UNIVERSITÀ DEGLI STUDI DI PALERMO
FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica

IMPLEMENTAZIONE DI UN SISTEMA DI VIDEOSORVEGLIANZA PER ANDROID

TESI DI LAUREA DI
VITO GENTILE

RELATORE
PROF. GIORGIO VASSALLO

ANNO ACCADEMICO 2010 – 2011

Implementazione di un Sistema di Videosorveglianza per Android

Vito Gentile
Università degli Studi di Palermo
Laurea in Ingegneria Informatica
A.A.: 2010/2011
Relatore: prof. Giorgio Vassallo

Abstract

L'obiettivo di questa tesi consiste nello sviluppo di un'applicazione per dispositivi mobili Android, che implementi le funzionalità di un sistema remoto di videosorveglianza. Tale applicazione sfrutta le API fornite da Android più alcuni software open source. Saranno utilizzate, inoltre, alcune librerie Java per l'utilizzo dei protocolli SSH ed MQTT per la comunicazione con un server.

Saranno descritti i dettagli implementativi e saranno motivate le scelte intraprese.

Infine, vengono proposte le possibili evoluzioni dell'applicazione, valutate considerando le potenzialità offerte dai software e dagli strumenti utilizzati.

1. Introduzione

Un'indagine della Gartner Inc. [1], società multinazionale leader nella consulenza strategica, nella ricerca e nell'analisi nel campo della tecnologia dell'informazione, mostra che nel 2010 il sistema operativo più utilizzato nell'ambito dei dispositivi mobili è stato Symbian di Nokia, seguito da Android di Google e da iOS di Apple. Ma la stessa indagine mostra anche alcune previsioni di crescita nel tempo: mentre Android ed iOS sono sistemi operativi "giovani", il progetto Symbian è stato abbandonato da Nokia, a favore di Windows Phone di Microsoft. Probabilmente per questo motivo il sistema operativo maggiormente quotato risulta essere Android, il quale, stando sempre a questa tesi, dovrebbe raggiungere il 49,2% del market share entro la fine del 2012.

Nonostante questo primato, e nonostante il vasto numero di applicazioni disponibili sul market place fornito da Google, Android non fornisce software validi per la videosorveglianza tramite smartphone. Il principale motivo di questa assenza deriva, probabilmente, dal fatto che si tratta di applicazioni distribuite, per le quali è necessaria non solo una implementazione lato client (ovvero sullo smartphone vero e proprio), ma anche una lato server (e quindi su un computer che, verosimilmente, monterà un sistema operativo diverso da Android). Perciò è la natura stessa del progetto a rendere la commercializzazione di un prodotto simile molto più difficoltosa delle comuni applicazioni per smartphone, quali giochi, agende o piccoli software di grafica.

Le possibilità offerte da un sistema di videosorveglianza come quello che verrà descritto sono innumerevoli. I costi di una webcam sono ormai accessibili a tutti, e le applicazioni possono riferirsi agli ambiti più disparati. Non solo è possibile offrire un servizio di sicurezza riservato a case private, banche, musei o altri istituti pubblici, ma sono anche ipotizzabili applicazioni accessibili al pubblico: sorvegliare aree pubbliche e offrire servizi di sicurezza a più utenti. Sarebbe possibile, ad esempio, limitare gli incendi dolosi sorvegliando aree urbane o forestali. Questi scenari rappresentano solo un settore delle possibilità di applicazione. Tutti hanno la caratteristica di utilizzare tecnologie a basso costo.

Al fine di presentare il progetto, nel seguito si esporranno dapprima le caratteristiche fondamentali delle applicazioni e degli strumenti utilizzati.

Una volta stilata rapidamente una lista di tali strumenti, innanzitutto verrà presentato il sistema operativo Android e le API fornite per l'implementazione dei software. In seguito, verrà posta l'attenzione sul protocollo MQTT, utile per l'implementazione delle notifiche push su Android, e sul funzionamento di Motion, applicativo per sistemi operativi Linux, in grado di riconoscere l'avvenuto movimento con l'utilizzo di webcam o videocamere.

La parte centrale tratterà del funzionamento dell'applicazione, spiegandolo nei termini dettati dalla struttura di base delle applicazioni Android. Per finire, verrà presentata una lista dei possibili sviluppi futuri dell'applicazione, evidenziando i punti di maggiore interesse, e sottolineandone le possibilità di ampliamento.

2. Gli strumenti di lavoro

Sarà utilizzato un server Linux (nel caso specifico, è stata utilizzata la distribuzione Ubuntu 10.04 per la facilità d'uso, per il supporto che fornisce la comunità, e per la vasta gamma di applicazioni disponibili nei repository ufficiali), che dovrà disporre di alcuni strumenti software che svolgano funzionalità di supporto allo sviluppo dell'applicazione. Gli strumenti utilizzati sono i seguenti:

- *Motion*: si tratta di un programma che consente di monitorare i segnali video prodotti da una videocamera, ed è in grado di individuare se tra due immagini vi sono variazioni significative. È facilmente configurabile, e consente di eseguire comandi al verificare di particolari eventi [3].
- *Mosquitto*: message broker open source, essenziale per utilizzare il protocollo MQTT di IBM [4]. Nei prossimi paragrafi sarà approfondito il funzionamento di tale protocollo e, conseguentemente, quello di Mosquitto. Per ora è sufficiente spiegare che tale message broker sarà utile per realizzare un meccanismo di notifiche push che consentano di avvisare l'utente in tempo quasi-reale.
- *OpenSSH Server*: il computer remoto funzionerà anche da server SSH, acronimo che sta per "Secure Shell", e indica un protocollo sicuro che consente di stabilire connessioni remote cifrate. OpenSSH è una implementazione open source di tale protocollo [9], ed è disponibile nei repository ufficiali di Ubuntu.
- *Apache HTTP Server*: si tratta di un server HTTP open source, utilizzato in questo contesto unitamente a PHP per richiedere le immagini da visualizzare sullo smartphone.
- *PHP 5*: linguaggio di scripting interpretato utilizzato, in questo ambito, per generare pagine web dinamiche che saranno visualizzate dallo smartphone, simulando uno streaming con frame rate molto basso.

Ai fini dell'implementazione dell'applicazione lato client, cioè quella che dovrà funzionare direttamente su Android, sarà necessario utilizzare alcune librerie Java che consentano di effettuare connessioni SSH e di utilizzare i paradigmi tipici del protocollo MQTT. Tali librerie sono:

- *Ganymed SSH-2*: si tratta di una libreria open source che implementa il protocollo SSH, consentendo di effettuare connessioni criptate. Tutte le funzionalità di cifratura sono incluse nella libreria, scritta unicamente in Java; ciò consente l'utilizzo su Android senza problemi di compatibilità [6].
- *WMQTT IA92*: è una utility sviluppata da IBM, facente parte di una collezione di implementazioni Java di client MQTT, che consente la pubblicazione e la sottoscrizione a topic attraverso la connessione ad un message broker [7] (si veda il paragrafo 5.3).

Verranno ora presentate le principali caratteristiche degli strumenti software sopra elencati.

3. Android: un S.O. per smartphone

Android è un sistema operativo open source per dispositivi mobili, basato sul kernel Linux 2.6. Al di sopra del kernel, vi è un gran numero di librerie di sistema, utilizzate a loro volta dalle applicazioni [5].

