UNIVERSITÀ DEGLI STUDI DI TRENTO
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea (triennale) in Informatica

Elaborato Finale

Progettazione e sviluppo di un sistema di video streaming per cellulari: applicazione in ambito domotico

Relatore: Prof. Paolo Giorgini

Laureando: Maurizio Bonani

Anno Accademico 2008 - 2009

“Ringrazio tutti coloro che mi sono stati vicino e che mi hanno sostenuto durante il percorso per il raggiungimento di questo importante traguardo”

“La mente è come un paracadute, funziona solo se si apre”

Albert Einstein

“L’uomo più saggio non è colui che sa,
ma chi sa di non sapere”

Socrate

“Il problema dell'umanità è che gli sciocchi e i fanatici sono estremamente
sicuri di loro stessi, mentre le persone più sagge sono piene di dubbi.”

“Non morirei mai per le mie opinioni: potrei avere torto”

Bertrand Russell

“La disumanità del computer sta nel fatto che, una volta programmato e messo in funzione,
si comporta in maniera perfettamente onesta.”

Isaac Asimov

INTRODUZIONE

Negli ultimi anni si è assistito alla forte espansione della scienza dell'informazione, fino a coprire campi di applicazione che prima si pensavano naturalmente incompatibili. Si pensi soltanto agli sviluppi in campo medico o biologico che hanno portato la conoscenza umana al di là di barriere che, fino a pochi decenni fa, sembravano invalicabili.

Al giorno d'oggi l'informatica presenta contorni poco definiti, nonostante sia un termine ormai largamente utilizzato. Come scienza si accompagna, si integra o è di supporto a tutte le discipline scientifiche e non; come tecnologia pervade ormai qualunque "mezzo" o "strumento" di utilizzo comune e quotidiano. E' facile immaginare, a questo punto, l'impiego di un dispositivo informatico in qualsiasi ambito digitale. Per avere un'idea più precisa del significato concreto di questi termini, è utile ricordare le parole di *Robert Metcalfe*, inventore di Ethernet e fondatore di 3Com Corporation, che nel 2001 disse:

> *"Quest'anno verranno prodotti 8 miliardi di microprocessori, di cui solo il 2% diventeranno pc. La restante parte verrà utilizzata per fabbricare tutti quei dispositivi che fanno ormai parte della nostra vita e che, nella maggior parte dei casi, non pensiamo nemmeno siano computer."*

Soffermiamoci un attimo a pensare, per esempio, all'interno della nostra abitazione quali categorie di strumenti elettronici potrebbero essere arricchite di una parte intelligente. L'impianto elettrico, che potrebbe regolare automaticamente i carichi per evitare spiacevoli blackout; il sistema di sicurezza, il quale sarebbe in grado di rilevare, in modo autonomo, la presenza di intrusioni e di notificarle appropriatamente; l'impianto del gas, che potrebbe bloccare le valvole di erogazione nel caso venisse rilevata una minima presenza di incendio. Anche gli elettrodomestici non sono esenti a questa tendenza. L'aggiunta di una parte intelligente permetterebbe loro di ottimizzare l'utilizzo e fornire importanti informazioni all'utente, evitando sprechi e garantendo sempre la massima efficacia.

Il passo successivo potrebbe essere l'interconnessione di tutte questi dispositivi, per permettere un integrazione totale dei dispositivi domestici, fino al collegamento degli stessi su Internet. Si pensi alle possibilità e ai vantaggi di controllare e gestire la propria casa da qualunque luogo, magari con l'ausilio di uno strumento diffusissimo quale il telefono cellulare.

> *"Come l'elettricità ha modificato l'architettura, così la rivoluzione informatica sta modificando gli impianti […] ma anche l'essenza stessa dell'architettura e la sua funzione."* [1]

Scenari futuristici? Lusso per pochi? Assolutamente no. Questa piccola rivoluzione è già in atto. L'evoluzione delle varie tecnologie di recente concezione si sta sviluppando in una direzione che coincide con il termine "integrazione". La convergenza digitale dei sistemi di controllo, comunicazione e il software di interfaccia hanno un comune denominatore, chiamato domotica.

La domotica è una scienza interdisciplinare di più recente interesse rispetto al networking e al mobile computing. Sostanzialmente è orientata a migliorare la qualità della vita negli ambienti antropizzati. Essa è considerata dai più una eccezione, un vizio, addirittura un eccesso per chi ama le comodità e può permettersi di spendere per automatizzare i processi e il controllo della propria casa. Questa considerazione è un errore di valutazione. Un'attenta analisi costi-benefici, se fosse fatta in maniera accorta, non darebbe adito a coloro che la pensano in questo modo. Se prendiamo in considerazione un arco temporale medio di vita all'interno di una stessa proprietà immobiliare, potremo constatare che il risparmio scaturito dal controllo automatizzato delle risorse della casa a lungo andare permette di recuperare oltre misura la spesa necessaria per installare un sistema di domotica, che oltre al risparmio è una comodità.

Ulteriore scopo di questa tesi, oltre all'approfondimento di una scienza che, nonostante alcune diffidenze, sembra portare a nuovi standard nelle nostre abitazioni, è la ricerca di una risposta al seguente quesito: è possibile utilizzare le tecnologie a nostra disposizione per creare qualcosa di nuovo e diverso dallo scopo principale per cui queste stesse tecnologie sono state pensate? Questa era la domanda essenziale che ci siamo posti nell'intraprendere questa sfida e a cui avremmo dovuto dare risposta. Come premessa ci siamo imposti di puntare su tecnologie a basso costo e facilmente reperibili sul mercato.

La scelta è ricaduta sulle webcam e sui telefoni cellulari. Entrambe le categorie sono nate con lo scopo di favorire la comunicazione tra le persone, entrambe hanno costi più o meno contenuti ed entrambe nascondono potenzialità che vanno ben oltre gli utilizzi principali per cui sono state pensate. La sfida consiste proprio nel ricercare e sfruttare questo potenziale.

L'idea iniziale alla base del progetto era, quindi, quella di fondere tecnologie moderne, al fine di realizzare un sistema di video streaming a basso costo, utilizzando delle comuni webcam e un telefono cellulare dotato della tecnologia Java. Il sistema avrebbe quindi permesso di catturare le immagini dalla webcam e trasmetterle, successivamente, ad un telefono cellulare che ne avrebbe fatto richiesta.

Un sicuro campo di applicazione è certamente il settore della video-sorveglianza. La potenzialità di visionare gli ambienti della propria abitazione da remoto spalanca le porte ad un nuovo concetto di casa.

"Home is no longer a location, home is a concept.
I want the feelings of home wherever I am."

Watts Wacker
(CEO, Futurist
First Matter LLC)

Man mano che il sistema prendeva forma si è poi aperta l'opportunità di integrarlo all'interno di un ambiente domotico, espandendo ulteriormente gli ambiti di applicazione. In particolare si è potuto usufruire del laboratorio domotico in Via della Cascata a Povo e delle tecnologie presenti al suo interno. Si è cercato di andare oltre al concetto di visualizzazione remota degli ambienti, conferendo al telefono cellulare un ruolo ulteriormente attivo all'interno del sistema. La direzione in cui si è svolto il lavoro ha portato alla possibilità di controllare, in modo relativamente semplice, dispositivi da remoto. Tramite una scheda a relé in dotazione al laboratorio è stato quindi possibile comandare da distanza l'accensione e lo spegnimento di apparecchiature elettriche, sempre utilizzando un dispositivo mobile.

Al centro di tutto il sistema c'è il concetto di telefono cellulare, inteso non solo come un potente strumento di comunicazione, ma anche come strumento di gestione universale, simile ad un telecomando, ma con la fondamentale differenza che, utilizzando Internet e i moderni sistemi di connessione, questo telecomando può essere usato in qualsiasi luogo e non solo all'interno della propria abitazione. I vantaggi e gli utilizzi di queste tecnologie sono, a mio parere, infiniti.

La seguente tesi descriverà le tecnologie utilizzate e il sistema progettato. Non vuole assolutamente essere un esempio di progetto completo, né tanto meno una proposta di standard da utilizzare, lo scopo è dimostrare la relativa semplicità di creare, con le tecnologie che abbiamo oggi a disposizione, delle applicazioni in grado di integrarsi con l'ambiente domestico in cui viviamo, permettendo meccanismi di gestione e controllo.

STRUTTURA DELLA TESI

La tesi si è sviluppata sostanzialmente in due direzioni che hanno dato origine a due progetti separati, ma fortemente integrabili.

La prima parte, compresa nell'ambito della video-sorveglianza, aveva come obiettivo la progettazione di un sistema software in grado di interfacciarsi con una comune webcam e di inviare il flusso video ad un telefono cellulare collegato.

La seconda parte, invece, aveva come obiettivo la definizione di un ruolo attivo del cellulare all'interno di un ambiente domotico. Partendo dall'applicativo sviluppato nella prima parte, si è progettato un sistema in grado di integrarsi con delle telecamere IP e con un software di gestione domotica, precedentemente sviluppato da un altro studente, chiamato EventEngine.

La tesi è costituita da 5 capitoli:

Capitolo 1: viene fornita una descrizione generale sulla domotica, le origini, i vantaggi, le caratteristiche e gli sviluppi futuri. Viene dato spazio anche all'importante ruolo del telefono cellulare e alla piattaforma cardine per lo sviluppo di applicazioni in ambito mobile: Java Micro Edition.

Capitolo 2: viene presentata una panoramica generale sul sistema che si intende realizzare e sugli obiettivi che si intende raggiungere. Vengono inoltre sottolineate le scelte tecnologiche per la realizzazione di questi obiettivi. Infine sono presentati i possibili scenari d'utilizzo

Capitolo 3: materia di questo capitolo è l'analisi dei requisiti e la progettazione del sistema. In particolare si esaminano le funzionalità e l'architettura del sistema utilizzando i diagrammi dei package e delle classi.

Capitolo 4: è qui presentato il primo progetto, denominato MotionDomus. Vengono descritti il funzionamento, le caratteristiche e l'implementazione.

Capitolo 5: viene presentato il secondo progetto, denominato ControlDomus. Questa parte comprende anche la trattazione del client per telefono cellulare.

Capitolo 1

DOMOTICA E STATO DELL'ARTE

1.1 La Domotica

Il progetto si inquadra all'interno del crescente panorama della domotica, una scienza interdisciplinare orientata a migliorare la qualità della vita negli ambienti antropizzati [2]. Il termine domotica[i] è un neologismo che deriva dal francese "domotique", a sua volta contrazione della parola latina "domus" (casa, o dal greco "domos", con lo stesso significato) e di "automatique" (automazione), un sinonimo è automazione domestica o home automation.

La domotica si occupa dell'integrazione dei dispositivi elettronici, degli elettrodomestici e dei sistemi di comunicazione e di controllo che si trovano all'interno delle nostre abitazioni.

Nel corso degli anni si è assistito sempre di più al proliferare di dispositivi elettronici all'interno degli ambienti domestici. Ai più comuni elettrodomestici, che consentono di aiutare l'utente nelle attività di tutti i giorni (frigorifero, forno, lavatrice…), si sono aggiunti dispositivi il cui fine principale è intrattenere e divertire l'utilizzatore. Sono comparsi sul mercato, quindi, televisori interattivi, decoders, consolle di gioco, lettori multimediali… Infine, in un ruolo che possiamo tranquillamente definire "cardine", in quanto è compreso in entrambe le categorie, c'è il Personal Computer. Sicuramente il fenomeno più appariscente di questa tendenza.

La sfida è quindi quella di aggiungere intelligenza e funzioni di comunicazione a tutti i dispositivi alimentati da elettricità, in modo che questi strumenti digitali possano scambiarsi dati e servizi. La domotica estende il concetto stesso di integrazione ed automazione degli apparati elettronici presenti in una casa, o addirittura in un edificio, fino alla connessione degli stessi in Internet.

Trova le sue radici negli anni '70 quando cominciarono ad essere studiati e realizzati i primi progetti che permettevano l'interconnessione di alcuni sistemi come l'impianto di illuminazione e quello di sicurezza, dando all'utente la possibilità di automatizzarne alcune funzionalità.

La tendenza attuale è di arricchire sempre più gli elettrodomestici e gli impianti di casa di una parte elettronica "intelligente", che permetta già ora a tali apparecchiature di riconoscere una determinata serie di eventi o situazioni e di autoregolarsi di conseguenza.
Il prossimo passo sarà quello di avere tutte le apparecchiature domestiche compatibili tra loro e con la possibilità di controllarle e programmarle in modo coordinato attraverso un'unica stazione di input (ad esempio un Personal computer o un interfaccia a muro) e quindi anche a distanza, tramite reti telefoniche e dati. Un'immagine emblematica di questa tendenza è mostrata in figura 1.1 e rappresenta alcune delle componenti che possono trovare una convergenza comunicativa all'interno dell'abitazione.

"Ogni decade ha una parola che la contraddistingue. Negli anni '80 era PC. Nei '90 era Internet. Questa decade la parola chiave è convergenza."

Steve Case (AOL TimeWarner)

Di primaria importanza è l'integrazione, in quanto integrare vuol dire trasformare la sommatoria dei singoli prodotti in un sistema omogeneo, rendere le diverse parti reciprocamente compatibili, sia dal punto di vista fisico che da quello logico, uniformare il cablaggio e le tecniche di trasmissione e comunicazione.

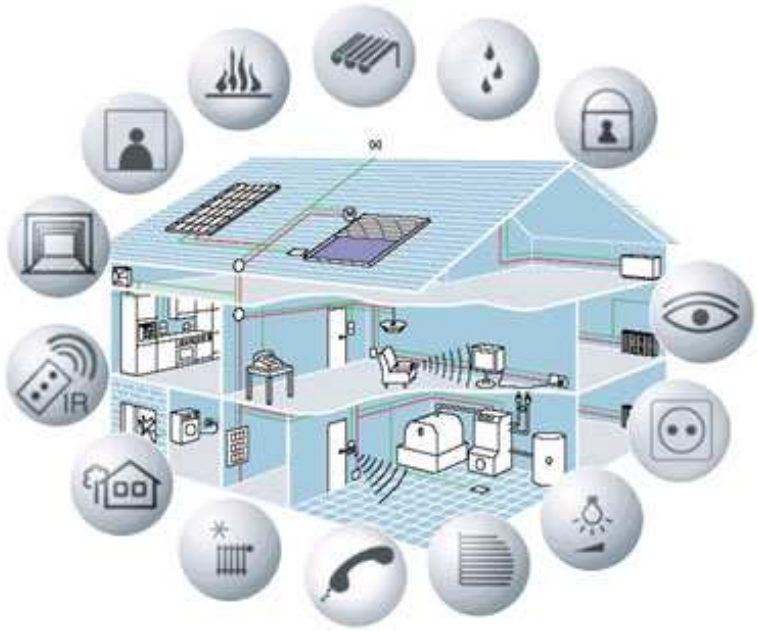

Figura 1.1: rappresentazione schematica dell'automazione domestica

Il vantaggio di un sistema integrato è tanto maggiore quanto prima si inserisce nella progettazione. Utilizzando le capacità di gestione e programmazione di un sistema integrato, si possono risparmiare molti costi legati all'inserimento di funzioni in ciascun impianto.

1.2 Scenari attuali

Cosa può offrire l'automazione domestica?

La casa domotica non è solo un sistema chiuso i cui vantaggi hanno come unico destinatario il nucleo familiare, ma, come tutte le nuove tecnologie, non può non incrociarsi con altre attese tecnologiche, creando nuove prospettive di notevole interesse sociale.

Gli ambiti principali in cui opera la domotica e che dimostrano prospettive e applicazioni promettenti sono il risparmio energetico, il comfort ambientale, la sicurezza, , il controllo degli elettrodomestici, la gestione dell'entertainmennt, i teleservizi.

1.2.1 Domotica e risparmio energetico

La gestione automatizzata degli elettrodomestici consente un tangibile risparmio di energia elettrica e un più sicuro funzionamento degli stessi carichi. Questa è diventata ormai una necessità per l'aumento esponenziale dei dispositivi elettrici avutosi negli ultimi anni nelle nostre abitazioni e il conseguente potenziale consumo di picco che può manifestarsi, provocando black-out e danni reali all'impianto. Il risparmio tangibile per il consumatore può derivare, come già succede in America e in alcuni paesi dell'Europa (con al primo posto la Francia[ii]), da politiche di fasce orarie con costi energetici differenziati. Il sistema demotico può quindi farsi carico, ad esempio, di accendere la lavatrice la sera o la notte.

