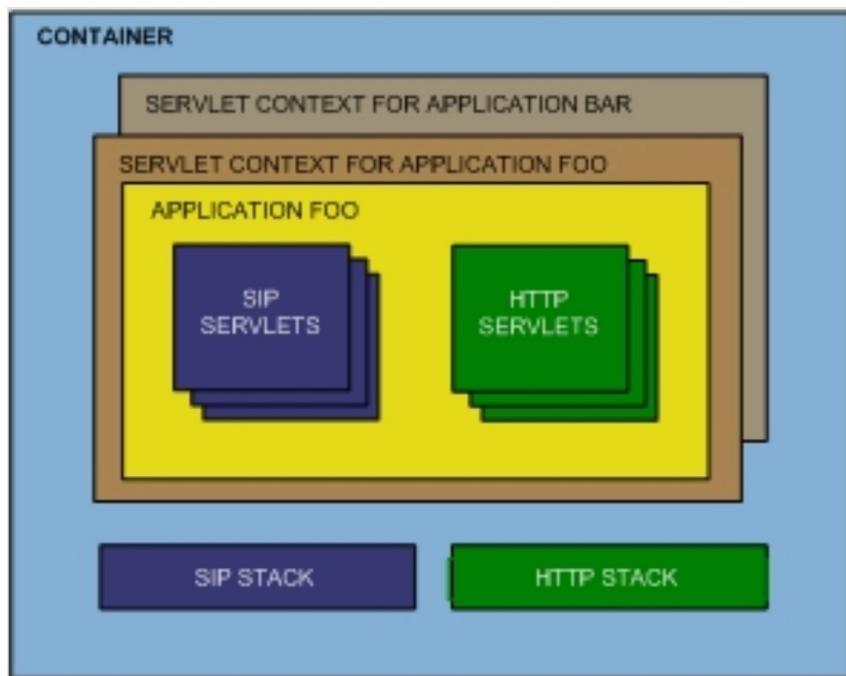


Fig. 2: Container convergente

Come è possibile vedere dalla figura, il nostro contenitore gestisce in maniera separata le diverse applicazioni e per ognuna di essa fornisce un contesto che ingloba entrambe le tipologie di Servlet.

Le Servlet di una stessa applicazione possono così condividere lo stesso contesto, dove risulta estremamente facile scambiarsi informazioni. Il container sfrutta poi i due diversi stack protocollari per comunicare all'esterno.

Oltre al contesto le Servlet di una stessa applicazione condividono anche la SipApplicationSession, o più semplicemente l'ApplicationSession.

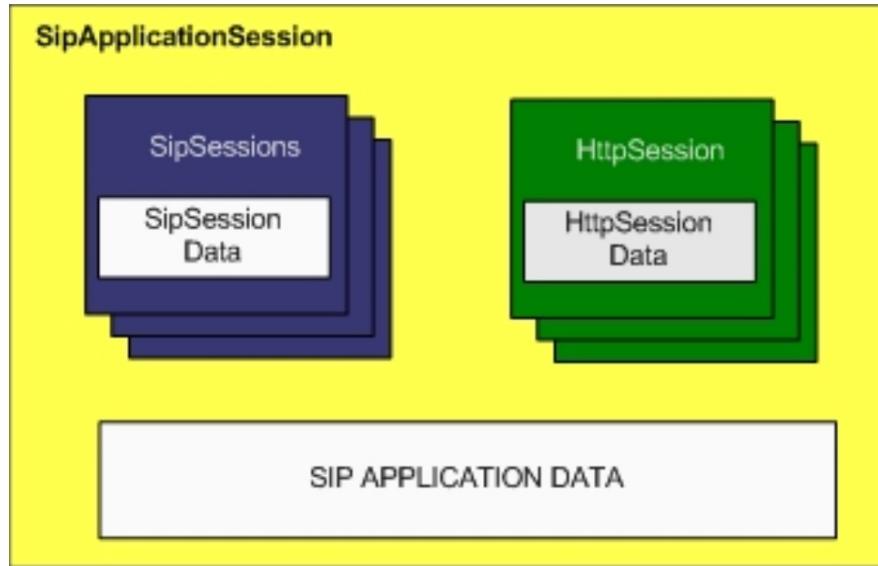


Fig. 3: Application Session

Un'applicazione può essere coinvolta in diverse interazioni con l'esterno, sia attraverso il protocollo Sip che quello Http. Le sessioni che fanno riferimento alla stessa applicazione sono mantenute nella SipApplicationSession, dove possiamo trovare sessioni di diversa tipologia.

Le Sip Session sono simili alle Http Session, ma mentre quest'ultime hanno un dialogo semplice, caratterizzato dalla sola interazione richiesta/risposta, le Sip session devono gestire un concetto di sessione più articolato. Infatti, possiamo ricevere messaggi che fanno riferimento ad un dialogo già presente nella nostra sipApplicationSession. Mentre la stessa cosa per http può essere implementata solo con i cookies e coinvolgendo interazioni req/res differenti.

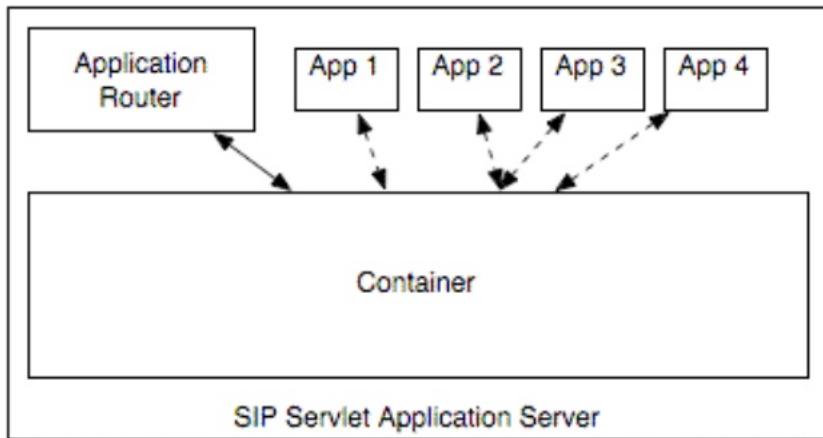
Attraverso il Context e l'Application Session possiamo implementare le nostre applicazioni convergenti, potendo comunicare con l'esterno attraverso diversi protocolli e gestendo servlet diverse nello stesso contesto. Le applicazioni convergenti infatti, sono applicazioni che richiedono l'uso di entrambi i protocolli per realizzare un determinato servizio.