Ogni singola applicazione Android viene eseguita su una macchina virtuale, la *Dalvik Virtual Machine* (DVM), e ad ogni esecuzione viene generato un singolo processo di questa macchina virtuale. Tale iter di esecuzione è del tutto simile a quanto accade alle applicazioni scritte in Java e compilate a formare file in bytecode. I file eseguibili per la DVM hanno estensione .dex, e risultano ottimizzati per un'occupazione di memoria minimale.

Tutte le applicazioni per Android sono, quindi, scritte in Java. La compilazione dei sorgenti, però, non converte il codice in bytecode per la Java Virtual Machine, ma la conversione viene fatta per la Dalvik Virtual Machine sopra citata. Questo tipo di approccio consente ad ogni applicazione di essere perfettamente portabile, esattamente come accade per Java.

Ogni applicazione, tuttavia, non è costituita soltanto da un file eseguibile, bensì da più file, dalle risorse grafiche (quali immagini o file audio e video) ai file XML, utili a definire stringhe e, soprattutto, utilizzati per strutturare, al di fuori del codice vero e proprio, l'interfaccia grafica delle applicazioni. Tutti i file che costituiscono l'applicazione, dopo la compilazione, vengono inclusi in un pacchetto con estensione .apk (acronimo che sta per *Android Package*): si tratta di una variante "ad-hoc" del formato .jar, ben noto e utilizzato nella programmazione Java.

Al di sotto di ogni applicazione vi è un set di servizi e sistemi che includono:

- un ricco ed estensibile insieme di viste (*View*), che possono essere utilizzate nelle interfacce grafiche come pulsanti, moduli di inserimento di testo, immagini, liste, tabelle ed anche un browser incorporabile;
- dei *Content Provider*, che consentono alle applicazioni di accedere ai dati di altre applicazioni (come la rubrica), o di mettere i propri dati a disposizione di applicazioni esterne;
- un gestore di risorse (*Resource Manager*), che consente l'accesso a risorse non incorporate nel codice, quali immagini, suoni o files di layout;
- un gestore di notifiche (*Notification Manager*), che consente ad ogni applicazione di visualizzare notifiche personalizzate nella barra di stato;
- un gestore di attività (*Activity Manager*), che gestisce il ciclo di vita di ogni applicazione.

Al fine di sfruttare tutte le risorse disponibili, Android fornisce un plug-in per l'IDE Eclipse, denominato ADT (acronimo che sta per *Android Development Tools*). Questo plug-in include un simulatore, che consente il test delle applicazioni senza la necessità di utilizzare un dispositivo fisico, e una serie di strumenti utili per il debug e per l'interfacciamento con un qualsiasi smartphone (e con il simulatore).

3.1. Caratteristiche di base delle applicazioni

Android è un sistema operativo multiuser, nel quale ogni applicazione rappresenta un diverso utente. Ciò comporta che ogni applicazione potrà compiere un certo tipo di operazioni, a meno che i permessi di utilizzo di particolari risorse (quali la rete internet o la vibrazione del cellulare) non vi siano espressamente assegnati, e comunque dopo che l'utente ne abbia approvato l'uso. I permessi devono essere inclusi nel file *Manifest.xml* (figura 1), insieme ad una descrizione della struttura dell'applicazione in termini di componenti utilizzati. Questo file consente, inoltre, di specificare i requisiti minimi richiesti dall'applicazione (come, ad esempio, la minima versione delle API necessaria al corretto funzionamento).

Ogni applicazione viene eseguita su un suo processo, a cui è assegnato un ID univoco dal sistema operativo. Questo processo verrà creato non appena un componente qualsiasi dell'applicazione richiederà di essere avviato. Fatto ciò, il sistema operativo distruggerà il processo quando la sua esecuzione non sarà più necessaria, o quando è richiesta ulteriore memoria.