L'integrazione di sensori di presenza con il controllo delle luci e del riscaldamento/condizionamento porta all'ottimizzazione delle risorse energetiche, adattando indipendentemente la temperatura dell'ambiente alle varie stanze ed illuminandole alla effettiva presenza di persone al loro interno. L'integrazione degli stessi sensori con l'impianto elettrico può portare, inoltre, a una riduzione dei campi magnetici nelle stanze dove c'è la presenza di persone.
L'introduzione di sensori ambientali situati all'esterno dell'abitazione porta all'attivazione dell'impianto di irrigazione del giardino o delle piante non solo in fasce orarie predefinite, ma soprattutto quando vi è una reale necessità, comportando un notevole risparmio di acqua.

L'introduzione della domotica con le sue ampie possibilità di controllo e gestione automatizzata è probabilmente un'ottima occasione per il rilancio, già in corso tutt'ora[iii], di un uso diffuso dell'energia solare, e in generale degli apporti gratuiti, in ambiente domestico.
Ciò è dovuto al fatto che il pannello solare, per ottimizzare il proprio contributo energetico, ha bisogno di regolazione continua perché, a seconda delle condizioni esterne, produce acqua con temperature diverse, a volte superiori e a volte inferiori, rispetto a quella già in circolo e prodotta dall'impianto base.
Per trarne reale beneficio l'utente dovrebbe interrompere il flusso dell'acqua "solare" quando la sua temperatura è inferiore a quella desiderata, immagazzinarla quando è molto calda e in eccesso, aggiungerla a quella prodotta dall'impianto base interrompendo il funzionamento di quest'ultimo: in altri termini dovrebbe essere occupato costantemente a seguire il funzionamento del suo impianto di produzione di acqua calda. Un sistema domotico può farsi carico di gestire tutto il processo, ottimizzando la resa totale del sistema.

1.2.2 Domotica e comfort

Il sistema preposto all'automazione della casa gestisce tutto quello che può influire sui parametri ambientali modificandone gli assetti in funzione delle esigenze dell'utente e adeguandoli costantemente senza una continua richiesta di input.
La casa domotica può coordinare il sistema di riscaldamento o condizionamento con il sistema di ventilazione o con l'apertura o la chiusura delle finestre per mantenere all'interno di ciascuna stanza la temperatura e l'umidità richieste dall'utente con qualsiasi tempo e in qualsiasi stagione dell'anno. Se l'utente lo desidera, nei mesi invernali, mantiene la temperatura a livelli inferiori alla norma abituale quando l'alloggio non è abitato, e la rialza portandola ai gradi voluti prima del rientro degli occupanti.
Anche l'impianto d'illuminazione è coordinato in maniera da accendere le luci al passaggio di un utente e di spegnerle una volta che è passato.

1.2.3 Domotica e sicurezza

Esistono due tipi di sicurezza in relazione alla casa e alla vita che si svolge all'interno di essa. Questi tipi sono correttamente definiti dalle parole in lingua inglese "*security*", ad indicare la protezione da effrazioni ed intrusioni, e "*safety*" ad indicare la protezione da ogni forma di incidente domestico. La casa domotica ha il compito di incrementare le due forme di sicurezza, salvaguardando l'utente da potenziali situazioni di incidente che lo possano danneggiare.

Security

Prevenire il furto e l'intrusione nella propria abitazione è una delle esigenze primarie dell'uomo, secondo il concetto ormai ancestrale di casa come protezione dall'esterno.
Il sistema domotico dotato di telecamere e sensori di movimento, localizzati sia internamente che esternamente, rileva qualsiasi presenza estranea all'interno del perimetro abitativo. A questo si

aggiungono sensori di rilevamento dell'audio, in modo da reagire al superamento di un livello di soglia di tipo volumetrico. Le possibilità di reazione ad una intrusione sono molteplici e possono essere propriamente combinate: sirene di allarme, accensione delle luci, invio di un segnale di allarme telefonico a dei numeri preimpostati, invio di sms ai proprietari.

Safety

Ogni anno avvengono in Italia oltre 4 milioni di infortuni in ambiente domestico, con 8.000 casi mortali[iv]; un numero enorme di casi che colpisce prevalentemente la popolazione femminile. In particolare la "collettività" delle casalinghe esprime circa il 40 per cento del totale.
Il 15 per cento circa degli infortunati in casa subisce più di un incidente nell'arco dello stesso anno.
Nonostante le età estreme – e pertanto più esposte – risultino le più colpite, la maggior parte dei casi riguarda fasce di popolazione in età lavorativa. Tale quadro colloca gli infortuni in ambiente domestico, con gli incidenti stradali e quelli sul lavoro, in posizione di massima evidenza.
"*Il consumatore è divenuto sempre più cosciente dei pericoli di una presa non ben isolata, di eventuali fughe di gas, delle possibilità di incendio e dei danni economici che può procurare una lavatrice che perde o un accidentale allagamento*" [3]
E' richiesta, quindi, una attenzione sempre maggiore nei confronti degli incidenti domestici e della sicurezza nell'uso delle apparecchiature della casa. Per garantire questo tipo di sicurezza il sistema si comporta disattivando un elettrodomestico, togliendo l'acqua, spegnendo il gas al rilevamento di fuga e segnalando nel modo appropriato il guasto o l'interruzione del servizio, sia agli occupanti della casa, sia ai centri d'assistenza tecnica, in modo da ottimizzare i tempi di riparazione. Tale caratteristica è chiamata autodiagnosi.

1.2.4 Domotica e assistenza

L'introduzione dei comandi vocali e di strumenti per la tele-assistenza rappresentano ormai un valido, se non insostituibile, aiuto per anziani, malati e portatori di handicap. A beneficiare di questi aiuti non sono solamente gli stessi malati ed anziani, che possono mantenere il contatto con la propria abitazione, ma anche le strutture sanitarie pubbliche che in questo modo evitano un pericoloso sovraffollamento, testimoniato dal costante innalzamento dell'età media della popolazione. Uno studio del CNR [4] stima infatti che nel 2025 gli ultraottantenni rappresenteranno quasi il 7% della popolazione italiana.

La connessione dell'impianto domestico con l'esterno permette di monitorare eventuali parametri vitali dando la possibilità di intervenire rapidamente in situazioni di emergenza tramite l'invio automatico di richieste di soccorso per via telefonica. L'utilità è evidente soprattutto quando l'utente è una persona anziana o un soggetto a rischio che vive da solo.

1.2.5 Domotica e entertainment

Negli ultimi tempi anche il settore dell'entertainment ha subito una serie costante di rivoluzioni, accostando ulteriormente al gran numero di televisori, videoregistratori e impianti hi-fi e collocati in stanze diverse dell'abitazione, strumenti moderni come lettori DVD e BluRay, Decoder del segnale digitale terrestre e di quello satellitare, stazioni multimediali di archiviazione… La situazione è ulteriormente complicata dal fatto che ogni dispositivo ha il proprio telecomando che, a meno di alcune eccezioni, possiede le stesse caratteristiche.
Si registra inoltre una particolare tendenza alla cospazaialità e alla cotemporalità delle funzioni, sottolineata dal cambiamento delle abitudini soprattutto le più banali e quotidiane. Ognuno vuole vedere il suo programma preferito o ascoltare la musica nei modi e nei momenti che maggiormente gli aggradano. Il sistema domotico capta i segnali dall'esterno o dall'impianto centrale

dell'abitazione e li distribuisce sui vari apparecchi, controlla il livello di ricezione e di trasmissione badando alla sintonia, alla luminosità dello schermo, alla qualità ed al volume dell'audio. E' consentita, per esempio, la fruizione di un unico lettore DVD da più terminali video della casa. E ancora, se l'utente è un amante della musica, può predisporre una serie di altoparlanti in modo da poter ascoltare il suo album preferito in qualsiasi punto della casa e può, da dove si trova, comandare l'accensione e gestire l'impianto hi-fì (regolarne il volume, risentire un passaggio o registrare un brano…).

1.3 Home automation, building automation

A questo punto si impone la distinzione relativa alla dimensione dello spazio preso in considerazione. Infatti le applicazioni di un sistema domotico sono in relazione ad uno specifico spazio vitale, che può assumere varie tipologie: isolato, servito, oppure collegato ad altri simili spazi in modo dipendente dai requisiti dell'utente.
La scala su cui si realizza l'automazione si realizza quindi su tre livelli di abitazione: la singola unità, il condominio ed infine il centro residenziale (come può esserlo un grande edificio o un intero quartiere). Nel primo caso quindi si parla di *"home automation"* mentre nei livelli superiori si parla di *"building automation"* o "*automazione degli edifici*". Gli obiettivi restano sostanzialmente gli stessi, ma anziché essere orientati al singolo abitante sono invece finalizzati a tutti i servizi che si possono considerare centralizzati, in quanto comuni.

1.4 Standard comunicazione

Un sistema di home automation, come detto, coordina i vari impianti presenti nell'edificio con lo scopo sia di gestirli tutti omogeneamente da una o più stazioni di supervisione, ma anche di farli interagire e aumentarne le prestazioni. In termini di tecnologia si tratta di trovare una convergenza nelle strutture di cablaggio, nei protocolli di comunicazione e nelle funzionalità.

Esistono attualmente numerosi standard utilizzati e, come è facile prevedere, uno dei fattori che ha contribuito a rallentare la crescita di questo settore è proprio l'incertezza su quale di essi prevarrà e diventerà lo standard di mercato.

Elenchiamo qui gli standard aperti più rilevanti:

1.4.1 Ethernet

E' probabilmente la più popolare tecnologia per le comunicazioni su rete locale. Lo standard è noto anche come IEEE 802.3 [5]. E' nato negli anni '70 per condividere e collegare tra loro, in modo semplice ed efficiente, dispositivi e risorse di rete. I protocolli utilizzati permettono un transito di dati con velocità comprese tra 1Mb/s (Ethernet tradizionale) ed 1Gb/s (Gigabit Ethernet), utilizzando cavi coassiali (ora poco usati), doppino intrecciato (cavi Cat. 5 e 6), fibra ottica.
Diffusosi enormemente anche grazie alla crescita di Internet, oltre che in utilizzi di tipo domestico e in uffici, ha avuto, grazie alla sua affidabilità e soprattutto ai costi contenuti, una buona diffusione anche in campo industriale dove è utilizzato per il controllo e monitoraggio di impianti e sistemi.
Ultimamente viene impiegato anche in campo domotico grazie alle sue doti di versatilità e al grande numero di dispositivi ed applicazioni disponibili sul mercato.

1.4.2 X-10

X-10 [6] è uno standard storico dell'automazione domestica, sviluppato nel 1976 da Pico Electronics Ltd, è presente ormai da vent'anni in America, ha avuto una larga diffusione anche in

Europa e sembra mantenere, grazie anche alle innumerevoli periferiche disponibili sul mercato, una salda posizione nel panorama dell'Home Automation.
Esistono anche alcune applicazioni commerciali nell'ambito della gestione energetica, ma prevalentemente viene utilizzato in ambiente domestico.
È costituito da un'unità centrale che invia dei comandi ai dispositivi periferici utilizzando un sistema ad onde convogliate su rete elettrica per la trasmissione dei dati.
Il sistema centrale può anche essere controllato da un PC mediante software, oppure pilotato a distanza grazie a telecomandi a infrarossi.

1.4.3 Bluetooth

Il consorzio Bluetooth [7] è stato fondato nel 1998 da grandi aziende quali Ericsson, Intel, IBM, Toshiba, Nokia. Nel dicembre 1998 si aggiunsero altri importanti gruppi: Microsoft, Lucent, 3COM, Motorola e comprende attualmente i maggiori produttori di apparecchiature di telefonia, elettronica ed informatica.
La tecnologia Bluetooth mette in comunicazione dispositivi elettronici anche diversi, come stampanti, personal computer, palmari, cellulari, fotocamere digitali… all'interno di uno spazio "personale". Questo standard di comunicazione consente infatti di creare una rete a corto raggio, senza cavi, definita "Personal Area Network".
Mediante un sistema radio che opera sui 2,4 GHz consente uno scambio di dati con velocità fino ad 1 Mb/s in un raggio massimo di alcune decine di metri.

1.4.4 KNX (Konnex)

KNX [8] è nato in seguito alla fondazione dell'associazione Konnex, avvenuta nel maggio del 1999 da parte di EIBA (European Installation Bus Association) [9], BCI (Batibus Club International) [10] ed EHSA (European Home System Association) con lo scopo di realizzare e promuovere uno "standard unico" per applicazioni di Home e Building Automation. E' basato sulle migliori caratteristiche delle tre tecnologie di partenza e, di fatto, sul cuore dello standard EIB, con cui mantiene una totale compatibilità; integra inoltre le modalità di configurazione e i mezzi trasmissivi di BatiBus ed EHS. I componenti, realizzati da costruttori diversi, vengono garantiti, dopo una procedura di certificazione operata dall'associazione Konnex, per essere interoperabili, cioè per funzionare correttamente senza necessità di realizzare interfacce.
KNX è stato pensato anche per diventare un marchio di qualità ed aiutare quindi i consumatori ad orientarsi nell'offerta di componenti per la domotica.

1.4.5 Altri standard

Atri standard molto diffusi sono:

- Cebus (Consumer Electronics Bus)
- LonWorks
- Batibus
- EIB (European Installation Bus)
- EHS (European Home System)

1.5 Caratteristiche di un sistema domotico

Le caratteristiche che un sistema domotico deve possedere sono:

- **Integrazione:**

  La ragione d'essere della domotica sta su un elemento fondamentale: combinare una pluralità di oggetti capaci di comunicare tra loro in un sistema di gestione globale. E' del resto qui che si trova la differenza fondamentale con un'installazione domestica "classica", perché questa si compone di un insieme di elementi, totalmente separati gli uni dagli altri e che non permettono, per questo fatto, che delle azioni "stand-alone".

  Un sistema domotico è quindi prima di tutto una rete, nozione gia ben conosciuta nel settore della telefonia e dell'informatica, che permette una comunicazione tra gli elementi partecipanti. Dal punto di vista tecnico, una rete di comunicazione è composta di una parte fisica, costituita dalla topologia e dal medium di rete e da una parte logica, comprendente il protocollo di rete e gli elementi materiali specifici necessari per applicare il protocollo di comunicazione, le interfacce di comunicazione e gli apparecchi di collegamento.

  Una buona capacità di interazione del sistema con diversi tipi di apparecchi consente, quindi, la gestione ad un livello comune e rende il più possibile omogenei tutti i componenti.

- **Apertura:**

  Il sistema deve essere in grado di accettare l'inserimento di nuove tecnologie quando queste sono rese disponibili, anche se non appartenenti al sistema stesso. Questo ci porta a richiedere uno standard a cui i produttori di apparecchi devono attenersi.
  Il sistema inoltre deve poter comunicare in vari modi con il mondo esterno: telefono, internet, SMS.

- **Flessibilità:**

  Significa che il sistema si deve adattare alle diverse esigenze degli utenti senza che sia necessario modificare l'hardware dell'impianto, ma ottenendo semplicemente via software il risultato desiderato.

- **Semplicità:**

  Uno degli scopi principali del sistema è di rendere semplice l'utilizzo di tecnologie non immediate per un utente medio. Data la complessità degli impianti gestiti è fondamentale la disponibilità di funzioni facilmente comprensibili da parte dell'utente. Inoltre il sistema dovrebbe avere diversi tipi di interfacce per adattarsi a persone anziane e disabili.

- **Affidabilità:**

  E' la capacità del sistema di conservare nel tempo le caratteristiche prestazionali dichiarate dal
  produttore, nelle condizioni di utilizzo stabilite.
  Sono da evitare fenomeni di sovratensione e di surriscaldamento.

- **Continuità di funzionamento:**

  Il sistema deve essere in grado di funzionare con continuità nelle più disparate condizioni di esercizio. In particolare il sistema deve garantire il funzionamento durante i black out, mediante batterie tampone. In luoghi isolati, in cui si verificano interruzioni di corrente frequenti e per lunghi periodi di tempo, si può prevedere anche un gruppo elettrogeno.
  In caso di guasto devono essere previsti, per alcune componenti, i comandi manuali (tapparelle, finestre, luci…).
  Infine deve essere prevista una funzione di diagnosi automatica in modo da segnalare immediatamente ai centri d'assistenza eventuali malfunzionamenti.

- **Espansibilità:**

  Significa poter aggiungere nuove componenti e funzionalità. E' un requisito importante perché consente all'impianto di adattarsi a diverse tipologie abitative e a diverse esigenze.

1.6 Sviluppi futuri della domotica

I moltissimi standard in circolazione per l'automazione di edifici, che si differenziano in base ai dispositivi utilizzati, alle dimensioni dell'area da controllare e dal tipo di servizi scelti, possono comportare un rallentamento della crescita del settore e un aumento dell'incertezza sia da parte del consumatore che delle stesse case produttrici. Questo proliferare di standard necessita, quindi, di una rapida regolamentazione.

Per quanto riguarda gli elettrodomestici e i dispositivi hardware in genere si nota che quelli che forniscono un interfaccia di comunicazione sono pochi e il loro costo è nettamente superiore ad un prodotto "a scatola chiusa" di pari livello. L'integrazione con ciò che è già disponibile è difficile e i prodotti *"automation ready"* stentano a diffondersi.