## DAR

Il nostro Server convergente deve anche gestire la composizione. Potendo disporre di diverse applicazioni, ognuna delle quali realizza un determinato servizio, nasce la necessità di invocare diverse applicazioni in modo da ottenere un servizio più avanzato attraverso la composizione modulare che è possibile solo se le applicazioni vengono invocate nel giusto ordine

L'Application router è chiamato dal container per selezionare la Sip servlet application da invocare per gestire una richiesta di Initial, ovvero una richiesta per creare un nuovo dialogo sip. Questo procedimento ha luogo ogni volta che il container riceve una richiesta di tipo Initial, anche se questa è generata da una servlet interna al container stesso. Al termine dell'inoltro si viene a realizzare un

application path, che è fondamentale per le successive richieste e risposte collegate al messaggio di Initial ricevuto e quindi al dialogo creato.



L'Application router implementa l'interfaccia `SipApplicationRouter` che specifica le API tra il container e l'Application Router. Esso incorpora la logica da usare per scegliere la giusta applicazione da invocare.

Nel nostro caso l'Application router è implementato da Restcomm Sip Servlet mediante il Default Application router, una classe che legge il file DAR per eseguire le diverse scelte. Infatti, la logica nel nostro caso è rappresentata da un semplice file di configurazione, che è collegato al container Tomcat mediante descrittori nel file `server.xml`.

Il container è responsabile di passare le informazioni necessarie al DAR in seguito ad una Initial request, in quanto non ci sono interazioni tra le Application servlet e il DAR.

Il DAR ricopre un ruolo fondamentale nella composizione di applicazioni, infatti è il DAR a rendere possibile la realizzazione dell'Application path, quest'ultimo rappresenta la catena di applicazioni che devono essere attraversate dai Subsequent requests e le Response relative allo stesso dialogo.

## Composizione

Per realizzare la composizione di applicazioni e quindi realizzare una Pipe, sfruttiamo il DAR. Questo ci permette di definire quale applicazioni invocare e in che ordine.

Un'applicazione che riceve un Initial request (ad esempio `Invite`), può comportarsi da UAC, ad esempio facendo il proxy della richiesta. In questo modo si inoltra la Initial request al container che nuovamente invoca il DAR per sapere dove inviare la richiesta. A questo punto sono possibili due scenari:

1. Nel DAR file non ci sono applicazioni interessate alla richiesta, pertanto la richiesta è inoltrata all'esterno dal container.
2. Ci sono altre applicazioni interessate alla richiesta, quindi la richiesta è inoltrata all'applicazione seguendo un preciso ordine. Ordine mantenuto da uno stato interno al container che fa riferimento alla stessa richiesta.

In entrambi i casi questo procedimento termina quando un'applicazione si comporta da UAS. Questo evento termina la catena e produce un path di applicazioni, lo stesso path che sarà usato dai messaggi di response e dalle subsequent requests relativi allo stesso dialogo.

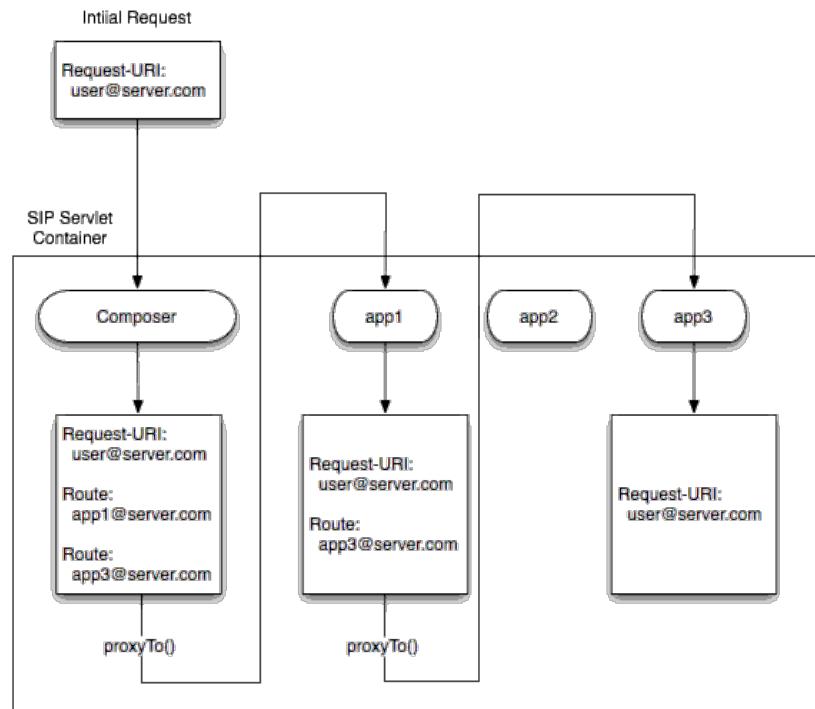


Fig. 5: Composizione

Il DAR fa uso di uno stato interno relativo alle precedenti transazioni, questo permette di invocare le diverse applicazioni interessate allo stesso evento ma in ordine diverso. È comunque possibile resettare questo stato creando una nuova richiesta o settando opportunamente il DAR

## IMPLEMENTAZIONE

Sono state implementate quattro applicazioni convergenti, tre di esse possono essere usate indipendentemente o essere composte per formare un'applicazione più evoluta. Di seguito sono mostrate le singole applicazioni, successivamente vedremo come comporre una Pipe.

### CallBlocking

CallBlocking è un'applicazione convergente che riceve una richiesta di Invite e controlla se il mittente è presente nella lista dei contatti da bloccare. Nel caso sia presente, l'applicazione risponde al mittente con un messaggio di Forbidden, altrimenti la chiamata procede normalmente.