È bene, inoltre, aggiungere che Android implementa il *principio del minimo privilegio*: ogni applicazione, di default, ha accesso solo e soltanto ai componenti che richiede per poter funzionare. Questo crea un ambiente molto sicuro, nel quale un'applicazione non può avere accesso a parti del sistema per le quali non ha i permessi.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest ... >
3.
4.   <uses-permission
5.     android:name="android.permission.INTERNET" />
6.
7.    <application
8.      android:icon="@drawable/app_icon.png" ... >
9.
10.     <activity
11.       android:name="com.example.ExampleActivity"
12.       android:label="@string/example_label" ... >
13.     </activity>
14.
15.    ...
16.
17.   </application>
18. </manifest>
```

Figura 1. Struttura del file *Manifest.xml*

3.2. Componenti principali delle applicazioni

3.2.1. Activity. Una activity rappresenta una singola schermata con un'interfaccia utente. Ogni activity è indipendente dalle altre, ed ha un preciso ciclo di vita, riassunto nel diagramma di figura 2. Una generica activity, è implementata estendendo la classe *Activity* delle librerie di Android.

3.2.2. Service. Un service è un componente che viene eseguito in background al fine di svolgere operazioni di lunga durata o per gestire processi remoti; perciò un service non fornisce un'interfaccia all'utente. Ogni service è implementato estendendo la classe *Service* delle librerie di Android.

3.2.3. Content Provider. Un content provider gestisce la condivisione di dati fra le applicazioni. È possibile salvare i dati su un database SQLite, sul web o in altri dispositivi di archiviazione persistenti (quali le memorie SD). Un content provider è implementato estendendo la classe *ContentProvider* delle librerie di Android.

3.2.4. Broadcast Receiver. Un broadcast receiver è un componente che risponde a messaggi di broadcast a livello di sistema (quali l'avviso che la batteria è scarica, o l'avvenuta cattura di un'immagine). Tali componenti non forniscono un'interfaccia utente, ma comunicano con questo tramite notifiche nella barra di stato. Un broadcast receiver è implementato estendendo la classe *BroadcastReceiver* delle librerie di Android, ed ogni messaggio di broadcast è inviato tramite un *Intent* (utilizzato anche per molti altri scopi, quali l'avvio di un service o di un'activity).

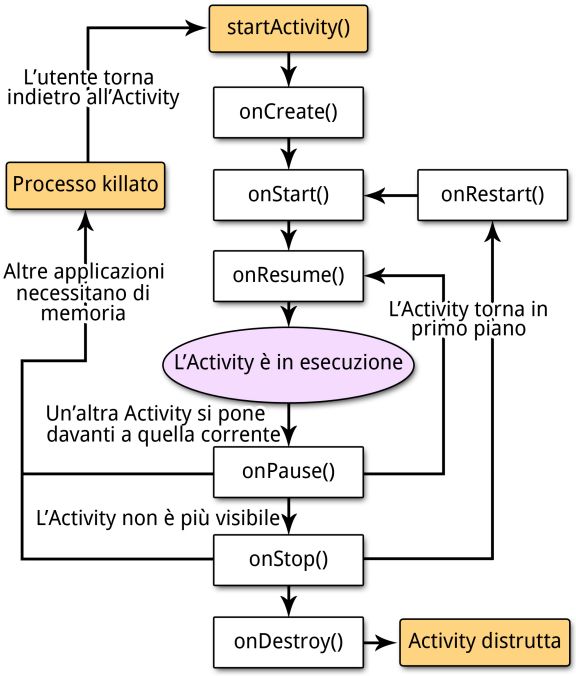

Figura 2. Ciclo di vita di una Activity [5]. La transizione da uno stato all'altro viene eseguita in seguito alla chiamata di metodi propri di ogni activity.

3.2.5. Avvio dei componenti. Su Android, ogni applicazione può avviare un componente di un'altra applicazione. Questo componente lascerà spazio all'applicazione che lo ha invocato non appena viene distrutto. Activity, service e broadcast receiver sono attivati da messaggi asincroni, utilizzando un *Intent*, ovvero un oggetto che comunica al sistema operativo "l'intenzione" dell'applicazione di inizializzare ed utilizzare il componente. I metodi startActivity(),

startService() o sendBroadcast() richiedono un Intent per potere essere utilizzati. I content provider, invece, sono attivati effettuando una richiesta ad un *ContentResolver*, ovvero un oggetto che ottiene i dati comunicando direttamente con la base di dati o recuperandoli dai dispositivi di archiviazione. Questa filosofia consente di mantenere un livello di astrazione tra l'applicazione ed i dati veri e propri.

In aggiunta a quanto detto, si introduce il funzionamento della preferenze condivise, implementate con la classe *SharedPreferences* delle librerie di Android. Si tratta di un meccanismo in grado di salvare dati relativi ad una particolare applicazione sfruttando la memoria interna dello smartphone, e consentendone l'accesso alla sola applicazione che ne ha gestito la creazione. Tale struttura è utile nel caso in cui si vogliano memorizzare per un periodo di tempo indefinito e superiore al tempo di esecuzione, delle "preferenze" relative ad un'applicazione.

4. Individuare un movimento: Motion

I repository ufficiali di Ubuntu 10.04 (ma anche quelli relativi alla maggior parte delle versioni precedenti e successive) mettono a disposizione *Motion*, applicazione a riga di comando open source, scritta in C e rilasciata sotto licenza GPL, che consente di registrare eventuali movimenti percepiti a partire da una sequenza di immagini provenienti da una webcam.

Una volta installato, Motion provvede alla creazione di un gruppo di utenti, denominato "motion". Gli utenti appartenenti a questo gruppo saranno tutti quelli che potranno eseguire l'applicazione, avviandola ed eventualmente terminandola.

Si tratta di un software altamente personalizzabile: è, infatti, presente un file di configurazione, posto nella directory /etc/motion, chiamato *motion.conf*; esso include un gran numero di parametri riguardanti alcune variabili fondamentali per l'utilizzo ed il funzionamento degli algoritmi. Le sezioni più importanti del file *motion.conf* sono:

- Sezione "DAEMON": consente di stabilire se Motion deve essere eseguito come demone (ovvero in background) o meno; inoltre, è possibile settare il nome predefinito di un file che conterrà il PID di ogni processo di Motion in esecuzione. Tale file, tuttavia, non verrà utilizzato nell'implementazione del sistema di videosorveglianza.
- Sezione "CAPTURE DEVICE OPTIONS": consente di modificare le preferenze relative ai driver da utilizzare e alla regolazione delle immagini (luminosità, contrasto, risoluzione, etc...). I valori che dovranno assumere le variabili dipenderanno comunque dal tipo di webcam/videocamera in uso.
- Sezione "MOTION DETECTION SETTINGS": contiene alcuni parametri che consentono di precisare dettagli utili all'algoritmo che riconosce il movimento. Oltre ai parametri che definiscono le tolleranze e le soglie di rumore per il riconoscimento, un'importante variabile è il "gap", che rappresenta l'intervallo di tempo necessario affinché Motion (nel caso di assenza di movimento) possa considerare terminato un evento. Per *evento* si intende, da qui in poi, l'intervallo temporale nel quale c'è stato un movimento, dal primo spostamento fino a "gap" secondi dopo l'ultimo movimento.
- Sezioni "IMAGE FILE OUTPUT" e "FFMPEG RELATED OPTIONS": consentono di modificare i parametri relativi al salvataggio delle immagini (formato e, nel caso di utilizzo di compressione jpeg, qualità) e delle registrazioni video. Riguardo ai video, è possibile scegliere diversi formati di salvataggio, alcuni dei quali (swf ed flv) si adattano molto alla riproduzione in streaming (e possono, quindi, essere utilizzati per la riproduzione delle registrazioni sul client).
- Sezione "TARGET DIRECTORIES AND FILENAMES FOR IMAGES AND FILMS": consente di modificare le preferenze riguardanti i nomi dei file e la directory in cui verranno salvate le immagini ed i video. Sarà importante definire una struttura precisa per le directory, descritta più avanti.
- Sezioni "LIVE WEBCAM SERVER" e "HTTP BASED CONTROL": consentono di utilizzare un browser web per la configurazione e la visualizzazione live, definendo la possibilità di accedere unicamente da locale o anche da remoto a queste funzionalità, eventualmente predisponendo uno username ed una password. È possibile definire la porta tramite la quale accedere ad un pannello di controllo predefinito (che fornisce sostanzialmente un'interfaccia per modificare le variabili di *motion.conf*), e quella su cui indirizzare le immagini prodotte dalla webcam in tempo reale. Tuttavia non è possibile sfruttare queste possibilità, perché la frequenza di aggiornamento e l'ampiezza di banda disponibile al momento della visualizzazione non possono essere sempre compatibili: frequenze non troppo alte (1 fps) unitamente ad una banda modesta impedirebbero totalmente la visualizzazione delle immagini in streaming, a causa dei tempi di caricamento dei browser di Android.