Una soluzione funzionale per avvicinare prodotti e sistemi differenti potrebbe ricercarsi nell'adozione di standard aperti e già affermati e in tecniche di progettazione delle applicazioni in maniera stratificata e intercambiabile, in modo da favorire l'integrazione di tecnologie diverse sia a livello di strutture hardware che software. E' coerente, quindi, la necessità di applicazioni estremamente flessibili in grado di adattarsi nel tempo ai nuovi sistemi ed allinearsi agli standard che si delineeranno man mano, in un settore in rapida espansione e mutamento.

1.7 Ruolo del telefono cellulare

All'interno del panorama domotico, un elemento, ormai diffusissimo, sta avendo un ruolo centrale e di primaria importanza per tutto lo sviluppo delle tecnologie di interfaccia. Sto parlando del telefono cellulare.

La sua grande diffusione e l'uso di tecnologie sempre più sofisticate ha imposto il telefono cellulare come nuovo trend del mercato[v]. Le sue ampie possibilità di connessione e la forte portabilità hanno fortemente contribuito a renderlo un elemento di primaria importanza all'interno di uno scenario domotico. Le applicazioni di un telefono cellulare all'interno di questo settore sono in continua crescita e rappresentano uno dei punti di maggiore interesse per le aziende produttrici.

Ad esempio è facile notare, guardandosi attorno, che l'esigenza di controllare da remoto un numero sempre più crescente di utenze di qualunque natura è quanto mai attuale: aumenta così in maniera

esponenziale il numero di telecomandi che è necessario avere. Un singolo telefono cellulare o un PDA può riunire in sé tutte le funzionalità dei telecomandi attuali, utilizzando standard già definiti, come l'infrarosso o il bluetooth.

Con il forte sviluppo delle reti 3G e quindi UMTS [11], si sono aperte possibilità di connessione che affiancano, alla classica trasmissione della voce, anche la videoconferenza e la trasmissione dati a pacchetto. Ciò ha reso possibile per la prima volta l'accesso, a costi contenuti, di dispositivi mobili al World Wide Web di Internet. Le attuali reti UMTS sono state potenziate mediante i protocolli High-Speed Downlink Packet Access (HSDPA), e High-Speed Uplink Packet Access (HSUPA) con una velocità massima teorica in download di 14,4 Mb/s e in upload di 7,2 Mb/s. Con il lancio di tariffe flat su tecnologia HSDPA, e con l'implementazione della tecnologia HSUPA (che migliora la velocità in upload), i servizi definiti "a banda larga mobile" possono essere considerati come alternativa alle connessioni ADSL fisse, e concorrenti delle future reti WiMAX.
Inoltre a queste si sono affiancate le connessioni wireless secondo lo standard 802.11x, ampliando ulteriormente le possibilità di connettività. I vantaggi sono indubbi. Innanzitutto una maggiore velocità e affidabilità rispetto all'UMTS, ma soprattutto una consistente diminuzione dei costi, considerando la forte diffusione di Access Point liberi e gratuiti.

I PDA ed i telefoni cellulari trascendono, quindi, il concetto stesso di telecomando, consentendo immaginariamente di portare con sé la propria abitazione, in qualsiasi luogo si desideri.

"So what used to fit in a building
now fits in your pocket,
what fits in your pocket now
will fit inside a blood cell in 25 years."

Raymond Kurzweil
(inventor and futurist)

Si può notare però che queste tecnologie sono piuttosto recenti, e quindi i modelli più vecchi ne sono parzialmente, o addirittura completamente sprovvisti. Nonostante tutto, il cellulare ha ancora un ruolo importante all'interno del meccanismo domotico, infatti esso può contare su un sistema di binding semplice ed efficace, ovvero il numero telefonico. L'invio di notifiche tramite SMS o messaggi telefonici preregistrati è quindi uno strumento potente e poco costoso, non richiede altri parametri che il numero di telefono ed ha una percentuale di successo vicina al 100%. L'unico difetto è rappresentato dalla possibile assenza di campo.

1.8 La piattaforma J2ME

La scelta del linguaggio da utilizzare in ambito mobile è ricaduta immediatamente sulla piattaforma J2ME [12]. In primis perché, essendo tutto il sistema sviluppato in Java, sarebbe stato molto più semplice integrare le varie parti. E' stata scelta, inoltre, per la sua flessibilità e la larga diffusione di dispositivi che supportano tale tecnologia La figura 1.2, infatti, mostra come nel 2005 erano ripartite le piattaforme per dispositivi mobili (Fonte: Evans Data Wireless Development Survey).

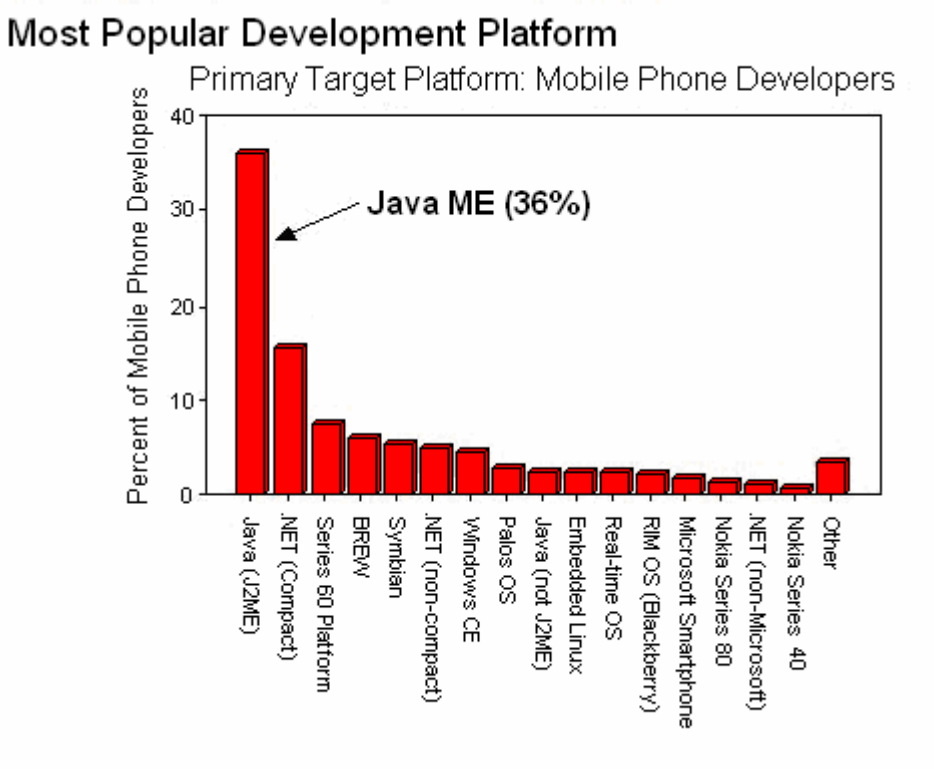

Figura 1.2: ripartizione delle piattaforme per dispositivi mobili

Java Micro Edition (noto anche come Java ME o J2ME), è un runtime e una collezione di API per lo sviluppo di software dedicato a dispositivi a risorse limitate come PDA, telefoni cellulari e simili. J2ME è la tecnologia più diffusa per lo sviluppo di giochi e utilities per i cellulari. Come le altre edizioni di Java, J2ME è una piattaforma portabile. Il suo funzionamento può essere emulato con un personal computer, cosa che semplifica l'attività di sviluppo e di collaudo.
Dal 22 dicembre 2006 il codice sorgente di J2ME è stato rilasciato sotto licenza GPL e quindi liberamente modificabile da chiunque.

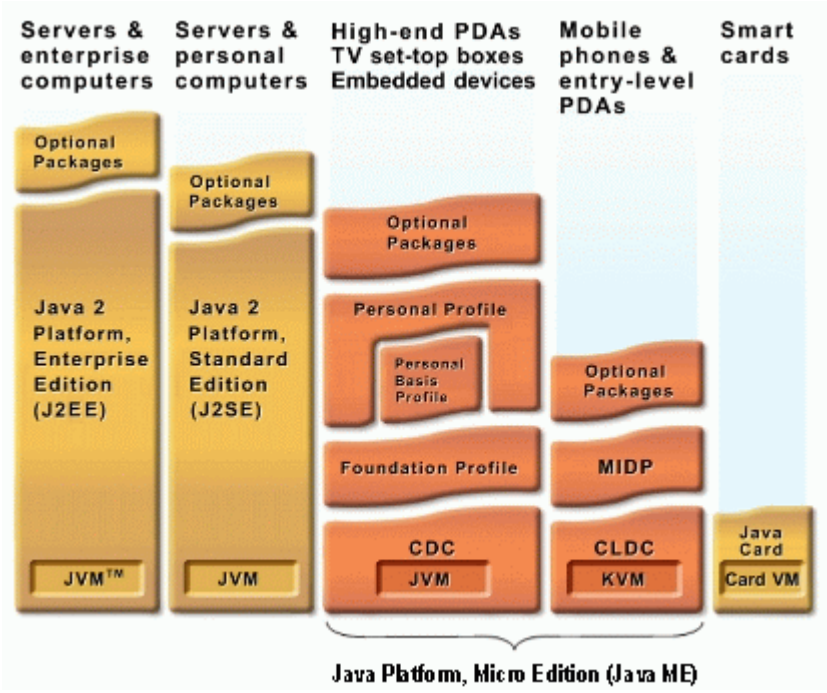

Figura 1.3: suddivisione del linguaggio Java per le varie categorie di dispositivi

L’architettura di J2ME, rappresentata nella figura 1.3, definisce le configurazioni, i profili ed i pacchetti opzionali la cui combinazione permette di ottimizzare le caratteristiche tipiche del dispositivo su cui si vuole operare. Le configurazioni definiscono il minimo set di librerie e le funzionalità della *virtual machine* supportata per un particolare insieme di dispositivi che condividono caratteristiche simili, come ad esempio la connettività alla rete e la memoria disponibile.

1.8.1 Configurazioni

Esistono due tipi di configurazioni: CDC (*Connected Device Configuration*) e CLDC (*Connected Limited Device Configuration*).

- **CDC**: per dispositivi con un minimo di 2MB di memoria disponibile e CPU a 32-bit. Questa configurazione include una *virtual machine* completa (JVM) e un set abbastanza ampio di API (derivante nella maggior parte da quelle della Java 2 Standard Edition).
- **CLDC**: è la configurazione più piccola, destinata a dispositivi con 128-512 KB di memoria, CPU a 16 o 32 bit e connessione *intermittente* alla rete. Include una *virtual machine* compatta e veloce (KVM), ed un ristretto ma funzionale set di API. Si adatta particolarmente bene al wireless, ma è anche utilizzata dalla maggior parte di dispositivi *embedded* a ridotte dimensioni.

Questa divisione sembra piuttosto chiara, ma le cose non sono così semplici, infatti dobbiamo ricordare che la tecnologia continua a progredire, è sufficiente provare a fare un confronto tra il primo cellulare posseduto e quello attuale per rendersi conto delle enormi differenze che intercorrono (figura 1.4).

Figura 1.4: due modelli di telefono cellulare Nokia appartenenenti a generazioni diverse

Con il passare del tempo si ha sempre più potenza di calcolo, più memoria, schermi più grandi e lo sconfinamento di queste due categorie diventa inevitabile. Questo ci porta all’introduzione dei profili.

1.8.2 Profili

I profili sono stati introdotti per rispondere alle differenze di funzionalità e avere maggiore flessibilità rispetto ai cambiamenti tecnologici. Possiamo vederli come una estensione della

configurazione, forniscono librerie per permettere i programmatori di scrivere applicazioni per un particolare tipo di dispositivi.
Un esempio di profilo è il MIDP – Mobile Information Device Profile, che descriveremo in seguito in quanto è stato usato per il nostro programma.

1.8.3 KVM – Kilobyte Virtual Machine

Come spiegato nell'introduzione, per eseguire un programma in Java abbiamo bisogno di una JVM – Java Virtual Machine, per il nostro target di dispositivi la JVM tradizionale non è utilizzabile, pertanto la Sun ha implementato una VM aderente alle specifiche CLDC: la KVM (notare che la KVM è l'implementazione della Sun, ma ciò non toglie che potrebbero esserci altre VM di altri produttori aderenti alle specifiche CLDC). La KVM è scritta in C per facilitarne la portabilità ed è concepita per essere piccola, efficace, e comunque rapida ad eseguire i comandi Java principali.
Sono richiesti 60k di memoria, 20-40k di memoria dinamica e può essere eseguita su processori a 16 bit con un clock di soli 25MHz, da qui il nome KVM, K sta per Kilobyte, ad indicare che può essere eseguita con poche decine di K di memoria

1.8.4 MIDP

Per fornire un ambiente di sviluppo completo e indirizzato ad una vasta categoria di dispositivi, le configurazioni devono essere combinate con i profili, ovvero con un set di API a più alto livello che definiscono ulteriori caratteristiche quali l'accesso a proprietà specifiche dei singoli dispositivi.
Il profilo di cui è stato dotato il CLDC è il MIDP (*Mobile Information Device Profile*), che offre funzionalità di base richieste dai dispositivi mobili. Presenta infatti API per gestire l'interfaccia con l'utente, la connettività alla rete, la memorizzazione permanente dei dati e la gestione dell'applicazione. Combinato con il CLDC, il MIDP fornisce un ambiente di sviluppo completo che beneficia delle capacità di dispositivi mobili, minimizzando il consumo di memoria e di energia. Questo ambiente fornisce infatti tre pacchetti java, ovvero `java.io, java.lang` e `java.util` che ereditano le funzionalità di J2SE adattate alla *Micro Edition*, e il pacchetto `javax` contenente le classi della `microedition`. In particolare, per la `microedition` sono presenti i seguenti quattro pacchetti:

- `javax.microedition.io`: per la gestione della connettività e le (minime) operazioni di I/O.
- `javax.microedition.lcdui`: per la creazione e la gestione dell'interfaccia con l'utente.
- `javax.microedition.midlet`: pacchetto base per la definizione di midlet e per le funzionalità ad esso assegnate.
- `javax.microedition.rms`: fornisce un semplice meccanismo di memorizzazione permanente dei dati.

Utilizzando il profilo MIDP è possibile sviluppare programmi chiamati *MIDlet*. La J2ME può essere ulteriormente estesa da un vasto numero di pacchetti opzionali, da combinare alle configurazioni utilizzate ed ai loro rispettivi profili. Creati per rispondere a specifiche richieste di mercato, questi pacchetti offrono API standard sia per tecnologie già esistenti, che per quelle emergenti. Tra questi pacchetti ci sono le *Wireless Messaging API* (WMA – JSR 120), definite dal Java Community Process (JCP). Esse forniscono delle API per inviare e ricevere messaggi binari e di testo – tipicamente del tipo store-and-forward, come i messaggi SMS (Short Messaging Service).

1.8.5 Le MIDlet

Parliamo ora delle MIDlet. E' il nome che si da alle applicazioni per cellulari programmate tramite CLDC e MIDP, il termine deriva dal profilo MID (*Mobile Information Device*), che identifica la

categoria di dispositivi su cui questi programmi devono essere eseguiti. Il suffisso *let* invece ricorda gli *applet*: semplici applicazioni Java con una struttura bene definita. Le MIDlet hanno effettivamente ereditato dagli *applet* molte delle loro caratteristiche: dalla struttura di base, alla semplicità implementativa.
L'applicazione deve estendere la classe `javax.microedition.midlet.MIDLET` ed implementare i suoi metodi astratti `startApp()`, `pauseApp()`, `destroyApp(boolean unconditional)`, che rappresentano lo stato in cui la MIDlet stessa può trovarsi (ed in effetti sono simili ai metodi *init*, *stop* e *destroy* delle Applet).
Dopo aver realizzato una MIDlet bisogna installarla ed eseguirla in un dispositivo (o in un suo emulatore).
E' bene sapere che nel dispositivo abbiamo un gestore di applicazioni (AMS – Application Manager Software) che è il software responsabile dell'installazione, dell'esecuzione e della rimozione delle MIDlet (i requisiti software richiesti dalla CLDC).
Quando viene eseguita una MIDlet l'AMS rende disponibili all'applicazione:

- l'accesso a CLDC e alla JVM,
- l'accesso alle classi MIDP,
- l'accesso ai file JAR e JAD

Tutti i file della MIDlet devono essere inseriti all'interno di un file JAR, che può contenere più MIDlet il cui insieme è detto MIDlet Suite. Il file JAR può essere caricato sul dispositivo in vari modi, a seconda del dispositivo e delle nostre preferenze (via porta infrarossi, cavo usb, bluetooth…). All'interno del file JAR abbiamo un file che ne descrive il contenuto e fornisce informazioni a proposito della MIDlet Suite, il file MANIFEST (manifest.mf).