L'applicazione è composta da una servlet sip e una servlet http. Attraverso una pagina web un eventuale amministratore può impostare una lista di contatti da bloccare. La lista è fatta in modo che ogni utente può avere la propria black list. Quando il server riceve una richiesta di Invite consulta il DAR e successivamente inoltra la richiesta alla servlet CallBlcocking. Quest'ultima, accedendo al

contesto del container, controlla la lista che è stata precedentemente creata dall'interazione http tra l'amministratore e la servlet Http. Nel caso il contatto è presente nella lista il server risponde con un messaggio di Forbidden, nel caso opposto inoltra la richiesta attraverso un Proxy. In questo modo, così come avviene per il B2Bua, la chiamata non è interrotta ma continua in modo tale che il container, attraverso l'uso del DAR, possa invocare la successiva applicazione. Cosa che non avviene se si crea una nuova richiesta, che corrisponde a resettare lo stato del DAR relativo alla precedente richiesta di Initial.

```
//Recupero la black list dal contesto dell'applicazione nel container. Precedentemente gestita dalla servlet http
HashMap<String,List<String>> blackList = (HashMap<String,List<String>>) getServletContext().getAttribute("BlockedListMap");
//Accedo alla lista solo se è stata creata, anche se potrebbe essere vuota
if(blackList != null){
    //Recupero a partire dalla destinazione dell'Invite, la black list associata
    blockedFromUri = blackList.get(toUri);
    //Se è presente una entry controllo se l'origine dell'invite è presente nella black list
    if(blockedFromUri != null)
        bloccato = blockedFromUri.contains(fromUri);
}

//Se la fromUri risulta bloccata invio un messaggio di Forbidden al mittente
if(bloccato){
    logger.info(fromUri + " è stata bloccata !");
    SipServletResponse sipServletResponse = request.createResponse(SipServletResponse.SC_FORBIDDEN);
    sipServletResponse.send();
    System.out.println("-----");
    System.out.println("[CallBlockingServlet]: doInvite()");
    System.out.println("-----");
    System.out.println(fromUri + " è stata bloccata !");
    System.out.println("-----");
} else {
    logger.info(fromUri + " Non è stata bloccata, inoltra la richiesta");

    System.out.println("-----");
    System.out.println("[CallBlockingServlet]: doInvite()");
    System.out.println("-----");
    System.out.println(fromUri + " Non è stata bloccata, inoltra attraverso il B2Bhelper la richiesta");
    System.out.println("-----");
}

request.getProxy().proxyTo(request.getRequestURI());
```

Fig 6: codice callblocking

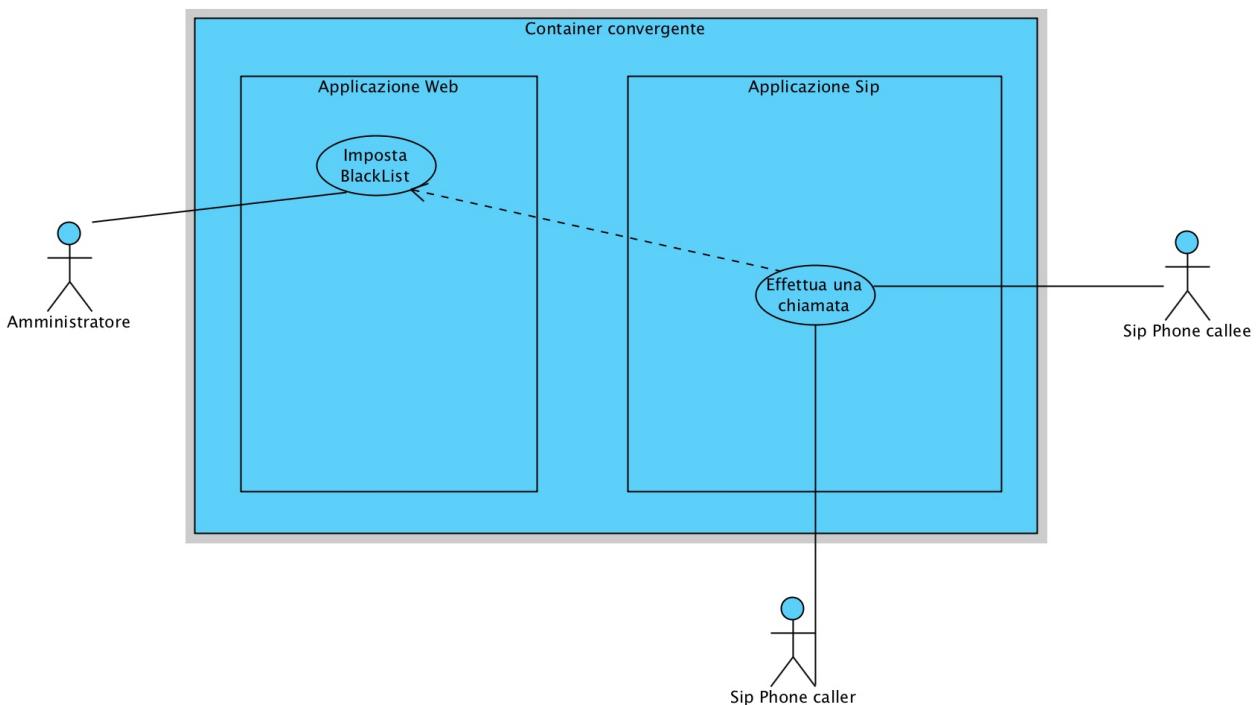


Fig. 7: Caso d'uso CallBlocking

Dalla figura è possibile osservare come la richiesta di chiamata dipenda dall'interazione che l'amministratore ha con la pagina web.

Black List

Uri:

Uri blocked:

[Pulisci la tua Black List](#)

[Ritorna alla Home](#)

Fig. 8: blacklist.html

Il seguente diagramma mette in mostra una possibile interazione con l'applicazione convergente.