- Sezione "EXTERNAL COMMANDS": fornisce a Motion una delle sue caratteristiche primarie, ovvero la possibilità di eseguire dei comandi al verificarsi di particolari situazioni: per esempio alla creazione o alla chiusura di un file video (cioè in corrispondenza dell'inizio e della fine di un evento, per come precedentemente definito), in corrispondenza di ogni movimento riconosciuto, eccetera. In particolare, questa sezione sarà utilizzata per eseguire un comando che invierà un messaggio di allarme al client ogni qualvolta verrà identificato un movimento (cioè all'inizio di un evento, come precedentemente definito).

Per avviare Motion è sufficiente eseguire, da terminale, il comando:

$ motion

Se nella sezione “DAEMON” del file motion.conf, il parametro “daemon” è valorizzato ad “on”, questo comando verrà eseguito in background, e per fermarlo sarà necessario fermare il demone, eseguendo il comando:

service motion stop

che equivale a

/etc/init.d/motion stop

Questa operazione richiede, tuttavia, i privilegi di amministratore. Ciò sarebbe un problema nel caso in cui l'avvio e la fine delle registrazioni debbano essere decise dal client remoto tramite una connessione SSH: sarebbe, infatti, necessario accedere come utente root. La soluzione consiste nel cambiare il gruppo di appartenenza del file */etc/init.d/motion*, da “root” a “motion”, con il comando chgrp:

chgrp motion /etc/init.d/motion

Quanto descritto finora comprende tutte le funzionalità sfruttate dal server. Va, però, aggiunto che è possibile configurare più webcam, ed ottenere i risultati delle registrazioni da ognuna di esse. Per farlo, è necessario creare tanti files di configurazione quante sono le webcam e rinominarli adeguatamente.

Fatte queste premesse, è chiaro che sfruttare le potenzialità di Motion è semplice ed allo stesso tempo molto conveniente. Utilizzando il protocollo SSH, infatti, è possibile stabilire una connessione ed eseguire i comandi di *start* e *stop* direttamente da un client remoto, che può, inoltre, ottenere le registrazione (cioè i file video salvati) utilizzando sempre lo stesso protocollo.

5. Notifiche “Push”: il protocollo MQTT

5.1. Tecnologia Push e Tecnologia Pull

Il problema più arduo da risolvere consiste nell'implementazione di un meccanismo di notifiche che soddisfi alcune importanti caratteristiche. Innanzitutto, sarà necessario ricevere le notifiche nel minore tempo possibile, ovvero in tempo “quasi-reale”. In secondo luogo, l'implementazione non dovrà richiedere troppe risorse al client, principalmente per evitare consumi in termini di risparmio energetico. Prima di descrivere la soluzione adottata, è bene precisare la differenza tra tecnologia “pull” e tecnologia “push”. Per farlo, si prenda ad esempio il caso in cui si vuole verificare se un file su un server SSH è stato modificato o meno.

La tecnica più immediata (e relativamente semplice da implementare) è quella di controllare periodicamente e continuamente se il contenuto del file presente sul server coincide con una copia del file precedentemente visionato, sfruttando, ad esempio, un ciclo while che effettua un controllo ad ogni iterazione. Questo tipo di approccio è un classico esempio di tecnologia “pull” (o di polling):

1. il client chiede al server il file
2. il server risponde
3. ottenuta la risposta, il client ricomincia dal punto 1 dopo una breve pausa

Questo approccio può essere utile per il controllo di dati che, pur essendo soggetti a mutamento temporale, non variano di molto: un classico esempio è quello delle condizioni meteo. Potrei, infatti, controllare i valori effettuando una serie di connessioni ad un certo server che contiene i dati che mi interessano. Anche se eseguo tale connessione ogni 5 minuti, verosimilmente mi aspetto che non ci siano variazioni sostanziali di temperatura (al più, l'errore sarà di 1° C). Perciò una filosofia pull è decisamente accettabile, poiché, pur non richiedendo troppe connessioni, rende l'applicazione efficiente.

Nel caso di un sistema di videosorveglianza, è consigliabile evitare un approccio di tipo pull. Utilizzando, infatti, una frequenza di richieste bassa, non sarebbe possibile ottenere la proprietà di real-time desiderata. D'altro canto, sarebbe notevolmente svantaggioso effettuare connessioni con una frequenza troppo elevata, poiché il consumo della batteria sarebbe eccessivo e la CPU sarebbe troppo stressata. Per questo genere di esigenza, è preferibile utilizzare una tecnologia di tipo “push”:

1. il client si connette al server
2. il server identifica il client e resta in attesa
3. non appena si verifica un certo evento, il server invia al client un messaggio (asincrono)
4. il client lo riceve, e si ricomincia dal punto 3

In questo modo il carico di lavoro dovuto alle connessioni, visto per la filosofia pull, viene interamente affidato al server, mentre il client non richiederà consumi eccessivi. Non appena il messaggio sarà inviato dal server, inoltre, il client lo riceverà “quasi subito”, in funzione della connessione e dell'implementazione di tale tecnologia.

5.2. Notifiche Push su Android

Purtroppo, le API fornite con l'SDK di Android non includono meccanismi di notifiche push. Il problema si riduce, quindi, all'implementazione di un tale strumento, in grado di soddisfare le esigenze esposte precedentemente. Per risolvere tale problema, sono stati analizzati diversi metodi e servizi. I più importanti sono i seguenti:

- *C2DM* (Cloud To Device Messaging): è un servizio di Google che consente di ricevere messaggi attraverso un cloud server, al quale è possibile accedere attraverso il proprio account Google [10]. Questa soluzione è implementabile su tutti i dispositivi che montano una versione di Android maggiore o uguale a 2.2
- *Deacon project*: è un progetto open source abbastanza giovane che ha l'obiettivo di sviluppare una libreria per Android che consenta facilmente l'implementazione di notifiche Push per Android [8].

- *Protocollo XMPP* (Extensible Messaging and Presence Protocol): si tratta di un protocollo creato per la messaggistica istantanea [11]. È possibile sfruttarlo per le notifiche Push su Android.
- *Protocollo MQTT* (MQ Telemetry Transport): è un protocollo sviluppato da IBM che consente di inviare e ricevere messaggi in modo estremamente leggero, basandosi su un meccanismo di iscrizione/pubblicazione attraverso il supporto di un message broker (strumento che distribuisce i messaggi agli host coinvolti) [2]. Esiste anche una libreria Java, implementata e fornita da IBM, il cui funzionamento sarà approfondito nei prossimi paragrafi.

Tra queste alternative, che sono risultate le più rilevanti, si è scelto di utilizzare il protocollo MQTT. Le motivazioni stanno sia nella leggerezza con cui vengono inviati i messaggi (proprietà di estrema importanza considerando che i piani tariffari delle compagnie telefoniche sono spesso basati sulla quantità di dati trasferiti), sia nella possibilità di avere un valido supporto software, che consiste nella disponibilità di una libreria Java per l'utilizzo di tale protocollo e di un message broker MQTT open source, facilmente utilizzabile. Prima di chiarire il funzionamento di MQTT, è bene precisare le motivazioni che hanno portato a scartare le altre alternative.

Innanzitutto, il Deacon project è un progetto troppo giovane, e richiede una ulteriore fase di test prima di essere utilizzato, sebbene la strada percorsa sembra quella giusta. C2DM necessita di una registrazione, ed il server sarebbe diverso da quello su cui andrebbe installato Motion; ciò potrebbe causare confusione, ed un'eccessiva distribuzione dell'applicazione, peraltro non necessaria. L'XMPP, infine, sarebbe una scelta discreta, ma tale protocollo non risulta leggero come MQTT, che richiede meno risorse.

5.3. Il Protocollo MQTT: funzionamento