Qui è riportato il file manifest della MIDlet sviluppata:

```
Manifest-Version: 1.0
MIDlet-Name: MobileDomusControl Midlet Suite
MIDlet-Vendor: Midlet Suite Vendor
MIDlet-Version: 1.0
MicroEdition-Profile: MIDP-2.0
```

Questi sono gli attributi obbligatori, poi ve ne sono altri opzionali, ad esempio:
MIDlet-Description
MIDlet-Icon

Oltre al manifesto in un JAR vi sono tutti i file class, tutti i file di risorsa (per esempio le immagini, il MIDP supporta il formato PNG – Portable Network Graphics) e le eventuali librerie di terzi.
Oltre al file JAR è possibile che sia disponibile un file JAD (Java Application Descriptor), che fornisce informazioni su quello che si troverà all'interno del file JAR.
Il file MANIFEST e il JAD sono molto simili, ma svolgono funzioni diverse. Il JAD serve in modo particolare quando si scaricano le applicazioni dalla rete, esso informa il gestore dell'applicazione di quali sono le caratteristiche richieste per il funzionamento del MIDlet prima ancora dello scaricamento del file JAR in modo che il gestore possa controllare se il dispositivo possiede le caratteristiche necessarie.

```
MIDlet-1: MoDoCo,/com/modoco/image/home.png,com.modoco.midlet.MobileDomusController
MIDlet-Jar-Size: 153033
MIDlet-Jar-URL: MobileDomusControl.jar
MIDlet-Name: MobileDomusControl MIDlet Suite Vendor
MIDlet-Push-1: sms://:5000,com.modoco.midlet.MobileDomusController,*
MIDlet-Vendor: Vendor
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

Anche qui ci sono degli attributi non indispensabili, per esempio:
MIDlet-Delete-Confirm
MIDlet-Description
Si possono anche inserire degli attributi definiti dall'utente. Gli attributi possono apparire in qualsiasi ordine, inoltre MIDlet-Name, MIDlet-Vendor, MIDlet-Version devono essere presenti in entrambi i file e devono corrispondere, mentre se gli altri attributi non dovessero corrispondere quelli del file JAD avrebbero la priorità.
Una volta installata la MIDlet apparirà nell'elenco delle applicazioni del cellulare e potrà essere lanciata ed eseguita.

1.9 Conclusioni di capitolo

In questo capitolo abbiamo posto le basi teoriche sulle quali verrà costruito il sistema. In particolare l'attenzione è stata focalizzata sullo stato attuale della domotica, identificando gli scenari applicativi e i benefici che questa scienza può portare. Inoltre, all'interno del capitolo, è stata fatta un'ampia analisi del linguaggio di programmazione utilizzato per sviluppare l'applicativo per dispositivi mobili, ovvero Java MicroEdition. Nei capitoli seguenti cercheremo di identificare un punto d'incontro tra domotica e mobile computing.

Capitolo 2

IL SISTEMA DOMUS

L'idea alla base del progetto è di analizzare una possibile integrazione tra domotica, networking e mobile computing. In particolare si vuole ricercare uno sviluppo innovativo dei dispositivi mobili in un contesto domotico, sfruttando tecnologie ad ampia diffusione.

L'analisi del sistema da progettare deve considerare l'ambito di applicazione e deve valutare con attenzione le tecnologie software più appropriate per il raggiungimento degli scopi che ci siamo prefissati.

2.1 Scenario generale

Lo scopo iniziale che ci siamo prefissati era di realizzare un sistema di video streaming per dispositivi mobili che consentisse la ricezione delle immagini catturate da una webcam. Il processo di sviluppo si sarebbe, quindi, dovuto concentrare sulla programmazione di due applicazioni distinte. La prima, orientata a piattaforme Desktop, in grado di interfacciarsi con le webcam e inviare in rete le immagini. La seconda, per dispositivi mobili, in grado di ricevere e visualizzare correttamente queste immagini. A partire da questo nucleo si sarebbero poi potute integrare funzionalità aggiuntive.

Il sistema strutturato in questo modo presentava un'evidente limitazione: rendeva accessibile le immagini della webcam solo su specifica richiesta dell'utente e non in automatico, a seguito magari di un particolare evento scatenante. Una parte del processo di sviluppo si è, quindi, concentrata sull'implementazione di un semplice motore di riconoscimento del movimento all'interno delle immagini. In questo modo il software sarebbe stato dotato di una parte intelligente, in grado di eseguire l'invio di una notifica all'utente in seguito alla rilevazione di movimento, ovvero l'evento scatenante. L'ambito di applicazione avrebbe così potuto estendersi dal monitoraggio remoto fino alla vera e propria video-sorveglianza in tempo reale. Parlando di telefoni cellulari il meccanismo di notifica più semplice ed efficace è sicuramente quello degli SMS. L'applicazione è stata quindi dotata di supporto per l'invio di SMS, in modo da legarsi strettamente al motore di rilevazione del movimento, in un meccanismo di causa-effetto.

Il sistema completo avrebbe dovuto, inoltre, funzionare il più possibile in modo autonomo, interrogando l'utente solamente per la configurazione strettamente necessaria al funzionamento. E' noto che nel campo del networking i parametri indispensabili alla creazione di una connessione sono tipicamente l'indirizzo IP e la porta della piattaforma di destinazione. Quindi per rimanere in linea con i principi di semplicità e trasparenza, si è reso necessario sviluppare un servizio che sollevasse l'utente finale, generalmente con scarse conoscenze informatiche, dalle problematiche correlate al sistema di indirizzamento. Infatti, nello scenario tipico di una connessione casalinga a internet via modem o ADSL, il provider assegna un indirizzo IP dinamico, cioè che cambia ad ogni collegamento. L'accessibilità di un servizio che "gira" sul proprio PC, come appunto il nostro applicativo da sviluppare, avrebbe dovuto essere nuovamente permessa ad ogni connessione, comunicando al mondo il nuovo indirizzo IP.

Era quindi necessario costruire un sistema d'astrazione dell'indirizzo IP che puntasse su informazioni più intuitive per L'utente medio. La ricerca in rete ha portato alla valutazione dei

servizi gratuiti per l'associazione dell'indirizzo IP ad un dominio di terzo livello. Il più famoso tra questi è certamente *DynDNS* [vi].

Da qui è nata l'idea di sviluppare un sistema di tracciabilità in proprio puntando, anziché su un dominio di terzo livello, su una semplice associazione tra una coppia username/password e l'indirizzo IP aggiornato. Un tale sistema avrebbe dovuto comprendere almeno 3 parti distinte: un client residente sul PC su cui è attivo il servizio e che deve comunicare al mondo il proprio indirizzo, un database sempre accessibile contenente l'associazione username/password – indirizzo ed infine un client per telefono cellulare in grado di interrogare il database per risolvere l'indirizzo IP aggiornato, semplicemente inserendo i parametri username/password. Particolare attenzione, inoltre, è stata dedicata alle politiche di inserimento e cancellazione di questi dati, in modo da non presentare alcun conflitto nel caso, ad esempio, di parametri identici.

Successivamente, con l'integrazione in ambito domestico, ci siamo imposti di portare il sistema ad un livello ulteriore, cercando di sfruttare pienamente tutte le caratteristiche che un ambiente domotico avrebbe potuto offrire. Di fondamentale importanza è stata, quindi, la valutazione delle tecnologie esistenti all'interno del laboratorio domotico e come queste avrebbero potuto essere utilizzate per i nostri scopi. In particolare abbiamo potuto usufruire di due telecamere IP e di un motore per l'esecuzione di eventi, chiamato EventEngine. La loro descrizione verrà approfondita in seguito.

2.2 Requisiti funzionali

I requisiti formali che l'applicazione finale dovrà possedere saranno:

Trasparenza

Il requisito di trasparenza è fondamentale nella progettazione di un'architettura domotica. Gli utilizzatori finali infatti non possono in alcun modo essere gravati del peso della tecnologia.
Per questo motivo è obbligatorio realizzare un sistema il meno intrusivo possibile, in modo da ridurre al minimo le interazioni con l'utilizzatore, una volta che il sistema sarà configurato appropriatamente.

Semplicità d'utilizzo

Requisito fondamentale per ogni applicazione orientata al vasto pubblico e all'utilizzo in un ambiente domestico. L'utilizzo deve essere intuitivo e il più possibile automatizzato, in modo da richiedere all'utente solamente i parametri strettamente necessari al suo funzionamento, esattamente come qualsiasi elettrodomestico. La maggior parte di questi requisiti si applicano alle interfacce grafiche che, quindi, devono presentare un livello molto elevato di intuizione. Fondamentali sono, ad esempio, le icone descrittive e le tooltip.

Immediatezza

Immediatezza significa raggiungere il proprio scopo in modo semplice e veloce. Pertanto le funzioni dell'applicazione devono eseguire i loro compiti nel minor tempo possibile e impiegando il minor numero di passaggi. E' importante ricordare che l'utilizzatore finale è, nella maggior parte dei casi, una persona comune senza alcuna conoscenza dell'informatica.

Indipendenza dalla piattaforma

Il sistema deve potersi adattare a diverse piattaforme hardware e software, in modo da garantire l'operabilità in più ambienti. Per questo motivo il software è stato interamente sviluppato in Java, linguaggio famoso per la sua portabilità. Analogamente il client per telefono cellulare è stato sviluppato con la piattaforma J2ME che garantisce il grado di diffusione e portabilità più alto in tutto l'ambiente del mobile computing.

Facile integrazione

Il sistema deve sapersi integrare efficacemente, utilizzando standard aperti, con tecnologie diverse in modo da favorire la stratificazione e una buona intercambiabilità con soluzioni diverse, sia a livello di strutture hardware che software. Il sistema deve, quindi, presentare un buon livello di configurazione per permettere l'adattamento a contesti differenti.
La particolare natura aggregata del sistema, formato da entità differenti che devono scambiarsi informazioni, implica l'adozione di un linguaggio comune, in grado di garantire semplicità ed efficacia. Questo protocollo deve, quindi, essere di facile implementazione per le future tecnologie che potrebbero venire inserite nel contesto, senza creare incompatibilità.

2.3 Prototipo iniziale

Il lavoro è quindi cominciato su un semplice prototipo client-server, sviluppato da Michele Mussi[vii], che permetteva la cattura del segnale della webcam e la successiva trasmissione direttamente ad un telefono cellulare compatibile con JavaME. Questo prototipo è stato utilizzato come nucleo principale attorno a cui sviluppare l'intera applicazione. In particolare è stato utile per comprendere il meccanismo di accesso, tramite Java, alla webcam e il successivo invio delle immagini verso un telefono cellulare. Quest'ultima parte è stata l'unica a rimanere immutata rispetto al prototipo, mentre il resto dell'applicazione è stato completamente riscritto.

2.4 Identificazione delle entità nel sistema

Viene qui operata una panoramica generale sulle entità coinvolte all'interno del sistema completo.

2.4.1 Sistema di video streaming - MotionDomus

Inizialmente l'obiettivo era creare un semplice sistema di video streaming che consentisse l'invio delle immagini di una webcam verso un telefono cellulare collegato.

Lo sviluppo, quindi, si è svolto in due direzioni diverse:

- La parte server, composta dal software Java in grado di interfacciarsi con la webcam e di creare una connessione per il telefono cellulare
- La parte client, formata dall'applicazione residente sul telefono cellulare, in grado di collegarsi con il server per ricevere, e successivamente visualizzare, le immagini.

Il nucleo lato server è stato ampliato includendo un'interfaccia grafica semplice e pratica ed implementando ulteriori funzionalità, come ad esempio la possibilità di catturare degli screenshot e salvarli in formato JPEG, di iniziare una sessione di streaming tramite il protocollo RTP e quindi di trasmettere in rete le immagini della webcam. Il software è stato dotato inoltre di un semplice sistema di Motion Detection, in grado di rilevare il movimento all'interno dei frames. E' importante

sottolineare che il sistema, inoltre, permette l'invio di messaggi di tipo SMS. Si tratta di uno strumento molto semplice, quanto potente. Infatti tramite uno speciale SMS è possibile attivare, tramite un meccanismo chiamato Push Registry, un applicazione remota residente sul telefono cellulare. Questa importante caratteristica permette al software di reagire immediatamente al verificarsi di un particolare evento, nel nostro caso la rilevazione del movimento, attivando tramite SMS l'applicazione client sul telefono cellulare in modo da inviare le immagini.
A supporto di questo progetto è stato sviluppato un sistema di tracciabilità dell'indirizzo IP. Infatti per funzionare correttamente il telefono cellulare deve disporre dell'indirizzo e della porta del pc su cui è presente la webcam e sui cui il programma server è in funzione. Notoriamente le connessioni Internet odierne in ambito domestico funzionano quasi esclusivamente con indirizzi dinamici, quindi si è reso necessario sviluppare un sistema che mantenesse sempre accessibile ed aggiornato l'indirizzo. Questo sistema si compone di quattro parti:

- Un software lato server che espone dei metodi pubblici per l'inserimento, la manipolazione e l'eliminazione di tuple all'interno di un database MySQL.
- Un client che si occupa di ricavare l'indirizzo pubblico (e quello privato) tramite un servizio web e, successivamente, salvarlo all'interno del database assieme ad una coppia username/password scelta all'avvio del programma stesso.