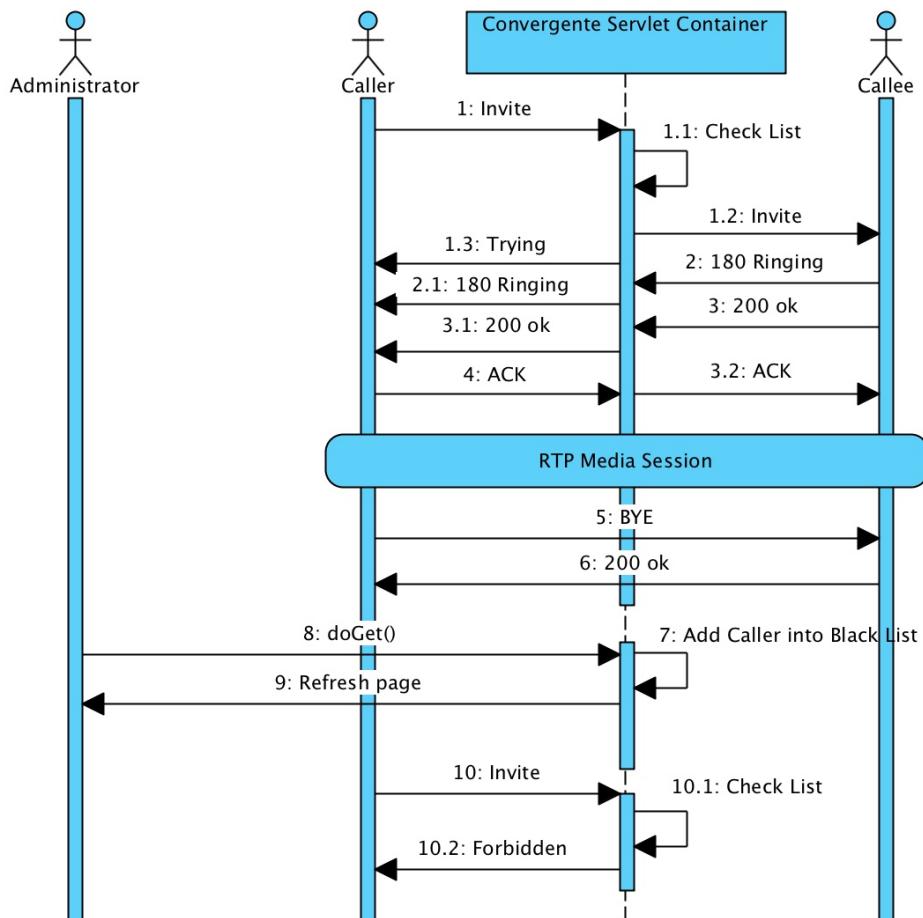


Fig. 9: Sequence diagram callBlocking

Inizialmente non è presente nessuna black list, quindi la chiamata procede come una comune Sip Call. Invece, dopo aver inserito il caller nella black list, eseguendo una nuova chiamata si ottiene in messaggio di messaggio di forbidden, dovuto dall'impossibilità di contattare il callee essendo stati inseriti nella sua black list.

## Location

Location è un'applicazione convergente che permette di individuare la locazione di un callee. Anche in questo caso abbiamo una possibile interazione con l'amministratore che attraverso una pagina web può impostare una lista di possibili locazioni relative ad una Uri Sip specificata.