Nel 1999, IBM ha rilasciato un protocollo di rete, del livello applicativo, estremamente leggero, chiamato MQTT. Tale protocollo non è standardizzato, ma è stato inizialmente pensato per essere aperto, semplice e facile da implementare (oltre che leggero). Originariamente fu pensato per l'utilizzo in ambiti di telemetria ed in contesti in cui l'ampiezza di banda disponibile fosse limitata. È stato utilizzato in applicazioni di comunicazione via satellite di dati ottenuti da sensori o in ambiti di automazione domestica.

Come già accennato, MQTT si basa su un meccanismo di *pubblicazione/sottoscrizione*: ogni client ha, infatti, la possibilità di iscriversi ad un particolare *topic*, e può pubblicare un messaggio su un qualsiasi topic (non essendovi necessariamente iscritto).

I topic possono essere generati al momento della sottoscrizione o pubblicazione del messaggio, e tutti i client inscritti ad un certo topic riceveranno tutti i messaggi pubblicati su quest'ultimo, indipendentemente dall'autore. Come si intuisce, si tratta di un meccanismo molto simile alla pubblicazione di messaggi e risposte all'interno di un qualsiasi forum.

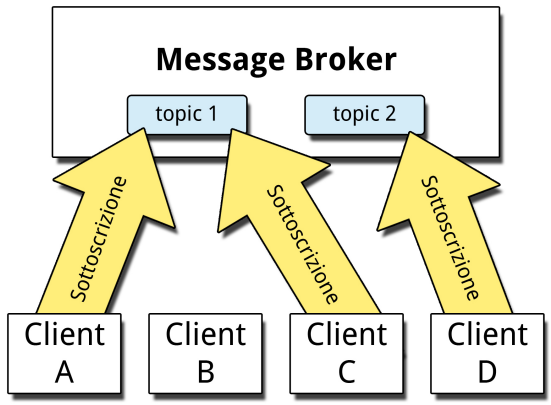

Figura 3. Sottoscrizione dei client a diversi topic. Più client possono iscriversi ad uno stesso topic, ed uno stesso client può iscriversi a più topic.

Per consentire la pubblicazione e la sottoscrizione, ed eventualmente per gestire l'invio o la memorizzazione di messaggi ed iscrizioni, è necessario che sul server sia installato un *message broker* (ovvero un "mediatore"). Tale message broker recapiterà i messaggi a tutti i client iscritti al topic, e memorizzerà la lista delle iscrizioni.

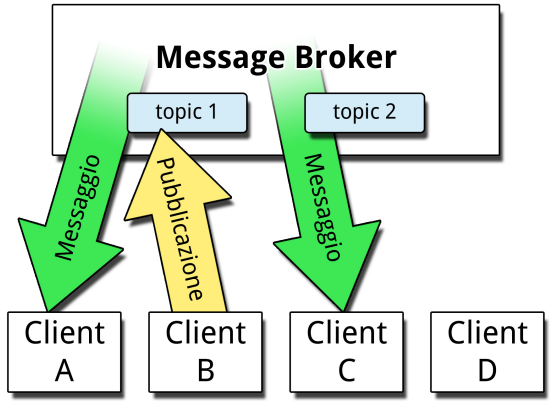

Figura 4. Pubblicazione di un messaggio su un topic da parte di un client. Quando B pubblica il messaggio sul topic 1, il message broker inoltra tale messaggio a tutti i client sottoscritti al topic 1 (con riferimento alla figura precedente, i client in questione erano A e C)

Compreso il meccanismo di funzionamento di questo protocollo (figure 3 e 4), risulta semplice comprendere in che modo è possibile sfruttarlo in uno smartphone, sottoscrivendo quest'ultimo ad un certo topic (che possiamo chiamare, per esempio, "videosorveglianza"). Non appena il message broker sul server riceve la pubblicazione di un messaggio da un certo client sul topic, tale messaggio sarà inoltrato allo smartphone (oltre che a tutti gli altri client eventualmente sottoscritti), ed interpretato come avviso di allarme.

Insieme al protocollo, la IBM ha anche sviluppato un message broker minimale: *RSMB*, acronimo che sta per "Really Small Message Broker". RSMB consente l'utilizzo delle funzionalità di base di un message broker, ma non è open source. Quest'ultima caratteristica è, invece, propria di un'altra valida alternativa: *Mosquitto*.

Nel sito ufficiale di Mosquitto è, infatti, disponibile il codice sorgente, insieme alle istruzioni per l'installazione e la configurazione. La pubblicazione di un messaggio su un topic può essere attuata dal server stesso (che può, quindi, fungere anche da client). Questo message broker dispone di un'interfaccia a riga di comando, attraverso la quale è possibile pubblicare messaggi su un qualsiasi topic.

La scelta del message broker ricade, dunque su Mosquitto, poiché risulta di facile configurazione ed utilizzo.

5.4. La libreria WMQTT IA92

Al fine di utilizzare le potenzialità di MQTT, è necessario uno strumento di programmazione che consenta di effettuare sottoscrizioni a particolari topic, ed allo stesso tempo sia in grado di gestire la ricezione di eventuali messaggi provenienti dal message broker. Lo strumento in questione è la libreria *WMQTT IA92*, fornita da IBM sul sito ufficiale, e creata al fine di implementare applicazioni WebSphere di integrazione aziendale per la connettività applicativa.

Tale libreria fornisce molte funzionalità che vanno oltre gli scopi qui prefissati. Le classi che interessano l'applicazione sono quelle che consentono le funzionalità base di un client MQTT, ovvero pubblicazione, sottoscrizione e ricezione. Al fine di utilizzarle, è possibile sfruttare la classe *IMqttClient*, che consente di effettuare una connessione al server tramite il metodo *connect()*, e di registrare un handler in grado di intercettare l'arrivo di un messaggio. Inoltre, tale classe fornisce i metodi *publish()* e *subscrive()*, in grado rispettivamente di pubblicare un messaggio su un topic e sottoscrivere il client ad un topic.

Per gestire le eccezioni, esistono apposite classi, quali la *MqttException* ed *MqttPersistenceException*, che consentono di catturare eventuali errori dovuti principalmente a problemi di connessione.

6. OpenSSH Server

SSH è un protocollo di rete di livello applicativo, che consente di stabilire connessioni remote sicure attraverso processi di criptazione. Un client SSH dispone di un'interfaccia a riga di comando, tramite la quale è possibile accedere ad un computer remoto, stabilire una sessione, e gestire l'host remoto in modo semplice.

Sono previsti due metodi principali per effettuare un login da un client ad un server SSH. Essi si basano su:

- *username e password*: l'utente che vuole connettersi fornisce il suo nome utente e la sua password, inviati attraverso un canale cifrato, che verranno poi verificati dal server e, se accettati, sarà stabilita la connessione;
- *chiave pubblica*: per evitare di utilizzare username e password, l'utente può generare una coppia di chiavi, basate sulle codifiche DSA o RSA, eventualmente utilizzando una passphrase per la creazione di queste. Tali chiavi, una privata ed una pubblica, dovranno essere in possesso rispettivamente dell'utente e del server, che le utilizzerà per verificare l'identità dell'utente, senza richiedere ulteriori credenziali (tranne la passphrase, se eventualmente utilizzata).

OpenSSH è un'implementazione open source di una serie di strumenti atti a stabilire connessioni cifrate tra più host, sfruttando il protocollo SSH. Nel contesto dell'applicazione che si vuole implementare, il server sfrutta tali strumenti per ricevere informazioni dal client, che può connettersi autenticandosi con username e password.

Al fine di utilizzare questo protocollo di comunicazione, è necessario munire il client di uno strumento che fornisca le funzionalità necessarie a stabilire una connessione cifrata SSH. Esistono diverse librerie Java che implementano tale protocollo. Il sito ufficiale di OpenSSH, tuttavia, ne cita una: *Ganymed SSH-2*. Tale libreria fornisce un'interfaccia semplice, che consente di creare una sessione dopo avere stabilito una connessione attraverso un'autenticazione via username e password, o utilizzando la logica della coppia di chiavi pubblica e privata. Le classi principali sono *Connection*, che rappresenta la connessione vera e propria, e *Session*, che implementa la sessione cifrata, e consente di eseguire comandi remoti. Le funzionalità fornite da Ganymed SSH-2 sono molte, ma l'utilizzo delle classi Connection e Session sarà sufficiente ai fini dell'implementazione del sistema di videosorveglianza in oggetto.

7. Apache HTTP Server e PHP

Apache HTTP Server, anche noto semplicemente come *Apache*, è il nome della più utilizzata piattaforma server Web, ed è compatibile con i sistemi operativi Linux e Windows. Esso consente di rendere accessibile il server tramite messaggi HTTP, è configurabile tramite opportune modifiche di un file, chiamato *httpd.conf*, e consente di utilizzare alcuni strumenti di sicurezza, per limitare l'accesso a determinati files.

PHP (acronimo ricorsivo che sta per *PHP: Hypertext Preprocessor)* è, invece, un linguaggio di scripting interpretato, la cui sintassi ricorda in parte i linguaggi C, Java e Perl. Viene utilizzato prevalentemente nella programmazione web, ma nulla vieta di utilizzarlo anche da riga di comando. Sebbene inizialmente presentasse alcune falle di sicurezza, le nuove versioni hanno visto significativi progressi da questo punto di vista.