- Una semplice servlet che, definiti i parametri username e password, restituisce l'indirizzo memorizzato nel database.
- Infine il client su telefono cellulare, all'atto di apertura, richiede i due parametri necessari a richiamare la servlet tramite una sessione http, in modo da risalire all'indirizzo.

Terminato lo sviluppo di questa prima parte stand-alone si è pensato a come potesse integrarsi ed espandersi ulteriormente all'interno di un ambiente domotico.
L'idea è stata quindi quella di adattarlo alle tecnologie e agli standard del laboratorio domotico in via alla Cascata a Povo, in provincia di Trento. Ci siamo avvalsi, quindi, delle telecamere IP presenti al suo interno e di un software sviluppato da Enrico Nicoletti, chiamato EventEngine.

2.4.2 EventEngine

L'EventEngine [13] è un motore per l'esecuzione di eventi altamente riconfigurabile e flessibile, slegato dallo strato di "sensing" e di "intelligence" e quindi adatto a diversi utilizzi. Il suo scopo è quello di coordinare un'insieme di strutture hardware e software rendendo l'ambiente sensibile ad eventi catturati mediante sistemi eterogenei di sensori. Il programma può interagire con l'ambiente tramite l'uso di sistemi hardware/software in grado di eseguire compiti specifici (accendere luci, attivare telecamere, aprire porte, ecc...). Le funzionalità sono descritte dal diagramma UML in figura 2.1.

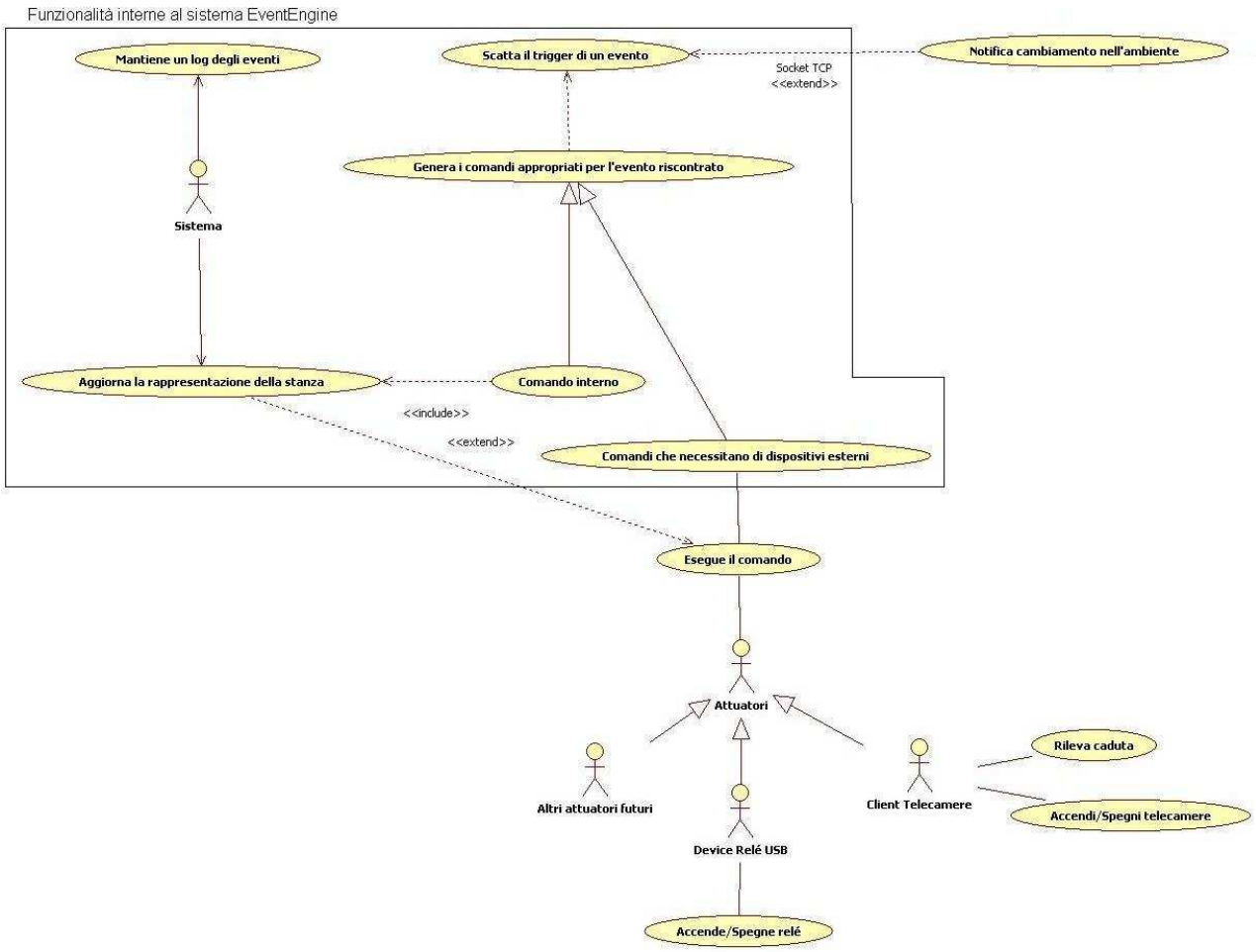

Figura 2.1: diagramma UML delle funzionalità dell'EventEngine

I principali utilizzi dell'EventEngine sono:

Utilizzo come ambiente di sviluppo per applicazioni domotiche
Può esser usato come ambiente di sviluppo e testing per applicazioni domotiche. Simula un ambiente virtuale con movimento casuale di persone al suo interno. Il software esegue comandi e aggiorna l'ambiente grafico monitorando la corretta esecuzione degli stessi.

Utilizzo come tool di reporting degli eventi generati in un,ambiente
Il software prevede un utility di log degli eventi generati, accompagnati da data ed ora. Vengono annotati tutti i cambiamenti in una struttura dati che può essere memorizzata su file per consultazioni future.

Utilizzo come programma di domotica
Accoppiato alla scheda di Relè USB può essere usato come programma di domotica per controllare l'accensione/spegnimento di apparecchiature elettroniche, in base ad eventi triggerati da sistemi di sensori connessi al programma tramite socket.

L'interfaccia dell'EventEngine (figura 2.2) è molto semplice ed intuitiva. Nella parte alta della finestra si trova la MapArea, in cui viene rappresentato l'ambiente. Per la simulazione viene usata una riproduzione dell'Application Lab, sito in via alla Cascata a Povo, provincia di Trento.
Nella parte inferiore sinistra si ha accesso alle informazioni sullo stato della connessione con i sistemi esterni di sensori ed attuatori. In particolare le tre icone rappresentano nell'ordine:

- il *client posizione*, che è un sistema di sensori di posizione (nel testo chiamato *client posizione*) in grado di inviare al sistema le coordinate in cui si trovano le persone che indossano i Mote. Questo client invia anche una notifica quando un Mote viene autenticato per l'accesso alla stanza.
- il *client telecamere*, che esegue i comandi di accensione/spegnimento delle telecamere Ethernet ed avvia lo streaming delle immagini riprese. Le telecamere in questa mappa sono rappresentate con dei triangoli in grigio quando sono spente e in giallo quando sono attive.
- Il *device relè USB*, in grado di comandare l'accensione e lo spegnimento di 4 relè, a cui sono collegati altrettanti apparati elettronici.

In basso al centro si trova un area testuale di reporting delle operazioni eseguite dal sistema e informazioni di utilità per l'utente.
In fondo alla finestra, verso destra si trovano tre pulsanti. Il primo denominato "*Sim*" avvia la simulazione di un ambiente con uno o più occupanti che si muovono al suo interno.
Il secondo pulsante, denominato "*Report*" salva su file il log degli eventi accaduti fino a quel momento e ne manda a video un estratto.
L'operazione di reporting non interferisce con il funzionamento del programma che, nel frattempo, continua ad accettare dati dai sensori.
L'ultimo pulsante disconnette eventuali client connessi al sistema, salva il log su file e chiude il programma.

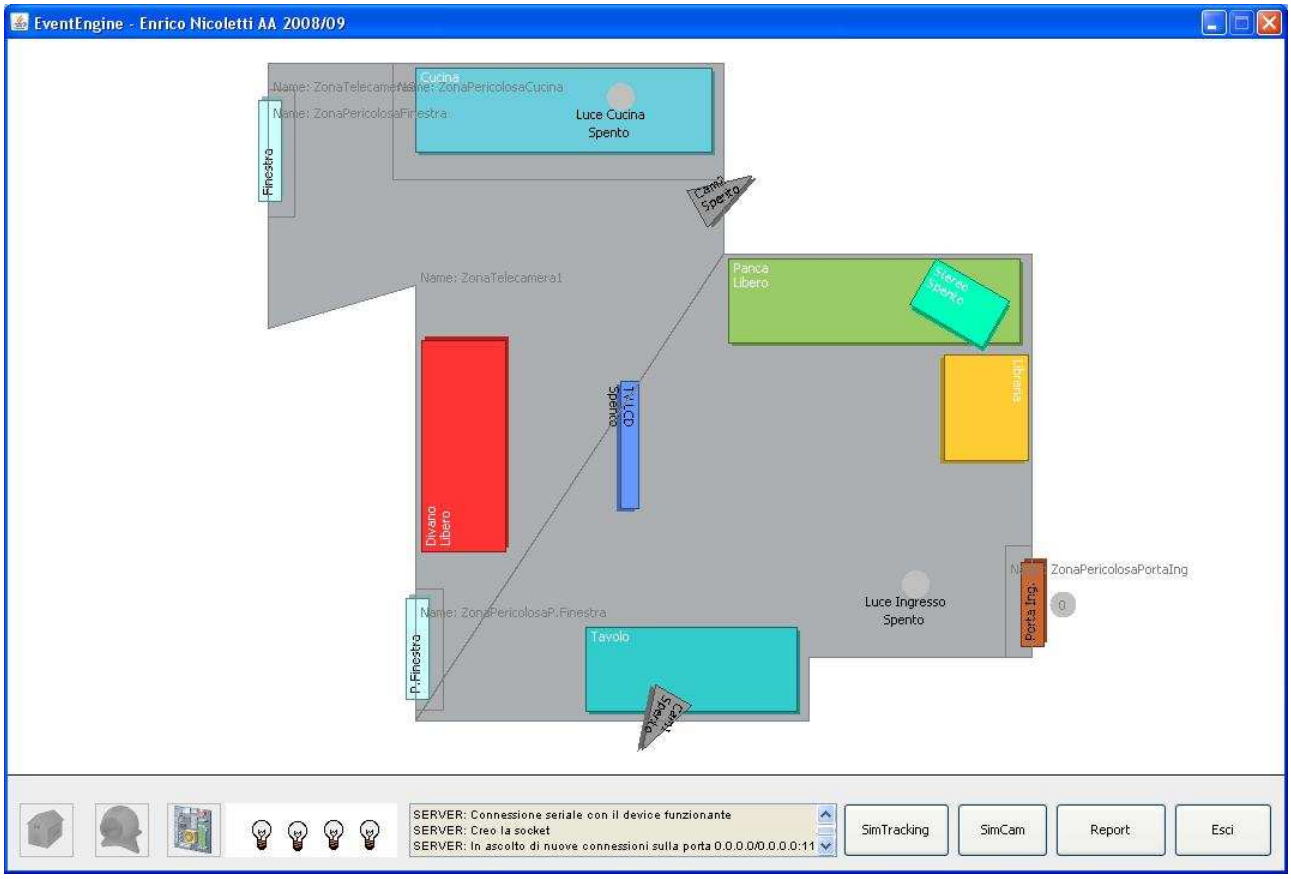

Figura 2.2: l'interfaccia dell'EventEngine di Enrico Nicoletti

2.4.3 Integrazione in ambito domotico - ControlDomus

Terminato lo sviluppo della prima parte stand-alone, gli sforzi si sono concentrati sulla possibile integrazione del software con le tecnologie del laboratorio domotico.

Analogamente al primo progetto, il lavoro ha riguardato sia l'ambito Desktop, con lo sviluppo di un applicazione lato server, sia l'ambito mobile, con l'estensione dell'applicazione per telefono cellulare. L'intero sistema ha preso il nome *ControlDomus*.

L'applicazione lato server è stata completamente riscritta da zero, cercando però di riutilizzare alcune tecnologie già impiegate per la prima parte del lavoro. Il nome identificativo è stato scelto nell'acronimo *DoCo*, ovvero *Domus Control*.

Per la parte di interfaccia con le telecamere è stato costruito un semplice decodificatore di flusso MJPEG. Invece per la parte di controllo ci siamo avvalsi, come già detto, della collaborazione di Enrico Nicoletti e del suo EventEngine.

In particolare, in questa direzione, il lavoro è stato fatto concentrandosi su due scopi:

- La possibilità di inviare comandi all'EventEngine in modo da regolare l'accensione/spegnimento di apparecchiature tramite telefono cellulare.
- La possibilità di ricevere dallo stesso EventEngine informazioni importanti, ad esempio la posizione di un entità in una zona potenzialmente pericolosa, come può esserlo la cucina, in modo da valutare se a queste informazioni devono essere corrisposte controazioni.

Inoltre studiando le possibilità di integrazione si è pensato di sfruttare le enormi potenzialità delle telecamere (figura 2.3) presenti all'interno dell'Application Lab. Queste infatti dispongono di un sistema di Motion Detection e di rilevazione audio integrato molto più sofisticato rispetto a quello definito nel sistema. Le telecamere inoltre sono pienamente configurabili in modo da reagire nel caso vengano superati i livelli di soglia. Le reazioni comprendono il caricamento di file tramite FTP, HTTP ed e-mail oppure l'invio di notifiche tramite e-mail, HTTP e TCP. Si è scelto di utilizzare quest'ultimo sistema, disponendo una socket in ascolto sulla porta 1987 in modo da catturare le eventuali notifiche delle telecamere, formattate tramite la convenzione *event-name*|*event-description*. Tali notifiche vengono quindi confrontate con quelle presenti nel file XML di configurazione, in modo da prendere i provvedimenti necessari.

Figura 2.3: un modello di telecamera Axis 212 PTZ

L’applicazione lato client, invece, è stata adattata a quella creata per il MotionDomus. Il nucleo è rimasto sostanzialmente lo stesso, ma a questo si sono aggiunte le funzioni per il controllo dei dispositivi. Il nome che si è scelto per identificare questa applicazione è *MoDoCo*, acronimo di *Mobile Domus Control*.

Le funzionalità del sistema ControlDomus sono riassunte nella figura 2.4 e verranno descritte con attenzione all’interno del capitolo 5.

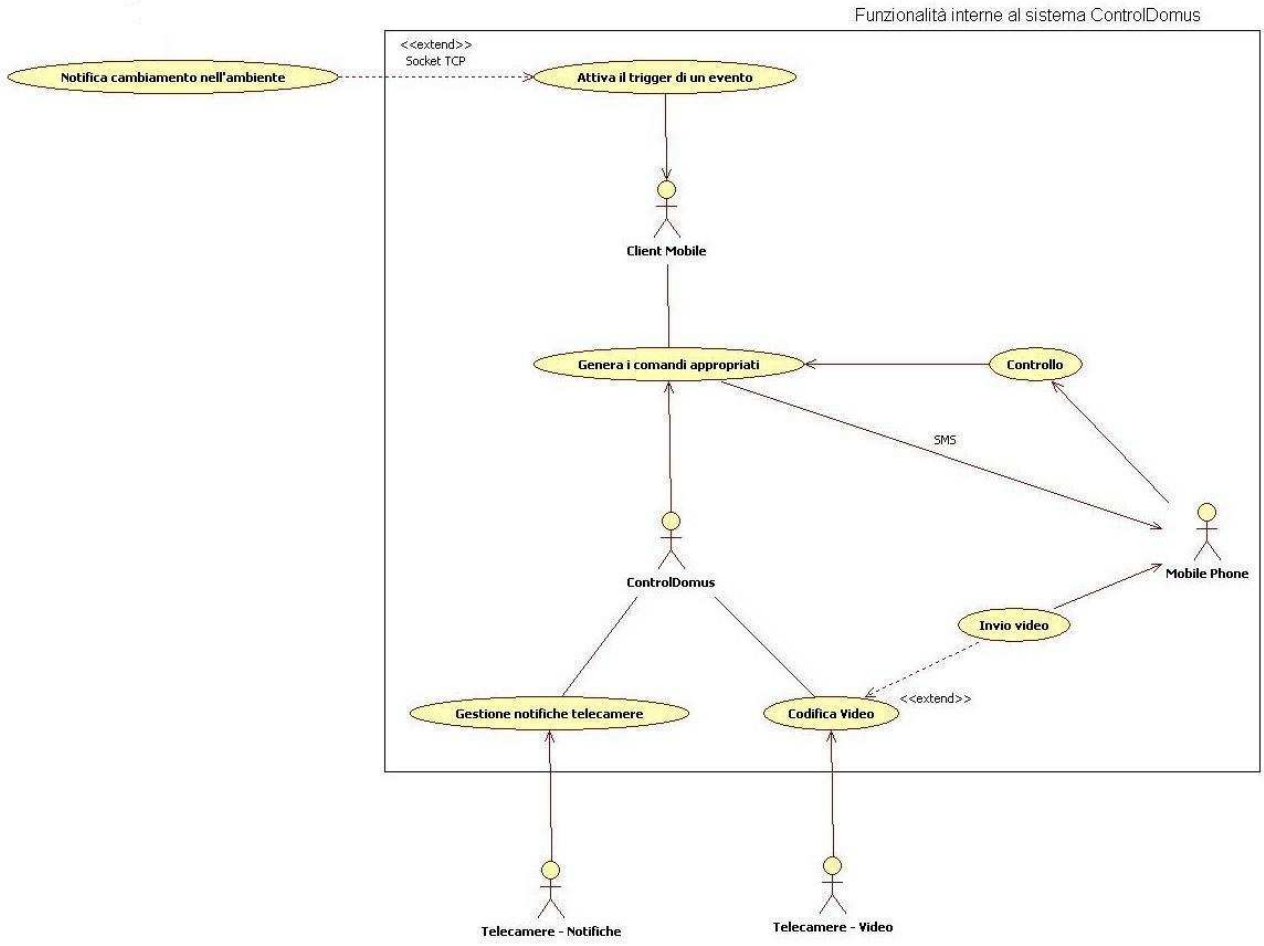

Figura 2.4: diagramma UML delle funzionalità del sistema ControlDomus

2.5 Problematiche incontrate e relative soluzioni

2.5.1 Configurazione dei software

Per la configurazione dei paramentri dei software MotionDomus e Domus Control era necessario scegliere un approccio che fosse in linea con i principi di portabilità ed espansibilità e che fosse, inoltre, relativamente semplice da gestire ed utilizzare. Per questo scopo è stato scelto lo standard XML. E’ un sistema ormai fortemente consolidato, libero da vincoli di piattaforme software; è gratuito; ben supportato; facile da generare e leggere, non ambiguo ed estensibile. E’ stato scelto, inoltre, per mantenere una corrispondenza con l’EventEngine di Enrico Nicoletti, considerando che i trigger definiti al suo interno devono essere uguali a quelli definiti nel Domus Control.
L’accesso diretto ai file XML non semplifica la procedura di configurazione per un utente medio, ma nel prossimo futuro la creazione di interfacce a maschera d’inserimento porteranno sicuramente benefici anche su questo fronte.

2.5.1.1 Il parsing dei file XML

Al centro dell'architettura di un sistema che utilizza documenti XML vi è sempre un parser XML. Il parser XML ha il compito di caricare il documento XML e di renderlo disponibile all'applicazione. Vi sono due categorie di parser, validanti e non validanti. I parser validanti permettono di verificare il contenuto del documento attraverso un file esterno che contiene la descrizione della struttura. La struttura può essere descritta attraverso DTD oppure attraverso XML Schema.
I DTD sono parte dello standard XML 1.0 e permettono di descriverne la struttura, mentre gli XML Schema sono un nuovo standard, in fase di approvazione, che permette di validare anche i tipi di dati utilizzati nel documento.
I parser non validanti non permettono di controllare la struttura del documento, delegando questo compito all'applicazione. Un parser XML può esporre i propri servizi attraverso due tipi di interfacce:

- L'interfaccia basata sullo standard DOM permette di vedere il documento XML come se fosse un albero e permette di navigare i vari nodi.
- L'interfaccia basata sullo standard SAX non mantiene il documento in memoria, ma mentre lo carica invoca i metodi dell'applicazione che si devono occupare di trattare gli elementi che gli sono stati passati.

Per fare il parsing dei file XML si è utilizzato un parser DOM (Document Object Model).
L'interfaccia DOM è stata scelta per le seguenti caratteristiche:

- Assicura il rispetto della grammatica del linguaggio
- Facilita la costruzione di documenti ben formati e validi
- Fornisce una rappresentazione dei dati indipendente dalla grammatica
- Facilita la manipolazione, anche complessa, dei documenti all'interno di applicazioni
- Non è necessario manipolare espressamente stringhe del linguaggio XML
- Può essere impiegato sia in fase di analisi che di costruzione dei documenti

Il problema della costruzione di oggetti tramite parsing di stringhe testuali è uno dei classici problemi della programmazione ed è stato risolto con l'utilizzo del design pattern di costruzione denominato "builder". Il pattern builder separa la costruzione di un oggetto complesso dalla sua rappresentazione. Ciò ha l'effetto immediato di rendere più semplice la classe, permettendo a una classe builder separata di focalizzarsi sulla corretta costruzione di un'istanza e lasciando che la classe originale si concentri sul funzionamento degli oggetti. Questo è particolarmente utile quando ci si deve assicurare che un oggetto sia valido prima di istanziarlo, e non si vuole che la logica di controllo appaia nei costruttori della classe. Un builder permette anche di costruire un oggetto "passo-passo", cosa che si può verificare quando si fa il parsing di un testo o si ottengono i parametri da un'interfaccia interattiva.

2.5.2 Integrazione con le webcam

Di primaria importanza per tutto il sistema era l'accesso semplice e rapido, da parte dell'ambiente Java, alla webcam installata sul PC. Si rendeva così necessario ricercare un meccanismo di astrazione che fornisse delle primitive per garantire l'accesso a tale dispositivo multimediale, indipendentemente dal produttore. La scelta è ricaduta sulle *JMF* (Java Media Framework) [14], in quanto ufficiali di Sun Microsystems e quindi maggiormente affidabili e supportate[viii].

2.5.2.1 JMF (Java Media Framework)

Il *Java Media Framework* è un insieme di API create per permettere di incorporare tipi di dati multimediali in applicazioni o applet Java; sono state sviluppate per supportare i più comuni standard, quali MPEG-1, MPEG-2, Quick Time, AVI, WAV, AU e MIDI.
Usando le *JMF* si ha la possibilità di riprodurre gli standard sopraccitati (ma non solo) e di gestirli su base temporale.

Com'è ben noto, la peculiarità di Java è l'utilizzo di una *Java Virtual Machine* (JVM) che, tramite emulazione software (utilizzando esclusivamente la CPU), interpreta il byte-code generato dal compilatore Java.

Questo meccanismo permette la portabilità del codice Java su più piattaforme ma, nel caso in cui sia richiesta una elevata velocità di esecuzione, impone dei seri vincoli di prestazioni.
Il trattamento di dati multimediali è uno dei casi in cui è richiesta un'elevata velocità computazionale (decompressione delle immagini, rendering…), dove, quindi, non è sufficiente la sola emulazione della CPU per ottenere delle prestazioni ottimali.
Ogni sviluppatore che desideri implementare un lettore multimediale in Java, e voglia ottenere delle prestazioni eccellenti, deve, necessariamente, ricorrere a codice nativo della piattaforma alla quale è interessato.
Questo procedimento comporta due problemi:

- Specifica conoscenza delle funzioni native da parte del programmatore.
- Un programma Java che utilizza codice nativo non è più trasportabile su piattaforme diverse da quella originaria.

Le API del *JMF* tentano di risolvere questi problemi, mettendo a disposizione del programmatore una serie di chiamate ad "alto livello" per la gestione del codice nativo.