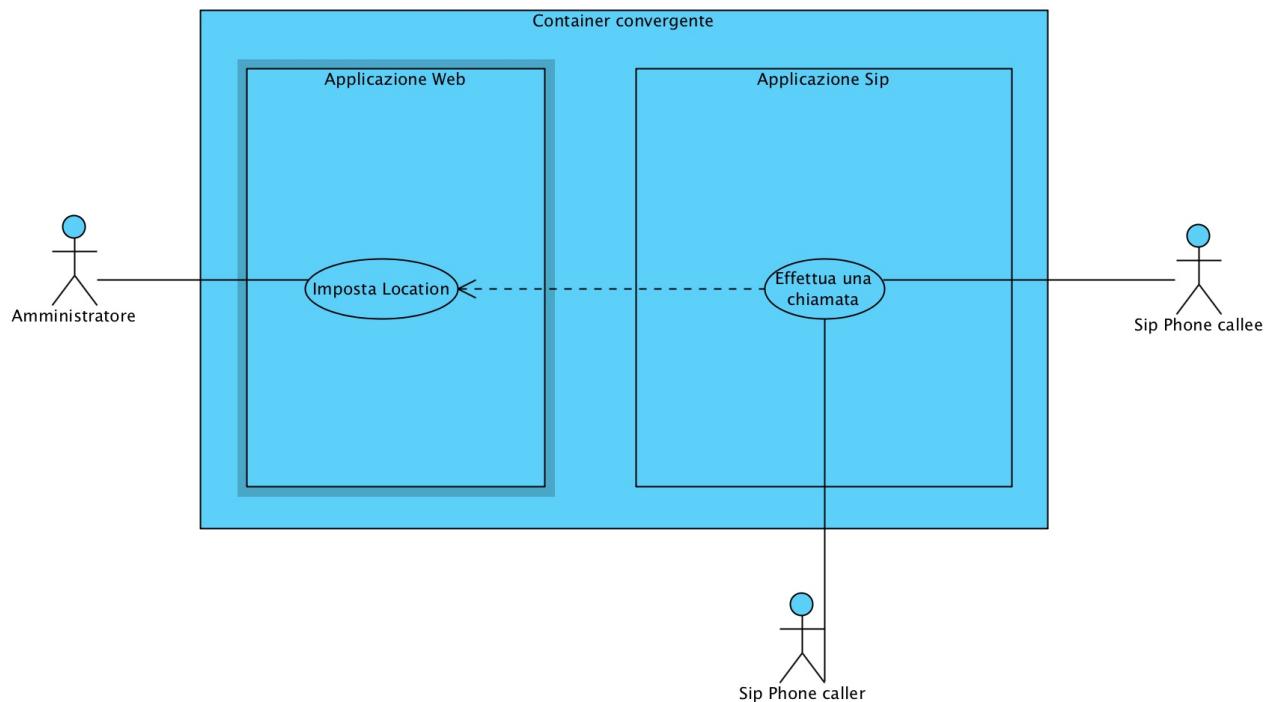


Fig. 10: Diagramma caso d'uso Location

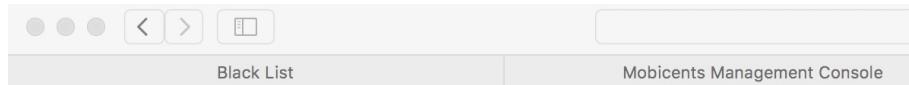
Il codice mostra il contenuto del metodo doGet() con la quale l'amministratore aggiorna le liste relative alla locazione delle Sip Uri.

```

HashMap<String, List<String>> locationList = (HashMap<String, List<String>>) getServletContext().getAttribute("LocationListMap");

if(locationList == null)
    locationList = new HashMap<String, List<String>>();
getServletContext().setAttribute("LocationListMap", locationList);

if(reset != null){
    if(reset.equals("all")){
        locationList.clear();
    }
} else{
    if(locationList.containsKey(uri)){
        locationList.get(uri).add(newUri);
    } else{
        List<String> from = new ArrayList<String>();
        from.add(newUri);
        locationList.put(uri, from);
    }
}
    
```



## Location List

URI:

New Location:

[Pulisci la tua Location List](#)

[Ritorna alla Home](#)

Fig. 11: location.html

In questo modo quando un caller esegue una chiamata tramite, la servlet sip, attraverso l'uso di un proxy, riesce ad inoltrare in parallelo l'Invite a tutte le uri specificate nella lista. La prima Uri che risponde annulla tutte le altre.

Nel caso non ci sia una lista, la servlet risponde con un messaggio di Moved Permanently.

```
List<URI> contactAddresses = null;
HashMap<String, List<URI>> registeredUsers = new HashMap<String, List<URI>>();
List<String> listUri;
List<URI> listUriFactory = new ArrayList<URI>();
HashMap<String,List<String>> URIs = (HashMap<String,List<String>>) getServletContext().getAttribute("LocationListMap");
if(URIs != null){
    listUri = URIs.get(request.getRequestURI().toString());
    if(listUri != null){
        Iterator<String> i = listUri.iterator();
        while(i.hasNext()){
            listUriFactory.add(sipFactory.createURI(i.next().toString()));
        }
        registeredUsers.put(request.getRequestURI().toString(), listUriFactory);
    }
}
contactAddresses = registeredUsers.get(request.getRequestURI().toString());

if(contactAddresses != null && contactAddresses.size() > 0) {
    Proxy proxy = request.getProxy();
    proxy.setRecordRoute(true);
    proxy.setParallel(true);
    proxy.setSupervised(false);
    proxy.proxyTo(contactAddresses);
} else {
    if(logger.isInfoEnabled()) {
        logger.info(request.getRequestURI().toString() + " is not currently registered");
    }
    SipServletResponse sipServletResponse =
        request.createResponse(SipServletResponse.SC_MOVED_PERMANENTLY, "Moved Permanently");
    sipServletResponse.send();
}
```

## CALL FORWARDING

CallForward è un'applicazione convergente che dopo aver ricevuto una richiesta di Invite, controlla la propria lista per scoprire se c'è l'indirizzo di destinazione. In caso positivo la chiamata ha bisogno di essere deviata su un differente indirizzo. Questo significa modificare la precedente richiesta e inviarne una nuova. In questo caso la sip servlet opera come un BackToBack User Agent creando una nuova chiamata che poi è legata alla precedente. Questo è necessario dal momento in cui l'uso del B2B rompe una chiamata ottenendo due dialoghi.

Va notato che in questo caso il DAR considera la nuova chiamata un continuo della precedente, e quindi lo stato del DAR, per quanto riguarda l'invocazione della successiva applicazione nel path, non è azzerato.

La servlet opera da relay e gestisce le due chiamate in maniera asimmetrica, ovvero risponde alle richieste di ACK o Bye inviando una propria risposta e allo stesso tempo inoltra il messaggio sul dialogo con cui ha un legame.