Senza approfondire troppo la comprensioni di questi strumenti, ci si limita a spiegare il perché di tale scelta.

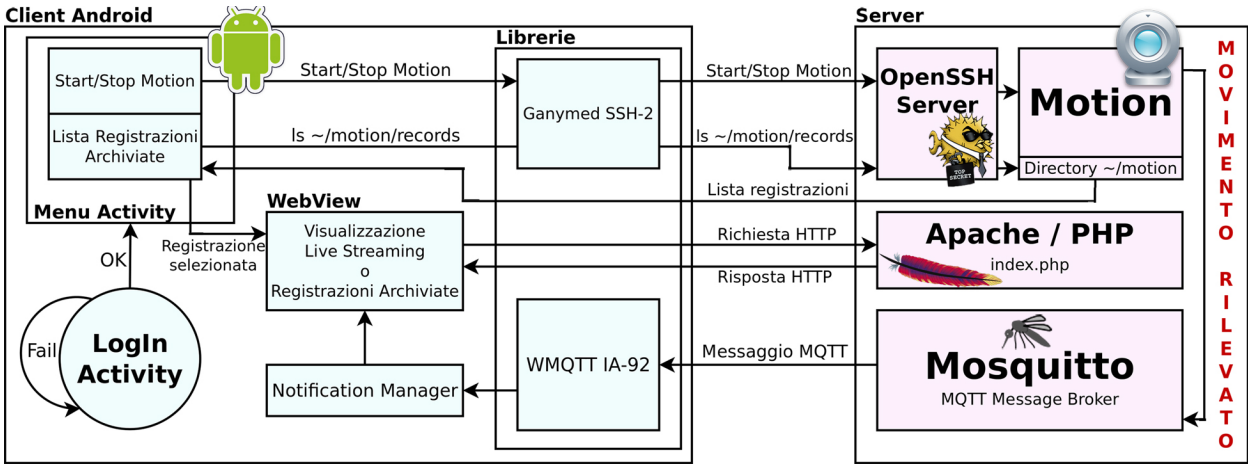

Figura 5 – Schema logico riepilogativo i funzionamento dell'applicazione. Dopo avere effettuato il login, l'utente, tramite l'activity del menu, può decidere se avviare/fermare l'esecuzione di Motion sul server, o eventualmente decidere se visualizzare una delle registrazioni archiviate. Ognuna di queste azioni viene eseguita sul server via SSH tramite la libreria Ganymed SSH-2. Una volta avviato Motion, alla rilevazione di un movimento sarà inviato un messaggio al message broker, Mosquitto, che lo inoltrerà all'applicazione Android. Per riceverlo, quest'ultima utilizza la libreria WMQTT IA-92 e, all'arrivo del messaggio MQTT, il gestore delle notifiche avviserà l'utente, offrendo la possibilità di visualizzare lo streaming in diretta

La loro utilità, infatti, è centrata sulla realizzazione di uno *streaming simulato* via HTTP. Tramite l'accesso ad una pagina web, infatti, sarà possibile visualizzare ciò che accade in tempo "quasi reale". In realtà, il frame rate che si ottiene da questo tipo di soluzione è molto basso; l'implementazione di questa funzionalità è da migliorare, ma la scelta adoperata fornisce una soluzione comunque accettabile per il raggiungimento degli scopi prefissati.

8. Funzionamento dell'applicazione

Prima di approfondire i dettagli implementativi dell'applicazione, si effettua una panoramica generale di come questa dovrà funzionare.

L'obiettivo principale è quello di potere ricevere, in tempo quasi-reale, un avviso sullo smartphone nel caso in cui venga rilevato un movimento dalla webcam remota. Ottenuto questo avviso, l'utente dovrà potere utilizzare le immagini registrate, e visualizzarle direttamente nello smartphone.

Per prima cosa, è necessario prevedere una connessione al server, per consentire di inizializzare il demone di Motion, ed avviare quindi la registrazione. Tale connessione avverrà fornendo l'IP del server, un nome utente ed una password, che corrisponderanno alle credenziali di accesso dell'utente sul server, tramite il protocollo SSH.

Una volta effettuato l'accesso, l'utente avrà tre possibilità:

- avviare la registrazione
- fermare la registrazione
- visualizzare le registrazioni archiviate

Con la registrazione avviata, ogni volta che Motion individua un movimento, esso notificherà al client questo dato, tramite l'invio di un messaggio. Ricevuto l'allarme, l'utente potrà visualizzare le immagini in diretta.

Nel prossimo paragrafo sarà approfondito l'aspetto implementativo, focalizzando l'attenzione sulle classi e sui meccanismi adoperati.

9. Sviluppo dell'applicazione

Prima di descrivere l'applicazione sullo smartphone, è bene introdurre un breve schema delle directory utilizzate sul server. All'interno della cartella home dell'utente che utilizzerà l'applicazione, sarà presente una directory chiamata *motion*. Tale directory conterrà, a sua volta, tre sottodirectory:

- *~/motion/tmp*: la directory tmp conterrà tutti i files della registrazione in corso, ovvero le immagini dei fotogrammi, il video in formato flv ed un file di log generato all'inizio della registrazione;
- *~/motion/records*: qui saranno salvate tutte le registrazioni, opportunamente organizzate in directory, rinominate con la data e l'ora della registrazione, nel formato AAAAMMGG_hh:mm:ss;
- *~/motion/scripts*: tale directory conterrà gli script che saranno eseguiti alla fine e all'inizio della registrazione. In particolare, all'inizio di ogni registrazione verrà eseguito uno script che creerà un file di log ed invierà un messaggio MQTT allo smartphone, mentre alla fine della registrazione, i file della cartella tmp saranno spostati in una sottodirectory appositamente creata nella directory records.

Sulla base di questo schema saranno implementate le funzionalità dell'applicazione necessarie per ottenere, ad esempio, la lista delle registrazioni. Va, inoltre, aggiunto, che la directory ~/motion sarà utilizzata anche per ospitare un file, chiamato *phone.data*, contenente il codice identificativo dello smartphone. Questo servirà per l'autenticazione di accesso allo streaming.

La directory /var/www, utilizzata come base dal server Apache, conterrà tre link alle directory motion, record e tmp, rinominati rispettivamente *motion*, *img* e *live*. Tali link saranno utilizzati da una pagina php per accedere alle directory fondamentali del progetto.

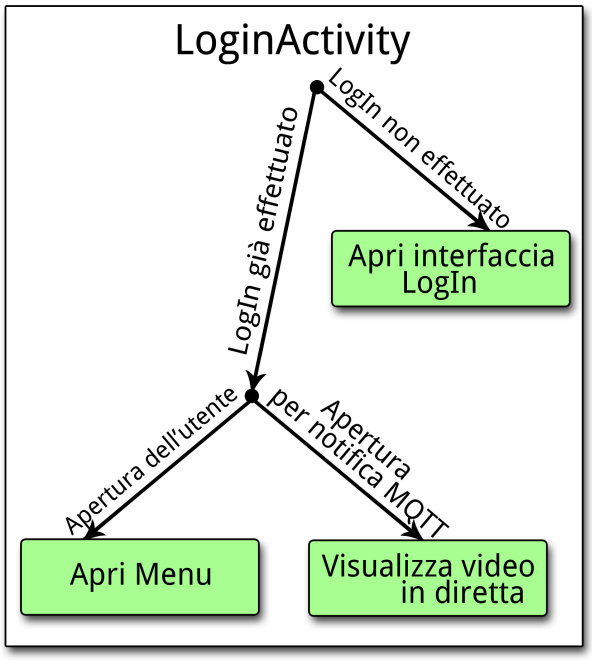

Figura 6. Comportamento pratico dell'activity del login. Dato che è possibile aprire l'applicazione soltanto tramite tale activity, sarà necessario verificare se è già stato effettuato in precedenza un login, o se è la prima volta che si apre l'applicazione. Nel primo caso, inoltre, dovrà essere verificato se è l'utente ad aprire l'applicazione dopo il login (ad esempio per visualizzare delle registrazioni archiviate, o per fermare la registrazione in corso), o se l'apertura è dovuta alla ricezione di un messaggio MQTT.

9.1. Le Activity principali

9.1.1. Login. Ogni applicazione Android, nel proprio file Manifest, deve specificare qual è la activity principale, cioè quella che verrà aperta quando l'utente esegue l'applicazione. In questo caso, l'activity in questione sarà quella del login. Questa scelta comporta che, all'apertura di tale activity, sarà necessario eseguire un controllo, per verificare se la si sta aprendo per la prima volta, o se l'apertura sta avvenendo dopo che l'utente ha già effettuato un login (e la ha poi richiusa, lasciando aperta la connessione). In quest'ultimo caso, è possibile che la richiesta di apertura dell'activity nasca dalla ricezione di un messaggio MQTT di allerta: se così fosse, l'activity del login dovrà subito consentire all'utente di visualizzare il video in diretta ottenuto dalla webcam remota. Questi controlli implicano che, nell'implementazione pratica, ci si discosterà da quanto descritto nella schema logico di figura 5, nel quale è rappresentato il solo caso del primo accesso. La figura 6 riassume, comunque, tali controlli.

Nel caso in cui il login è già stato effettuato, l'operazione che verrà eseguita consisterà nell'apertura di una nuova activity (il menu o la visualizzazione del video).

L'interfaccia grafica dell'activity in questione (figura 7) consente di immettere manualmente l'IP del server, lo username e la password per la connessione SSH, o eventualmente di scegliere da una lista di host conosciuti una terna IP/username/password già utilizzata e salvata. Se la connessione va a buon fine, la terna utilizzata verrà aggiunta alla lista degli host conosciuti.

Per effettuare il login, e verificare, quindi, la validità dei dati inseriti dall'utente, viene stabilita una connessione SSH con il server remoto, utilizzando le classi precedentemente descritte fornite dalla libreria Ganymed SSH-2. Stabilita la connessione, questa verrà subito chiusa. Se le operazioni di connessione e chiusura vanno a buon fine, l'activity del login aprirà il menu, descritto nel prossimo paragrafo.

9.1.2. Menu principale. Una volta effettuato il login, l'utente avrà accesso all'activity del menu principale. Da qui sarà possibile avviare la registrazione, fermarla o visualizzare tutte le registrazioni precedentemente archiviate. L'avvio e la fine della registrazione corrisponderanno all'esecuzione, via SSH, di due comandi, rispettivamente “motion” e “service motion stop”. Il primo, avvia motion in background sul server, il secondo ferma l'esecuzione del demone.

Figura 7. L'interfaccia grafica dell'activity del login. Ad ogni connessione riuscita viene salvato un record nelle preferenze condivise dell'applicazione, che consente, in seguito, la connessione rapida.