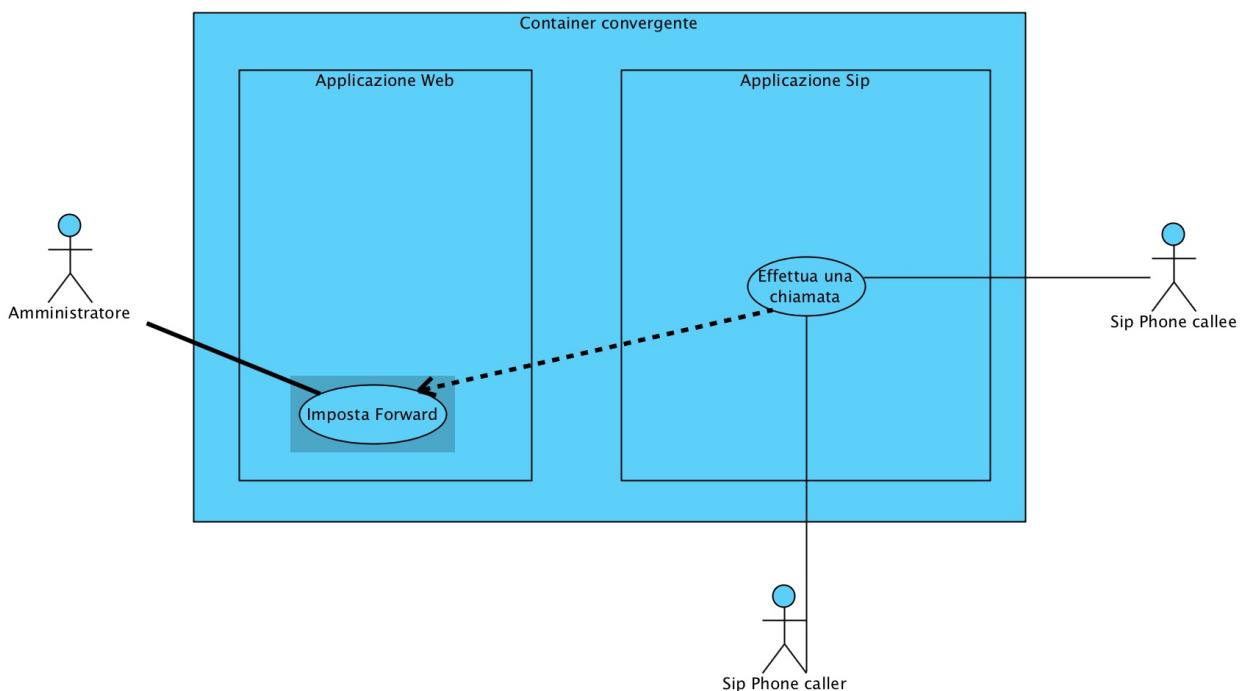


Fig. 12: Caso d'uso Forwarding

```

HashMap<String, String> forwardList = (HashMap<String, String>) getServletContext().getAttribute("ForwardListMap");
if(forwardList != null){
    forwardFromUri = forwardList.get(toUri);
    System.out.println("-----");
    System.out.println(toUri+" Forward to "+ forwardFromUri);
    System.out.println("-----");
}

SipFactory sipFactory = (SipFactory) getServletContext().getAttribute(SIP_FACTORY);
B2buaHelper helper = request.getB2buaHelper();

Map<String, List<String>> headers=new HashMap<String, List<String>>();
List<String> toHeaderSet = new ArrayList<String>();
//Creo una Uri sip che indica dove inoltrare la chiamata. Se non modificata, la request uri resta invariata
SipURI sipUri = (SipURI) request.getRequestURI();

//Se la lista non contiene entry per la toUri
if(forwardFromUri == null){
    //Non modifico la destinazione della request
    logger.info(fromUri + " non è stato modificato il destinatario");
    toHeaderSet.add(request.getRequestURI().toString());
} else {
    logger.info(fromUri + " è stato modificato il destinatario");
    //Cambio destinazione
    toHeaderSet.add(forwardFromUri);
    //Ho bisogno della sipFactory per creare l'uri che userò per inoltrare la nuova richiesta
    sipUri = (SipURI) sipFactory.createURI(forwardFromUri);
}

//Mi occorre per creare la nuova richiesta. in questo modo gli comunico i parametri della request che voglio modificare, i
headers.put("To", toHeaderSet);

//La nuova richiesta che creo è copiata dalla request, alla quale modifico il parametro To.
SipServletRequest forkedRequest = helper.createRequest(request, true, headers);

//Posso ora settare la destinazione della nuova richiesta, che è la stessa dell'originale
forkedRequest.setRequestURI(sipUri);
if(logger.isInfoEnabled()) {
    logger.info("forkedRequest = " + forkedRequest);
}
//Questo è il legame che devo sfruttare nella pipe tra le diverse applicazioni. Posso gestire i subsequent requests
//Mantengo nel contesto la precedente richiesta, mi servirà per inoltrare al giusto mittente la risposta ottenuta
forkedRequest.getSession().setAttribute("originalRequest", request);

forkedRequest.send();

```

Dal codice possiamo vedere l'uso di un B2Bua che opera da relay per la richiesta ricevuta.

## Composizione di Applicazioni convergenti

Dopo aver parlato singolarmente delle tre applicazioni, possiamo unire i diversi servizi per ottenerne uno più sofisticato.

La cosa si ottiene abbastanza facilmente modificando il DAR file dove oltre a specificare le applicazioni interessate ad una determinata richiesta, specifico anche l'ordine di invocazione. La modifica del DAR può essere fatta anche sulla Sip Servlets Management Console dopo aver attivato il server Tomcat.

Nel Primo esempio unisco l'applicazione di callBlocking e Location, quello che ottengo è una Pipe con filtro a monte. Quindi se il caller è presente nella black list del callee la chiamata è rifiutata, mentre nell'altro caso è fatta passare al successivo blocco che implementa il servizio di location con la quale è possibile contattare il calllee su uno dei suoi possibili indirizzi.

Fig. 13: DAR Loc+Block

Un servizio molto più evoluto è quello che realizza una Pipe componendo tutte e tre i servizi come mostrato di seguito.

## Default Application Router Source

```
REGISTER=("AppTelematiche-Sip-MyClickToCall","DAR:From","ORIGINATING","","NO_ROUTE","0")
INVITE=("AppTelematiche-Sip-CallForwarding","DAR:From","ORIGINATING","","NO_ROUTE","0"),("AppTelematiche-Sip-CallForwarding","DAR:From","NEUTRAL","","NO_ROUTE","1"),("AppTelematiche-Sip-Location","DAR:From","TERMINATING","","NO_ROUTE","2")
```

**Close**

Fig. 14 DAR file

Quello che si vuole realizzare è rappresentato nel seguente diagramma:

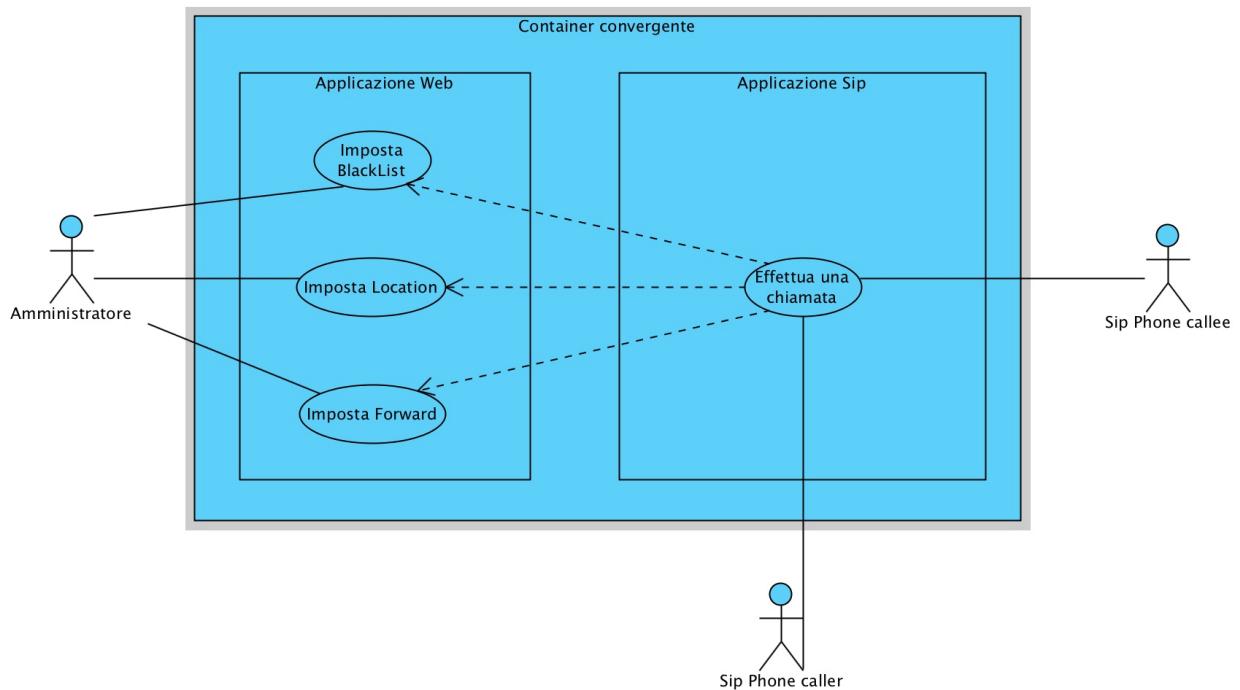


Fig. 15 Caso d'uso composizione

In questo caso l'amministratore può modificare le tre liste relative ai diversi servizi offerti, Mentre gli utenti che effettuano una chiamata dovranno attraversare una catena di tre applicazioni prima di raggiungere il chiamato.

Fig. 16: Sip Management Dar

Di seguito riportiamo anche il diagramma di sequenza relativo alla composizione

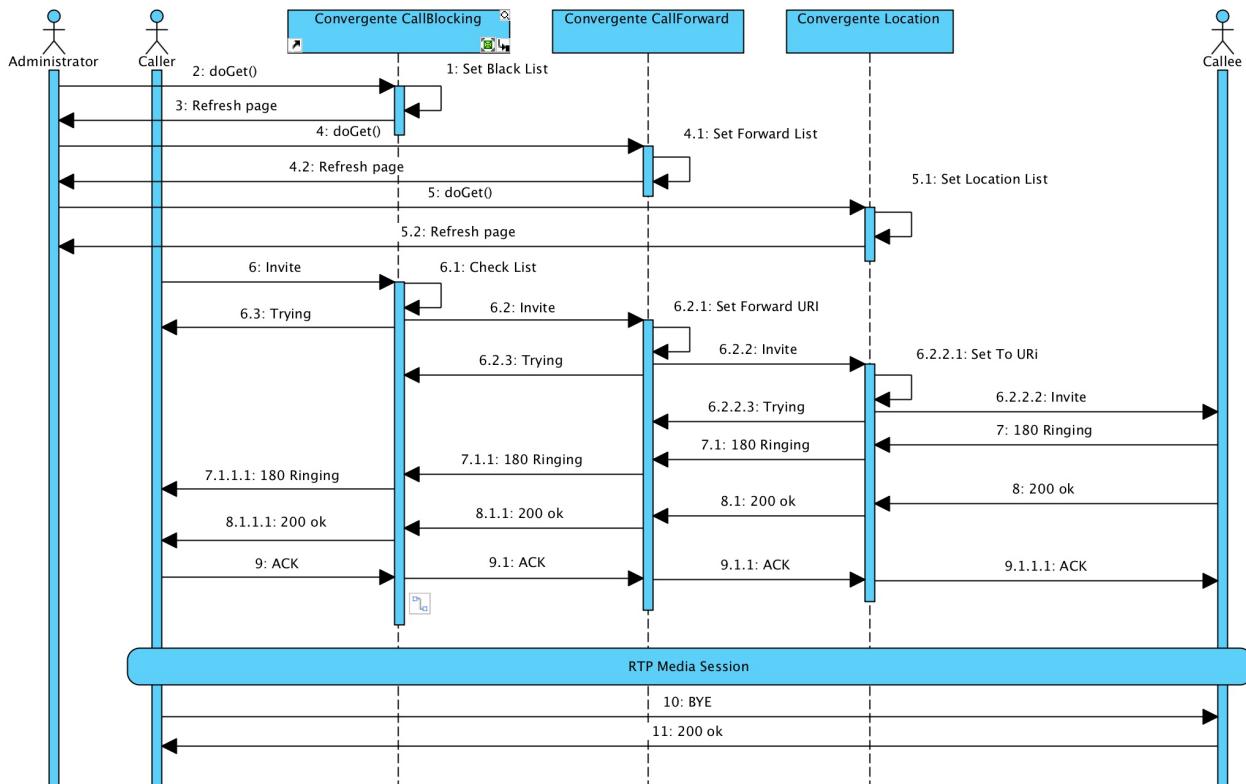


Fig. 17: Seq Diagram composizione

Una volta che l'Invite raggiunge il callee possiamo sfruttare l'application path per i messaggi di risposta e per i subsequent request. Quest'ultimi sono tutte le richieste che fanno parte di un dialogo già esistente e non chiedono di creare uno nuovo.

## myClicktocall

L'applicazione in esame è quasi una replica dell'applicazione fornita come esempio da RestComm. Essa è stata usata solo per testare il server Tomcat per quanto riguarda la convergenza.

Sono comunque state apportate delle leggere modifiche. È stato usato javascript per implementare la validazione della form mostrata nella pagina web. In particolare si controlla se l'utente che si vuole contattare è registrato o meno, in caso positivo l'Uri inserita diventa verde.

Il controllo è stato fatto usando Ajax in modo tale da avere interazioni asincrone col server ed evitando di richiedere l'intera pagina. Sempre attraverso la servlet di validazione, viene controllato e mostrato a video l'assenza di utenti registrati. Inoltre prima di generare la chiamata viene controllata la regolarità delle uri sip usate.