```
public static ArrayList<String> lsViaSSH(String ip,
String usr, String psw, String dir)
{
ArrayList<String> ls = new ArrayList<String>();

try {
       Connection conn = new Connection(ip);
       conn.connect();

       boolean isAuthenticated =
           conn.authenticateWithPassword(usr, psw);

       if (isAuthenticated == false) {
           return null;
       }

       Session sess = conn.openSession();
       sess.execCommand("ls -r " + dir);

       InputStream stdout =
          new StreamGobbler(sess.getStdout());
       BufferedReader br =
          new BufferedReader(
              new InputStreamReader(stdout));

       while (true) {
           String line = br.readLine();
           if (line == null)
               break;
           ls.add(line);
       }

      sess.close();
       conn.close();
    } catch (IOException e) {
           return null;
    }

    if(StringUtils.isEmpty(ls.get(0)))
       return null;

    return ls;
}
```

Figura 8. Funzione che esegue il comando ls via SSH. Utilizzando le classi e i metodi forniti dalla libreria Ganymed SSH-2, è possibile catturare l'output di un comando remoto

La visualizzazione delle registrazioni richiede di ottenere la lista delle sottodirectory contenute sul server in ~/motion/records, e per farlo verrà utilizzata un'apposita funzione, che sfrutta le classi fornite dalla libreria Ganymed SSH-2. Il codice Java di tale funzione (chiamata *lsViaSSH*) è mostrato in figura 8. Una volta ottenuta la lista delle registrazioni, questa verrà passata all'activity successiva, che sarà avviata per consentire all'utente di scegliere cosa visualizzare.

9.1.3. Visualizzazione delle registrazioni. Quando viene richiesto di visualizzare le registrazioni archiviate, l'utente ne ottiene l'elenco. Selezionando un elemento di tale elenco, verrà passata l'informazione della scelta ad un'altra activity, che servirà a visualizzare la pagina web che simulerà lo streaming.

Tale pagina web verrà visualizzata all'interno di una WebView (un browser incorporabile all'interno di una activity). Essa controlla innanzitutto se l'ID del telefono che sta visualizzando lo streaming è lo stesso di quello che ha avviato la registrazione (sfruttando il file phone.data). Fatto ciò, visualizza in sequenza le foto contenute nella directory relativa alla registrazione, simulando i frame di un video, tramite un refresh in Javascript. Tale soluzione non garantisce un frame rate elevato, ma il risultato che si ottiene è comunque accettabile.

Al fine di consentire, alla ricezione di un messaggio MQTT di allarme, la visualizzazione delle immagini in diretta, è possibile sfruttare un procedimento simile, ed includere questo caso all'interno della stessa pagina web, così da utilizzare la stessa activity. Questa volta, però, per visualizzare le immagini in diretta, sarà necessario utilizzare quelle salvate nella cartella ~/motion/tmp, della quale è presente il link in /var/www/live. La pagina php si aggiornerà tutte le volte che sarà presente una nuova immagine.

9.2. PushService: le notifiche push

Oltre alle activity, che consentono l'interazione diretta con l'utente, una delle parti più importanti dell'applicazione è un service, chiamato *PushService*, che consente di generare delle notifiche alla ricezione di un messaggio MQTT. Per descriverne il funzionamento, è necessario capire come inviare il messaggio MQTT di allarme dal server allo smartphone.

9.2.1. Invio di un messaggio MQTT. Le figure 3 e 4 mostrano il funzionamento del message broker MQTT. Tuttavia esse non lasciano trapelare un particolare: sebbene il message broker deve stare su un server, non è detto che questo server non possa comportarsi da client, che può sottoscriversi a topic e pubblicare messaggi come tutti gli altri host connessi al message broker. Questa caratteristica è proprio quella che verrà sfruttata.

L'installazione di Mosquitto consente al server di pubblicare messaggi su un qualsiasi topic, utilizzando un'interfaccia a riga di comando, con la seguente sintassi:

```
$ mosquitto_pub -m "contenuto del messaggio"
        -h IP_MESSAGE_BROKER -p PORTA
        -t topicA/topicB/topicC -q QoS
```

Utilizzare tale comando, settando l'ip del message broker a 127.0.0.1 (cioè localhost), consente al server di pubblicare un messaggio su un qualsiasi topic.

Motion, all'interno del file di configurazione, nella sezione “EXTERNAL COMMANDS”, consente di specificare un comando da eseguire all'inizio della rilevazione di un movimento. Tale comando sarà proprio quello appena visto. Ma su quale topic dovrà essere pubblicato tale messaggio?

9.2.2. Ricezione di un messaggio MQTT. Non appena l'utente avvia la registrazione, lo smartphone invierà via SSH un file contenente al suo interno un ID alfanumerico, generato tramite un metodo della classe *Secure* delle API di Android. Tale ID costituirà il nome del topic a cui il client si sottoscriverà, e sul quale il server pubblicherà gli eventuali messaggi di allarme.

Avviata la registrazione, verrà avviato anche PushService, che consentirà di ricevere i messaggi MQTT. Per prima cosa, tale service sottoscriverà lo smartphone al topic sopra citato. Al fine di ricevere i messaggi, verrà utilizzata la sintassi fornita dalla libreria WMQTT IA92, ed in particolare sarà definita una classe privata all'interno di PushService, che implementa l'interfaccia *MqttSimpleCallback*. Tale interfaccia necessita dell'implementazione, tra gli altri, di un metodo, chiamato *publishArrived*, che verrà invocato ad ogni ricezione di un messaggio. Tale metodo, al suo interno, ne invoca un altro chiamato *showNotification*. Questo si occuperà, sfruttando un gestore delle notifiche, di mostrare all'utente le notifiche di ricezione dei messaggi.

PushService, inoltre, implementa un meccanismo di controllo che riguarda la possibilità che il sistema operativo termini forzatamente il service. Tale evenienza può verificarsi qualora il S.O. necessiti di memoria. Se viene terminato, il service viene riavviato in modo pulito, assicurando la corretta valorizzazione delle variabili.

9.3. Dettagli aggiuntivi

9.3.1. Layout e stringhe. Android offre una duplice possibilità per strutturare il layout grafico di una activity. È possibile, infatti, creare direttamente nel codice Java tale layout, utilizzando opportune View. Tale soluzione, tuttavia, è sconsigliata dal team di sviluppo di Android, in quanto è possibile definire tali schemi attraverso l'utilizzo di un file di layout in formato XML. Questo consente di separare il codice dal layout, ed allo stesso tempo di facilitare la modifica della visualizzazione grafica senza la necessità di toccare direttamente il programma vero e proprio. Nel caso d'interesse, sono stati utilizzati diversi file di layout, uno per ogni activity. A questi vanno aggiunti altri due file XML che descrivono il layout delle notifiche Toast (messaggi di avviso che scompaiono dopo pochi secondi; tale funzionalità è fornita direttamente con le API di Android) e il layout che definisce la struttura di ogni riga dell'elenco delle registrazioni archiviate.

Un altro importante file XML, utile, ad esempio, per arricchire l'applicazione del supporto multi-lingue, è il file *strings.xml*, che contiene la definizione di tutte le stringhe visualizzate nelle varie parti del programma. Tramite la funzione *getString*, ereditata da tutte le activity, è possibile ottenere con facilità tali informazioni. Tradurre tali stringhe significa tradurre l'intera applicazione.

9.3.2. File *Manifest.xml*. Dato il principio del minimo privilegio implementato da Android, e dato che ogni applicazione su Android è un utente, è necessario fornire al programma i privilegi di cui necessita per potere funzionare correttamente. A tale scopo, viene utilizzato il file *Manifest.xml*, che contiene sia la descrizione dell'applicazione in termini di activity e service, sia i permessi che l'applicazione deve avere. Al momento dell'installazione, l'utente sarà informato e deciderà se accettare o meno di eseguire il software sullo smartphone, conoscendo i permessi che vi sono affidati.

Nel caso specifico del sistema di videosorveglianza, i permessi necessari sono quelli relativi all'utilizzo di *internet*, alla possibilità di leggere lo *stato della rete* e alla capacità di fare *vibrare* lo smartphone.

10. Possibili sviluppi dell'applicazione

Prima di concludere, è bene precisare che il lavoro svolto è finalizzato principalmente ad illustrare le caratteristiche principali di un sistema di videosorveglianza per Android, fornendo le basi per un possibile sviluppo. Molti aspetti dell'applicazione presentata risultano migliorabili, e le potenzialità dei protocolli MQTT ed SSH, oltre che le funzionalità offerte dall'utilizzo di Motion e Mosquitto, consentono di immaginare possibili sviluppi del sistema finora presentato. Nel seguito se ne elencano alcuni.

- *Connessione a più client*: il protocollo MQTT consente a diversi client di sottoscriversi ad uno stesso topic. È dunque facile intuire che più smartphone potrebbero sottoscriversi ad uno stesso topic, per ricevere contemporaneamente un eventuale messaggio di allarme.
- *Utilizzo di più webcam*: Motion offre la possibilità di utilizzare più di una webcam, e di definire, per ognuna delle videocamere connesse al server, una particolare configurazione. Ciò consente non soltanto di settare dettagli grafici ottimizzati per ogni webcam, ma anche di eseguire comandi “ad hoc” per il salvataggio delle immagini in una struttura di directory più articolata.
- *Sicurezza migliorabile*: sebbene l'utilizzo di SSH fornisca connessioni cifrate, resta ancora da migliorare l'accesso allo streaming tramite la richiesta HTTP. Attualmente viene passato all'interno dell'URL (metodo GET), in chiaro, l'ID dello smartphone, che funge anche da password di accesso allo streaming. Inoltre, lo stesso URL conterrà l'IP del server. Questa funzionalità è certamente migliorabile: per fornire un esempio, utilizzando HTTPS sarebbe possibile inviare messaggi HTTP utilizzando un canale cifrato, e quindi sicuro.
- *Streaming migliorabile*: la soluzione proposta per lo streaming è molto rudimentale. Tuttavia, per produrre soluzioni più efficaci, è possibile utilizzare applicazioni Android esterne che si occupano di riprodurre file flv (Flash Video) in streaming, richiedendo eventualmente l'installazione di un server di streaming sull'host remoto.

11. Riferimenti bibliografici

[1] “Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012”, http://www.gartner.com/it/page.jsp?id=1622614

[2] “MQ Telemetry Transport – Documentation”, http://mqtt.org/documentation

[3] “Motion - Web Home”, http://www.lavrsen.dk/foswiki/bin/view/Motion/WebHome

[4] “Documentation – Mosquitto”, http://mosquitto.org/documentation

[5] “Android Developers”, http://developer.android.com

[6] “Ganymed SSH-2 for Java – Documentation”, http://www.cleondris.ch/opensource/ssh2/javadoc

[7] “WMQTT IA92 Java utility”, http://mqtt.org/wiki/ia92

[8] “The Deacon Project”, http://deacon.daverea.com

[9] “OpenSSH”, http://www.openssh.com

[10] “Android Cloud to Device Messaging Framework”, http://code.google.com/intl/it/android/c2dm

[11] “The XMPP Standards Foundation”, http://xmpp.org