Attraverso la form presente nella pagina web è possibile chiamare le Sip Uri usando le loro Sip complete, mentre nel progetto originale bisognava inserire le URI di contatto, ovvero l'indirizzo esplicito del server.

# TUTORIAL

---

## PIATTAFORMA:

**OS X El Capitan(10.11) oppure macOS Sierra(10.12)**

## INSTALLAZIONE:

### 1. Java:

Assicurati di avere installato almeno una versione della Java Development Kit sulla tua macchina. Puoi eseguire il seguente comando da terminale:

```
javac -version
```

In alternativa puoi scarica una nuova versione della JDK dal sito [Oracle](#), la versione da scaricare è la standard edition, Java SE. Puoi installare una delle seguenti versioni:

JDK 6, 7 o 8.

Attraverso il seguente comando puoi vedere tutte le versioni JDK installate:

```
/usr/libexec/java_home -V
```

Ed eventualmente settare una versione di default con la modifica della variabile d'ambiente

```
export JAVA_HOME= '/usr/libexec/java_home -v 1.7.0_51'
```

Non dovrebbero esserci problemi con le variabili di ambiente, essa è comunque aggiornata per puntare alla versione della JDK più recente.

### 2. Tomcat:

Il progetto è stato realizzato usando un versione modificata di Tomcat 8 che ingloba al suo interno la piattaforma Restcomm. Quest'ultimo estende le caratteristiche di Tomcat permettendo di realizzare applicazioni convergenti(Http/Sip).

Fino al 2016 era possibile scaricare dal seguente [link](#) la versione 8 di Tomcat.

Attualmente è solo presente la versione 7 di Tomcat e la versione 7 di JBoss.

Consiglio pertanto di scaricare la versione messa a disposizione [Tomcat 8](#)

1. Creare la directory `/Library/Tomcat`.

2. Decomprimere il file scaricato e spostare la cartella all'interno della propria libreria, nella directory appena creata. Vi verrà chiesto di inserire la password del vostro account per abilitare l'operazione.

3. Configurare il file `tomcat-users.xml` che si trova nella

directory `/Library/Tomcat/mss-3.1.633-apache-tomcat-8.0.26/conf` bisogna aggiungere alla fine del file le seguenti righe:

```
<role rolename="manager-gui"/>
```

```
<user username="YourAccount" password="YourPassword" roles="manager-gui"/>
```

4. Avviare Tomcat da linea di comando:

```
> /Library/Tomcat/mss-3.1.633-apache-tomcat-8.0.26/bin/catalina.sh  
run
```

5. Testare se il server è raggiungibile alla pagina <http://localhost:8080>, dovrebbe comparire la classica pagina di tomcat

6. Per arrestare il server, sempre da linea di comando, digitare i tasti `Ctrl + c`.

### 3. Eclipse:

1. Scaricare e installare [Eclipse EE](#), questa versione include al suo interno il framework WTP. Se già avete una versione di Eclipse potete installare il framework come plug-in.
2. Collegare Eclipse a Tomcat, questo servirà ad eclipse per eseguire l'avvio e la chiusura del server, il deploy dei progetti e il recupero delle API Sip Servlet che insieme alle API Servlet Http costituiscono le librerie Java per realizzare le nostre applicazioni.  
Da Eclipse, sfruttando il framework WTP, dovrebbe essere possibile creare un nuovo server. Se non è impostata, bisogna scegliere la perspective Java EE, successivamente andando su File/new/other si dovrebbe aprire un wizard dalla quale è possibile selezionare la voce Server. Si seguono le istruzioni per impostare il server avendo cura di collegare come ambiente di esecuzione la cartella dove abbiamo importato il server Tomcat.
3. Al termine del passo 2 dovremmo avere l'icona del server con la quale è possibile attivare e disattivare il server. Per ultima cosa dobbiamo verificare se riesce ad importare in maniera corretta le librerie che ci interessano, per fare questo basta creare un nuovo Dynamic Web Project e selezionare come target runtime il server Tomcat 8 creato. Se tutto va bene dovremmo avere il nostro progetto e andando nella libreria dovrebbe essere visibile quella di Tomcat 8, dove Possiamo trovare le API servlet che ci interessano.

### 4. Linphone:

Per le prove è stato usato il client Sip [Linphone](#).

## TESTING

1. Dopo aver attivato Tomcat da linea di comando, eseguire il deploy degli archivi .war relativi ai 4 progetti.
  2. Da browser andare alla pagina <http://localhost:8080/sip%2Dservlets%2Dmanagement/> e impostare il DAR come mostrato nell'elaborato. Oppure copiare il file DAR presente nella repository GitLab nella cartella DAR di Tomcat.
  3. Lancia un browser qualsiasi e vai alla pagina localhost:8080/App-Telematiche-Sip-myclicktocall/index.html
  4. Accedere alle diverse pagine di configurazioni per impostare le liste di location, forwarding e black list. Inserisci nel primo campo la Uri che vuoi gestire.
  5. Apri l'applicazione Linphone. Visto che ti occorrono almeno due softPhone puoi aprire Linphone in multi-istanza, oppure aprire altre applicazioni per gestire chiamate Sip.
  6. Configura Linphone impostando come proxy l'ip della macchina su cui è stato fatto il deploy del file war. Nel caso in cui il sipPhone gira sulla stessa macchina modificare il porto usato per il protocollo Sip.
- L'username della Sip che crei può essere arbitrario, mentre il nome del server deve coincidere con l'ip del server Tomcat.

7. Esegui una chiamata da Linphone verso l'Uri dell'altro SipPhone, se l'uri del chiamante si trova nella black list del chiamato si otterrà una risposta Forbidden. Altrimenti la chiamata è passata al successivo blocco. In quest'ultimo caso la chiamata è deviata verso la nuova destinazione se era specificato un forward. In ogni caso la richiesta arriva al terzo e ultimo blocco che inoltra la richiesta verso le diverse locazioni in cui è possibile trovare il chiamato. Se esiste almeno un indirizzo valido la ha inizio.