Usando il *JMF*, l'applicazione o applet non ha necessità di conoscere quando e se, deve sfruttare particolari metodi nativi per svolgere una determinata azione.

Il Player

Il Player è la struttura che le API del *JMF*, mettono a disposizione del programmatore per la riproduzione di dati multimediali.

Come già accennato, le funzioni messe a disposizione dal Player consentono agli sviluppatori di software di non interessarsi direttamente delle chiamate al codice nativo, di impegnare le risorse necessarie per la riproduzione e di effettuarne un eventuale rilascio, quando queste non siano più necessarie.
Le chiamate a metodi e classi di "alto livello" , rendono quindi trasparente al programmatore la connessione che si stabilisce tra la *Java Virtual Machine* e le routines specifiche di sistema (proprio secondo la filosofia generale di Java).

Stati del Player

Per la riproduzione audio e video sono necessarie delle risorse che impegnano il sistema in modo non indifferente.

Le *JMF* hanno, allora, diviso lo stato di un Player in 6 fasi, visibili in figura 2.5, ciascuna

caratterizzata dalle operazioni che è possibile effettuare. Questi stati percorrono la "vita" di un Player durante la riproduzione di un media.

Essi sono i seguenti:

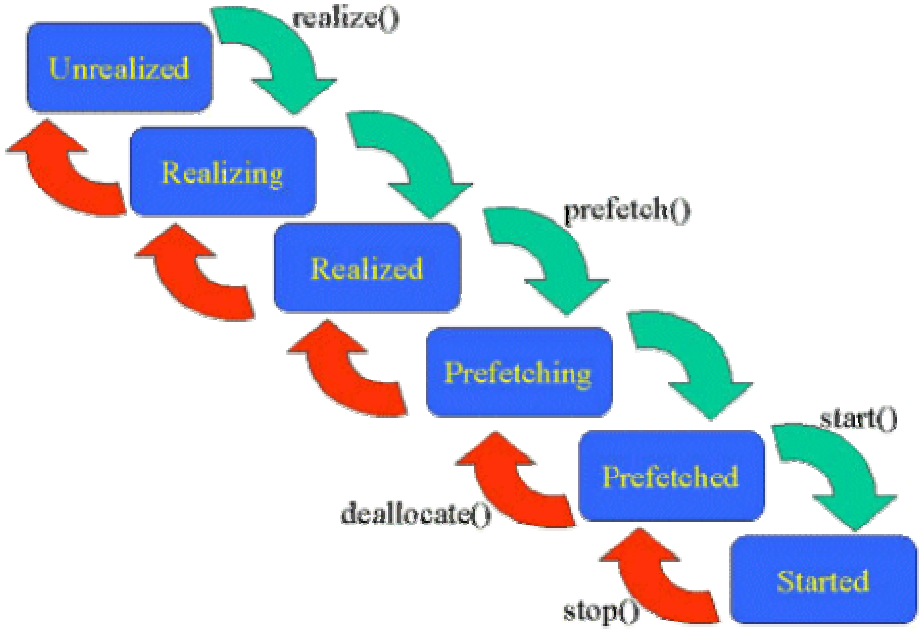

Figura 2.5: sequenza degli stati di un player e dei metodi associati

- **Unrealized**: il player si trova in questo stato all'atto della sua creazione, cioè quando è solamente a conoscenza dell'URL della risorsa da riprodurre. Un opportuno *Manager* è responsabile della creazione del player attraverso il metodo `createPlayer()` il cui unico parametro è proprio l'URL associato.
- **Realizing**: una volta che il player è stato creato, attraverso il metodo `realize()` della interfaccia *Controller*, è possibile portarlo in questo stato. Qui il player acquisisce le risorse di sistema che utilizzerà durante la riproduzione ma che non sono utili ad altri eventuali riproduttori. Si dice che il player si impossessa delle risorse non esclusive.
- **Realized**: una volta che il player ha acquisito le risorse relative alla fase precedente, si porta automaticamente nello stato *Realized*. Qui il player è a conoscenza del suo compito per cui è qui che si rendono disponibili i Control-Panel ed il Visual-Panel. Un player nello stato *Realized* non blocca nessun altro player candidato alla riproduzione del media, per cui possono esistere più player in questo stato.
- **Prefetching**: quando il player è nello stato *Realized*, è possibile utilizzare il metodo `prefetch()` per portarsi nello stato di *Prefetching*. Questo stato è caratterizzato dal fatto che il player inizia ad acquisire le risorse per la riproduzione del media, comprese le risorse esclusive se accessibili. Esso si riserva anche un certo buffer in cui farà il preloading dei primi dati da riprodurre. Il contenuto di questo buffer potrà eventualmente essere visualizzato da un Control-Panel di tipo *CachingControl*.
- **Prefetched**: terminata la precedente fase, si ha il passaggio automatico allo stato di *prefetched*. Qui il player è pronto a partire. Tutte le risorse necessarie sono state acquisite ed eventuali altri player che utilizzassero le stesse risorse sono impossibilitati all'esecuzione fino a che le risorse stesse non saranno rilasciate.
- **Started**: finalmente attraverso il metodo `start()` è possibile iniziare la riproduzione della risorsa attraverso il player.

Per il passaggio da uno stato ad un altro esistono cinque metodi principali che possono essere applicati solamente quando il player è in un particolare stato, pena la generazione di una eccezione anch'essa esplicativa del problema. Esistono i seguenti cinque metodi:

- **`start()`**: appartiene all'interfaccia *Clock* e può essere eseguito solamente quando il player è nello stato *Prefetched*. Lo stato seguente è lo stato *Started*.
- **`stop()`**: appartiene all'interfaccia *Clock* e può essere eseguito solamente quando il player è nello stato *Started*. Lo stato seguente è lo stato *Prefetched*.
- **`realize()`**: serve per portare il player dallo stato di *Unrealized* allo stato di *Realizing*.
- **`prefetch()`**: serve per iniziare la fase di prefetching
- **`deallocate()`**: serve per liberare le risorse esclusive acquisite dal player. In questo modo un eventuale altro player nello stato di *Realized*, può eseguire la `prefetch()`.

Il passaggio tra gli stati è notificato attraverso l'emissione di un evento la cui gestione è alla base della programmazione con il *JMF*.

Sorgente dei dati multimediali

Un Player creato con il *JMF* permette di trattare molteplici formati multimediali sia che provengano da files locali, da files remoti o tramite broadcast interattivi.
Il Player è in grado di supportare due meccanismi di consegna:

- **Pull Data Source**: il programma client inizia il trasferimento dei dati e ne controlla il flusso. I protocolli prestabiliti per questo tipo di consegna sono HTTP e FILE
- **Push Data Source**: il processo server inizia il trasferimento dei dati e ne controlla il flusso. Per dati broadcast, è prestabilito il trasferimento tramite le specifiche del protocollo RTP.

Le operazioni di controllo (avanzamento, riposizionamenti su base temporale, ritorno a capo, ecc.) che una applicazione permette di effettuare su uno stream di dati multimediali, dipendono dal meccanismo di consegna utilizzato.
Un filmato MPEG, ad esempio, può subire avanzamenti, riposizionamenti, ecc., se viene usato il meccanismo di *Pull Data Source*; cioè, se è l'applicazione a gestire il flusso di dati, può decidere a piacimento come manipolare lo stream.
Al contrario, se l'applicazione client riceve un broadcast da un server, questa si deve limitare a riprodurre le informazioni ricevute, dato che, non ha la possibilità di modificare l'afflusso di dati a piacimento.

Creare ed utilizzare un Player

Attraverso un "Media Manager" messo a disposizione dal *JMF* è possibile creare un *Player*.
Il *Player* è costituito da due elementi distinti ma intimamente correlati: *Visual Component* e *Control Component*.
Il primo serve per restituire una rappresentazione visiva della componente video del media (se ne esiste una); il secondo mette a disposizione delle funzioni per regolare l'andamento della riproduzione (per ogni standard multimediale supportato il *JMF* mettono a disposizione un *Control Conponent* di default).

Per la creazione del *Player* si deve chiamare il metodo `createPlayer()` del *Manager*: questo utilizza un URL per definire la fonte dei dati.

Una volta che il *Player* è stato creato e che si è impadronito delle risorse necessarie è possibile accedere al *Visual Component* e al *Control Component*.

Visual Component

Il *Visual Component* è quello in cui il *Player* mostra all'utente i dati processati. Se il media non ha una componente video da riprodurre il *Visual Component* sarà restituito pari a null; questo può essere il caso di uno stream audio (generalmente non si può affermare che tutti gli stream audio non abbiano un *Visual Component* poiché, ad esmpio, questa struttura può servire per visualizzare la forma d'onda del segnale).

Control Component

Il *Control Component* funziona da interfaccia tra le richieste di chi visiona il filmato (play, stop, ecc.), e il *Player*. Il risultato di tale mediazione è visto dall'utente attraverso il *Visual Component* (avanzamento in play, stop immagine, ecc.).

Questo componente può essere ottenuto tramite la chiamata `getControlPanelComponent()` di *Player*. Il metodo restituisce il pannello di controllo standard per il media che il *Player* sta processando.
Il *JMF* concede la possibilità di personalizzare il pannello di default per ogni media supportato; è quindi permesso aggiungere funzionalità personalizzate al *Control Component*, sviluppando le relative routines di gestione.

2.5.3 Causa - Motion Detection

La caratteristica chiave del sistema è la capacità di rilevare movimento all'interno dello stream video catturato dalla webcam. Questa caratteristica, sottolineata anche dal nome dell'applicazione (Motion Domus – da Motion Detection), rappresenta la parte intelligente del sistema. Infatti alla rilevazione del movimento viene corrisposto l'invio di un SMS di attivazione ad un numero di telefono preconfigurato, secondo una politica di causa-effetto.
Successivamente, con l'integrazione in ambiente domotico, sono stati aggiunti nuovi micro-sistemi esterni in grado di estendere l'intelligenza globale del sistema garantendo maggior precisione e completezza. In congiunzione a questi anche il meccanismo d'invio degli SMS è stato ulteriormente evoluto.

Gli approcci alla tecnica di Motion Detection sono molteplici, ma generalmente tutti quanti si basano sullo stesso concetto di comparazione del frame video corrente con uno precedente o, al più, con qualcosa che chiameremo background. Scopo della tesi non è l'analisi di questi approcci, verrà quindi descritto, nella sezione Implementazione, solamente il metodo utilizzato all'interno delle applicazioni.

2.5.4 Effetto – Invio e ricezione SMS

Un sistema intelligente deve corrispondere al verificarsi di un evento chiave, chiamato "causa", un "effetto" appropriato. Per rimanere nel mondo dei terminali mobili si è scelto di utilizzare gli SMS (Short Message Service) come meccanismo di notifica o di attivazione remota della MIDlet. Le tecnologie analizzate di seguito, ovvero le *WMA* [15] e il *Push Registry* [16], consentono infatti di attivare automaticamente una MIDlet all'arrivo di un messaggio. Questa funzionalità è utile per gestire la comunicazione tra l'applicazione Desktop e la MIDlet, evitando che quest'ultima debba essere sempre in esecuzione. Inoltre, è possibile utilizzare direttamente il numero di telefono per

indirizzare il messaggio, senza che sia necessario ricorrere a complessi sistemi di registrazione. Lo scenario ideale prevede, infatti, l'invio di un SMS da server al terminale mobile, l'attivazione automatica della MIDlet e la conseguente attivazione della connessione su rete IP da parte di quest'ultima. In tal modo il terminale si connette solo quando necessario, con evidente risparmio di costi e consumi.

In seguito, con l'integrazione in ambito domotico, si è voluto affiancare a questo meccanismo di attivazione anche quello più comune di SMS testuale, personalizzando accuratamente il contenuto in base all'evento scatenante.

Ora ci proponiamo di descrivere le tecnologie utilizzate per la ricezione in ambiente mobile.

2.5.4.1 WMA (Wireless Messaging API)

Le *WMA* sono un package opzionale (figura 2.6) basato sul *Generic Connection Framework* (GCF) destinati ad entrambi i profili CLDC e CDC e quindi per piattaforme J2ME. Si può però inoltre notare come anche la piattaforma Java 2 Standard Edition (J2SE) può utilizzare le *WMA*, in quanto la JSR 197 (Generic Connection Framework Optional Package for J2SE) è completa.

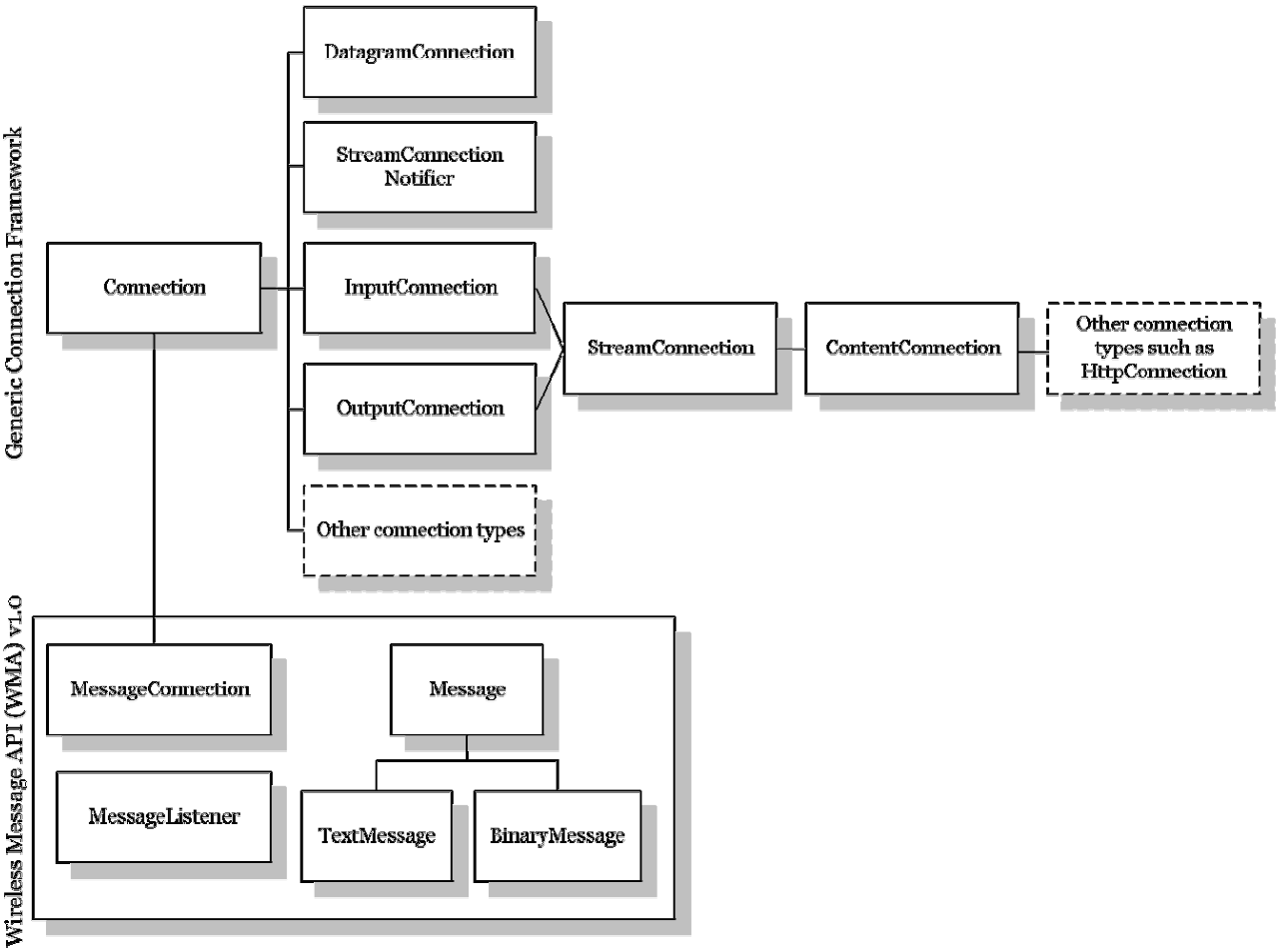

Figura 2.6: componenti del GCF e del WMA

Tutti i componenti della *WMA* sono contenuti in un singolo package, il `javax.wireless.messaging`, che definisce tutte le interfacce utilizzate per inviare e ricevere messaggi, sia binari sia di testo.

L’interfaccia Message

L’interfaccia `javax.wireless.messaging.Message` è la base per tutti i tipi di messaggio trasmessi utilizzando le *WMA*. Un messaggio è ciò che è mandato e ricevuto, prodotto e consumato. Le *WMA* offrono il supporto per messaggi binari e di testo. Definiscono inoltre un’interfaccia listener per ricevere messaggi in modo asincrono.

Sotto alcuni aspetti, un messaggio è simile ad un Datagramma: ha un indirizzo mittente e uno di destinazione ed un payload.

Le *WMA* definiscono due sottointerfaccie, `BinaryMessage` e `TextMessage`, la specifica inoltre è estensibile permettendo il supporto di tipi di messaggio addizionali.

L’interfaccia `BinaryMessage`

La sottointerfaccia `BinaryMessage` rappresenta un messaggio con un payload binario.

L’interfaccia `TextMessage`

La sottointerfaccia `TextMessage` rappresenta un messaggio con un payload di tipo testo, gli SMS ad esempio rientrano in questa categoria. Prima della trasmissione o della ricezione del messaggio, l’implementazione del dispositivo è responsabile della codifica o della decodifica della stringa da e verso il formato appropriato, per esempio GSM 7-bit o UCS-2.

L’interfaccia MessageConnection

L’interfaccia `MessageConnection` è una sottointerfaccia del package `javax.microedition.io.Connection` appartenente al *Generic Connection Framework.*

`MessageConnection` fornisce i metodi `newMessage()`, `send()` e `receive()` per, rispettivamente, creare, mandare e ricevere oggetti di tipo `Message`. Quest’interfaccia definisce inoltre due stringhe costanti, `BINARY_MESSAGE` e `TEXT_MESSAGE`, una delle quali è passata come parametro al costruttore `newMessage()` per creare l’appropriato oggetto `Message`.

Per creare una `Connection` (in questo caso una `MessageConnection`) è sufficiente chiamare il metodo `open()`, mentre per chiuderla si utilizza il metodo `close()`. E’ possibile avere più `MessageConnections` aperte simultaneamente.

Una `MessageConnection` può essere creata in due modi distinti: come connessione client oppure come connessione server. Nel primo caso l’unica azione possibile è l’invio dei messaggi, mentre nel secondo caso le azioni consentite sono sia l’invio, sia la ricezione dei messaggi. Come qualsiasi tipologia di connessione inclusa nel *GCF* anche una `MessageConnection` è creata a partire da un URL. Nel caso di una connessione client, l’URL include un indirizzo di destinazione:

```
(MessageConnection)Connector.open("sms://+5121234567:5000");
```

Nel caso di una connessione server, invece, viene specificato un indirizzo locale (non un host, ma un indirizzo locale specifico per il protocollo, tipicamente una porta). E’ questo il caso della nostra applicazione:

```
(MessageConnection)Connector.open("sms://:5000");
```

Se si prova ad aprire la connessione di un indirizzo locale già in uso viene lanciata una `IOException`.

Il meccanismo per creare una `MessageConnection` non è specifico per un singolo protocollo, ma è inteso per supportare più protocolli di tipo wireless. L'implementazione definisce quali adattatori di protocollo sono effettivamente da intendersi "wireless messaging protocols". La specifica della *WMA* definisce i seguenti adattatori:

- `"sms"` per il famoso Short Messaging System. La comunicazione SMS è bidirezionale: si possono creare, mandare e ricevere messaggi (il supporto è sia per connessioni client che server).
- `"cbs"` acronimo di Cell Broadcast Short Message. I messaggi CBS sono inviati in broadcast da una stazione base, tipicamente una cella, e possono solamente essere ricevuti. L'invio di un messaggio su una connessione di tipo `cbs` risulterebbe in una `IOException`.

2.5.4.2 Push Registry

In stretta relazione con gli SMS si è potuto utilizzare nella MIDlet un meccanismo noto come *Push Registry*.

Il concetto di "push" si riferisce tipicamente al meccanismo o all'abilità di ricevere e processare informazioni in modo asincrono, non appena l'informazione stessa risulta disponibile, anzichè forzare l'applicazione ad usare tecniche sincrone di polling che possono utilizzare molte risorse e incrementare la latenza.

Il *push registry* (figura 2.7), introdotto nel profilo MIDP 2.0 (JSR 118) permette alle MIDlet di essere avviate automaticamente, senza l'inizializzazione da parte dell'utente. Il *push registry* gestisce l'attivazione di MIDlet sia tramite rete, sia tramite un meccanismo di timer.

Il *push registry* fa parte dell'*Application Management System* (AMS), il software nel dispositivo che è responsabile del life-cycle di ogni applicazione (installazione, attivazione, esecuzione, e rimozione). Il *push registry* è il componente dell'AMS che espone le push API e tiene traccia delle applicazioni registrate.

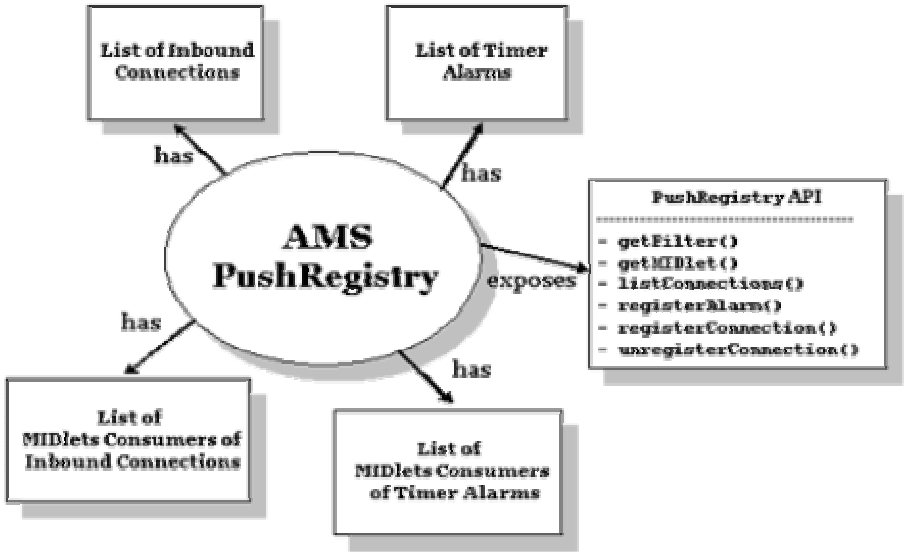

Figura 2.7: elementi tipici del Push Registry

Un tipico *push registry* mantiene le liste di connessioni e di allarmi in memoria e nel RMS, ovvero lo spazio di memorizzazione persistente.
Il *push registry* fa parte del *Generic Connection Framework* (GCF) ed è incapsulato all'interno di una singola classe, `javax.microedition.io.PushRegistry`, che espone tutti i metodi.

Attivazione MIDlet e Life-Cycle

L'avvento del *push registry* non cambia l'effettivo life-cycle di una MIDlet, ma introduce due nuovi modi attraverso cui la MIDlet può essere attivata (figura 2.8):

- Tramite connessioni di rete
- Tramite allarmi basati su timer

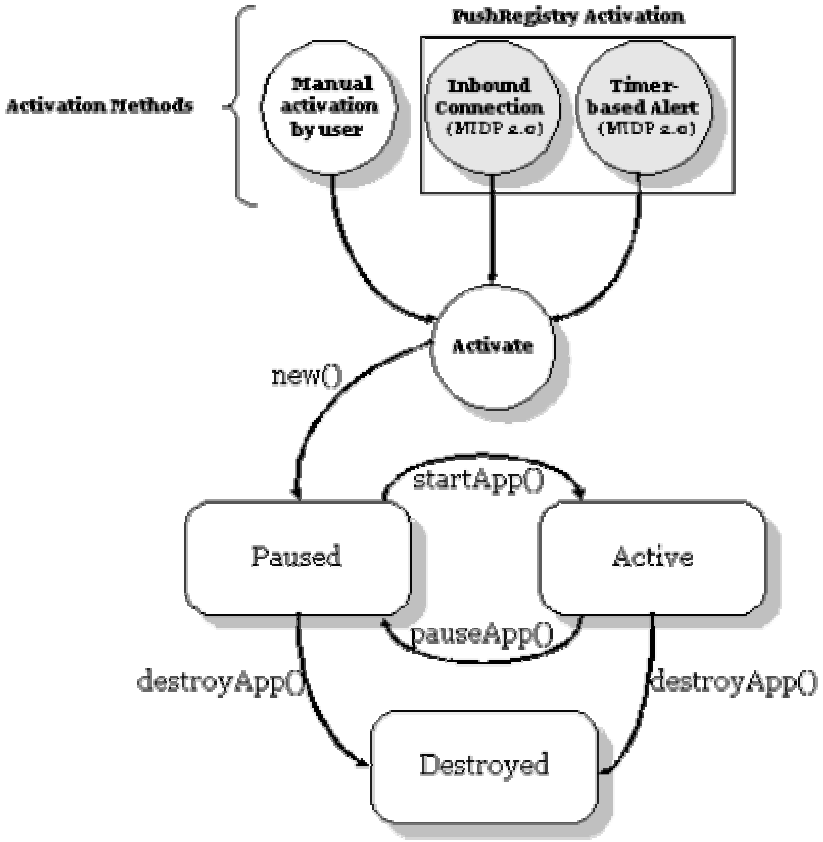

Figura 2.8: Life-Cycle e attivazione di una MIDlet standard MIDP 2.0

Condivisione della responsabilità per le azioni di Push

Una volta che la MIDlet si è registrata nel *push registry*, la responsabilità di processare le informazioni di push viene suddivisa in questo modo:

1. Quanto la MIDlet non è attiva, l'AMS monitora i push events registrati a favore della MIDlet. Quando si verifica uno di questi eventi, l'AMS attiva la MIDlet appropriata per gestirlo.

2. Se la MIDlet è attiva (running), la stessa MIDlet è responsabile per i push events.

Questa separazione di responsabilità non solo semplifica l'implementazione dell'AMS, ma inoltre evita l'imposizione di formati o metodi per gestire le informazioni di push. Queste decisioni sono lasciate al programmatore.

In figura 2.9 è rappresentato con un Sequence Diagram il ruolo di un agente esterno all'interno del processo di attivazione di una MIDlet.

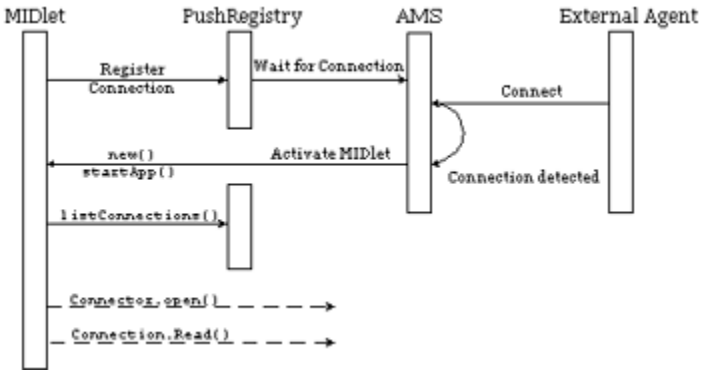

Figura 2.9: Sequence Diagram dell'attivazione tramite rete

Connessioni in ingresso

Per supportare l'attivazione basata sulla rete, la piattaforma deve supportare le connessioni in ingresso. Nel profilo MIDP 1.0 il solo tipo di connessione ufficialmente definito era l'`HttpConnection`, ma essendo una connessione solamente di tipo client non era utilizzabile attraverso un meccanismo di push. Successivamente con il profilo MIDP 2.0 sono stati definite ulteriori connessioni, oltre all'HTTP, come le socket TCP e i datagrammi UDP, perfettamente utilizzabili per il push. In aggiunta, le *Wireless Messaging API* (WMA, JSR 120) rendono possibile l'attivazione tramite push basato su SMS. Il profilo MIDP 2.0 non specifica quale tipo di connessione deve essere disponibile per il meccanismo di push; questa decisione è lasciata agli sviluppatori delle piattaforme.

Push Registration

Per usufruire del sistema di push, le MIDlet devono prima registrarsi al *push registry*, usando uno dei due tipi di registrazione:

- **Registrazione statica** – la registrazione di tutte le connessioni statiche avviene durante il processo di installazione della MIDlet. Vengono specificate elencando gli attributi MIDlet-Push nel file JAD o nel JAR manifest. L'installazione fallisce se si prova a registrare un indirizzo che è già stato utilizzato. Nel processo di disinstallazione di una MIDlet automaticamente le connessioni vengono liberate in modo da essere riutilizzate.
- **Registrazione dinamica** – le connessioni dinamiche e gli allarmi basati su timer vengono registrati a runtime, utilizzando le PushRegistry API.

Registrazione statica – Attributo `MIDlet-Push`

Le registrazioni statiche sono definite elencando uno o più attributi `MIDlet-Push` nel file JAD oppure nel file manifest. L'AMS esegue una registrazione statica quando la MIDlet suite è installata. In modo analogo, quando la MIDlet suite viene disinstallata, l'AMS automaticamente cancella le registrazioni associate all'applicazioni nel push registry.

Il formato per l'attributo `MIDlet-Push` è:

```
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>
```

dove:

- **`MIDlet-Push-<n>`** è il nome della proprietà che identifica la registrazione, mentre <*n*> è un numero che comincia da 1. Nella nostra MIDlet è `MIDlet-Push-1`. E' importante sottolineare che sono permesse più entries di questa tipologia.
- **`<ConnectionURL>`** è l'URL della connessione che identifica l'endpoint in ingresso da registrare. Il formato utilizzato è lo stesso usato nell'invocazione di `Connector.open()`. Nel nostro caso viene usata la stringa `sms://:5000` per riservare una connessione server in ingresso sulla porta 5000.
- **`<MIDletClassName>`** è il nome completo della MIDlet che deve essere attivata quando viene rilevata attività sulla rete in `<ConnectionURL>`.
- **`<Allowed-Sender>`** è un filtro utilizzato per restringere i server che possono attivare `<MIDletClassName>`. E' possibile l'utilizzo di caratteri wildcard. Una **`*`** indica uno o più caratteri, mentre **`?`** indica un carattere. Nella MIDlet sviluppata non vengono imposte restrizioni di questo genere.

Se l'indirizzo locale richiesto `<ConnectionURL>` è già in uso, l'installazione è destinata a fallire. Il modo tramite cui viene comunicato il fallimento all'utente dipende dal vendor e dall'implementazione.

L'attributo che identifica la nostra applicazione è il seguente:

```
MIDlet-Push-1: sms://:5000, com.modoco.midlet.MobileDomusController,*
```

Registrazione dinamica

La registrazione dinamica viene effettuata utilizzando le API del *push registry.* La registrazione avviene a runtime e può interessare sia le connessioni di rete, sia gli allarmi basati su timer. Ci limiteremo a descrivere brevemente la prima tipologia.

Registrare una connessione in ingresso

Le *Wireless Messaging API* (JSR 120) permette l'utilizzo di SMS per la comunicazione tra MIDlet. Uno dei vantaggi maggiori di questa API è la possibilità di specificare una porta per l'inoltro dei messaggi. In questo modo, con una procedura che ricorda le porte TCP/UDP, è possibile indirizzare i messaggi ad una specifica applicazione e, attraverso il p*ush registry* fornito da MIDP 2.0, attivarla se non già attiva.

È importante notare che i messaggi sono inviati in formato 8 bit (per semplificare il codice dell'encoder), per cui il messaggio ricevuto attraverso le *WMA* è una istanza di `BinaryMessage` anzichè un `TextMessage`.

2.5.5 Memorizzazione persistente

Il fatto di utilizzare una piattaforma di sviluppo orientata a dispositivi mobili, quindi ognuno con caratteristiche diverse, comportava la scelta di un sistema standard di memorizzazione persistente dei dati che possedesse caratteristiche di semplicità ed affidabilità. In campo Desktop si utilizza

generalmente la memorizzazione su file. In campo mobile invece, a parte alcune rarissime eccezioni, i dispositivi per i quali MIDP è stato inizialmente progettato erano privi di filesystem. Inoltre, le modeste risorse a disposizione del runtime Java ME non avrebbero consentito di implementare una libreria completa per la gestione dei file. Per questo motivo, MIDP è stato dotato di un sistema di persistenza semplificato, basato sulla lettura e scrittura di vettori di byte in un'area riservata della memoria flash.

2.5.5.1 RMS - Record Management System

Il *Record Management System* (RMS) [17], implementato dalle classi del package `javax.microedidtion.rms`, permette la creazione di archivi (RecordStore) a cui è assegnato un nome univoco. La specifica JTWI (JSR 185) [18] richiede che il runtime sia in grado di gestire almento cinque *RecordStore* indipendenti. La quantità di memoria dedicata all'RMS dipende dal dispositivo in uso: quelli di fascia alta generalmente non pongono alcun limite, se non quello fisico di memoria effettivamente disponibile nel terminale; i modelli entry-level forniscono generalmente alcuni MB di spazio. I dati memorizzati nel RMS sono accessibili a tutte le MIDlet appartenenti alla stessa MIDlet Suite e non è richiesta alcuna Permission per leggere e scrivere dati.

RecordStore

Un *RecordStore* è una collezione di record, univocamente identificati da un valore intero chiamato record ID. Questo rappresenta la chiave primaria per i record. Il primo record ha un ID pari a 1, mentre ad ogni record aggiunto verrà assegnato l'ID precedente incrementato di 1.

I *RecordStores* (file binari) sono dipendenti dalla piattaforma perchè vengono creati in aree di memoria dipendenti dal dispositivo. Le API del *RMS* forniscono le seguenti funzionalità:

- Permettono alle MIDlet di manipolare (aggiungere e rimuovere) dei record all'interno di un RecordStore
- Permettono alle MIDlet della stessa suite di condividere i record
- Non forniscono però un meccanismo per condividere i record tra MIDlet Suites differenti

Nomi dei Record Store

I nomi dei *RecordStore* sono case sensitive, inoltre non possono essere più lunghi di 32 caratteri. In aggiunta una MIDlet non può creare due *RecordStore* con lo stesso nome nella stessa applicazione, questo è possibile solo in applicazioni differenti.

Utilizzo dei Threads

L'implementazione dell'RMS nel profilo MIDP assicura che tutte le operazioni eseguite su di un *RecordStore* sono atomiche, sincrone e serializzate, in modo da permettere accessi multipli senza correre il rischio di corrompere i dati. Nonostante ciò, se la propria MIDlet utilizza thread multipli, per accedere ad un *RecordStore*, è responsabilità del programmatore sincronizzare l'accesso, altrimenti potrebbe verificarsi la sovrascrittura di record.

Utilizzare un RecordStore

Per aprire un *RecordStore* si utilizza il metodo statico `openRecordStore()`

```
RecordStore db = RecordStore.openRecordStore("myDBfile", true);
```

Il codice sopra crea un nuovo database file chiamato myDBfile. Il secondo parametro, impostato a true, indica di creare il *RecordStore* nel caso non sia già presente.

Un record è un array di byte. E' possibile utilizzare le classi `DataInputStream`, `DataOutputStream`, `ByteArrayInputStream`, e `ByteArrayOutputStream` per convertire varie tipologie di dati in array di byte.
L'inserimento di un record all'interno del *RecordStore* avviene attraverso il metodo `addRecord()`, che riceve come parametri il byte array contenente i dati e la coppia di interi che individuano rispettivamente l'offset all'interno dell'array e il numero di byte da leggere. Il metodo restituisce un intero, che rappresenta l'identificativo associato al record appena inserito. Tale ID è il riferimento che dovrà essere utilizzato per successive operazioni di lettura, modifica o cancellazione del record. Il solo ID però non è sufficiente a fornire un meccanismo di indirizzamento efficace nella gestione dei record, è necessario predisporre opportune strutture di indice o procedure di ricerca per poter individuare i dati sul *RecordStore*.

La scelta di utilizzare il *RMS* per memorizzare le informazioni persistenti è stata dettata dalla necessità di mantenere una compatibilità con i modelli più datati di telefono cellulare. Nell'immediato futuro un ulteriore sviluppo della MIDlet potrebbe essere l'implementazione di meccanismi di salvataggio su filesystem, certamente più flessibili e funzionali.

2.5.6 Invio immagini in rete

Dopo essere riusciti a collegare con successo il dispositivo mobile con l'applicazione si è pensato a come si potesse trasmettere in rete le immagini della webcam, in modo da renderle accessibili a più PC. Era quindi necessario valutare con attenzione i protocolli del livello applicativo, scegliendo quello più adatto alle proprie esigenze. Si è ricercato, anche in questo caso, di privilegiare la semplicità di implementazione in ambiente Java. La scelta è, quindi, ricaduta sul protocollo RTP.

2.5.6.1 RTP

L'*RTP* o *Real-time Transport Protocol* [18] è un protocollo del livello applicazioni (l'ultimo livello del modello OSI) per servizi in real time.

Questo protocollo permette distribuzione di servizi che necessitano di trasferimento in tempo reale, come l'interattività audio e video. Fra questi servizi si trovano anche:

- l'identificazione del payload type
- il sequence numbering
- il timestamping

È stato sviluppato da un gruppo di ricerca noto come Audio-Video Transport Working Group, facente capo alla IETF (Internet Engineering Task Force). Il corrispondente RFC è stato pubblicato nel 1996.

L'*RTP* doveva inizialmente essere un protocollo multicast, ma viene più spesso impiegato in applicazioni unicast. È basato sul protocollo UDP e viene usato in congiunzione con RTCP (RTP Control Protocol) [19] che monitora la qualità del servizio e trasporta le informazioni riguardo ai partecipanti ad una sessione. RTCP è sufficiente per sessioni "loosely controlled", in cui cioè non c'è un reale controllo dei partecipanti e set-up della sessione, e non è necessario che tutti i requisiti di controllo siano soddisfatti. Per questo RTP può essere coadiuvato da un protocollo apposito per

la gestione delle sessioni (come SIP o H.323). Solitamente le applicazioni pongono l'*RTP* sopra l'UDP per le operazioni di multiplexing e checksum, anche se può essere usato con altri protocolli di rete e trasporto sottostanti.

I numeri di sequenza (sequence numbers) che troviamo nel protocollo RTP permettono all'utente che riceve i dati di ricostruire la sequenza dei pacchetti del mittente.

2.5.7 Tracciabilità IP

Nel valutare la costruzione del sistema di tracciabilità dell'indirizzo IP sono state identificate, come già accennato, almeno 3 componenti necessarie: un applicativo su PC che comunica l'indirizzo IP, un database i cui viene salvato in relazione ad una coppia username/password ed infine il client su dispositivo mobile che legge il database per recuperare l'indirizzo. Una prima scelta per adempiere a queste necessità consisteva nell'istanziare socket di comunicazione tra le entità coinvolte nel processo. La poca flessibilità di questo approccio e l'assenza del meccanismo di socket per dispositivi mobili con profili MIDP inferiori alla versione 2.0 hanno portato alla ricerca di soluzioni alternative più adatte allo scopo. La scelta è ricaduta sulle *RMI*.

2.5.7.1 RMI – Remote Method Invocation

Remote Method Invocation (invocazione remota di metodi) o *RMI* [20] è una tecnologia che consente a processi Java distribuiti di comunicare attraverso una rete. Questa tecnologia include una API il cui scopo esplicito è quello di rendere trasparenti al programmatore quasi tutti i dettagli della comunicazione su rete. Essa consente infatti di invocare un metodo di un oggetto remoto (cioè appartenente a un diverso processo, potenzialmente su una diversa macchina) come se tale oggetto fosse "locale" (ovvero appartenente allo stesso processo in cui viene eseguita l'invocazione). In questo senso, la tecnologia *RMI* può essere ricondotta, da un punto di vista concettuale, all'idea di chiamata di procedura remota riformulata per il paradigma *object-oriented* (in cui, appunto, le *procedure* sono sostituite da *metodi*).

L'utilizzo di un meccanismo di invocazione remota di metodi in un sistema *object-oriented* comporta notevoli vantaggi di omogeneità e simmetria nel progetto, poiché consente di modellare le interazioni fra processi distribuiti usando lo stesso strumento concettuale che si utilizza per rappresentare le interazioni fra i diversi oggetti di una applicazione, ovvero la chiamata di metodo. Per contro, meccanismi di comunicazione fra processi remoti più primitivi (come lo scambio messaggi) rappresentano una forma di interazione "estranea" al paradigma *object-oriented*.

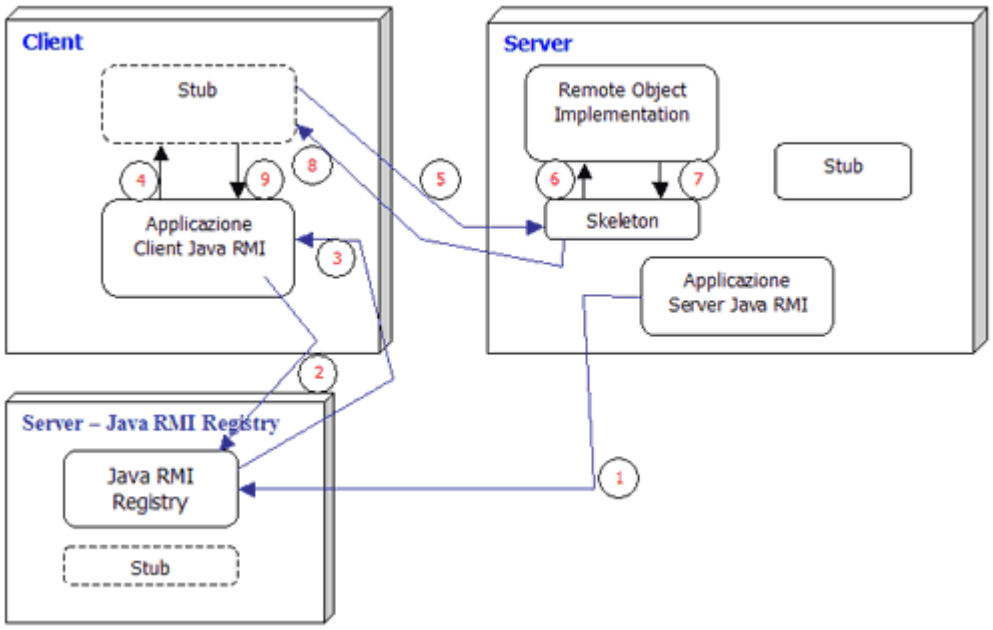

Figura 2.10: interazione tra i vari componenti in gioco nelle RMI di Java

Come si vede dal diagramma in figura 2.10, *RMI* si basa sull'interazione tra tre entità distinte:

- Uno o più Server RMI (per semplicità ne considereremo uno)
- Il Java RMI Registry (localizzato sul server)
- Uno o più Client RMI (per semplicità ne considereremo uno)

Il *Server RMI* implementa un'interfaccia relativa ad un particolare oggetto *RMI* e registra tale oggetto nel Java *RMI Registry*. Il Java *RMI Registry* è, semplicemente, un processo di tipo daemon (integrato con il JDK) che tiene traccia di tutti gli oggetti remoti disponibili su un dato server. Il *Client RMI* effettua una serie di chiamate al *registry RMI* per ricercare gli oggetti remoti con cui interagire.

A questo punto ci si potrebbe porre un paio di domande:

- Come fa un oggetto che si trova su una macchina client ad invocare dei metodi che sono definiti su un'altra macchina (server) ?
- Chi si occupa di gestire le problematiche legate alla comunicazione di rete?

Il trucco è tutto da ricercare nei due componenti che nel diagramma precedente sono denominati *Skeleton* e *Stub*[ix].

Lo *Skeleton*, sul server, si occupa di interagire direttamente con l'oggetto *RMI* che espone i metodi "remotizzati", inviando a quest'ultimo tutte le richieste provenienti dallo *Stub*.

Lo *Stub*, invece, rappresenta una sorta di classe "clone" che ripropone e mette a disposizione del client tutti i metodi che sul server sono stati definiti e implementati come remoti. In altre parole, lo *Stub* fa le veci di una classe spesso denominata "proxy class".

Sia lo *Skeleton* che lo *Stub* si occupano, infine, in modo trasparente all'utente (e al programmatore) della gestione della comunicazione tra il client ed il server.

Vediamo ora di dettagliare, passo per passo, il meccanismo descritto nel diagramma iniziale, utilizzando la medesima numerazione riportata in figura.

1. Viene creata sul server una istanza dell'oggetto remoto e passata in forma di stub al Java *RMI registry*. Tale stub viene, quindi registrato all'interno del registry stesso.
2. L'applicazione client richiede al *registry RMI* una copia dell'oggetto remoto da utilizzare.
3. Il Java *RMI registry* restituisce una copia serializzata dello *stub* al client
4. L'applicazione client invoca uno dei metodi dell'oggetto remoto utilizzando la classe "clone" fornita dallo *stub*
5. Lo *stub* richiama lo *skeleton* che si trova sul server chiedendogli di invocare sull'oggetto remoto lo stesso metodo che il client ha invocato sullo *stub*
6. Lo *skeleton* invoca il metodo richiesto sull'oggetto remoto
7. L'invocazione del metodo sull'oggetto remoto restituisce il risultato allo *skeleton*
8. Lo *skeleton* comunica il risultato allo *stub* sul client
9. Lo *stub* fornisce il risultato all'applicazione client iniziale

2.5.7.2 Middleman Architecture

Come già detto, un telefono cellulare equipaggiato con il profilo MIDP 1.0 non supporta connessioni tramite socket o datagramma (e quindi un utilizzo diretto delle RMI), a meno di utilizzare package opzionali che, comunque, sarebbero andati contro il principio di portabilità. L'unica possibilità di connessione possibile era quindi l'HTTP connection, basata sul protocollo request-response. La necessità di utilizzare questa tipologia di connessione ha condotto all'utilizzo di una particolare tipologia di architettura, chiamata *Middleman* [21], che comportasse un meccanismo di astrazione tra la connessione HTTP del dispositivo mobile e i metodi remoti. Questo genere di architettura implica la presenza di una servlet nel mezzo che accetta una richiesta per aprire una socket o invocare un metodo remoto. La servlet esegue (se necessario) il parsing della richiesta del client, la processa e ritorna il risultato al client stesso.

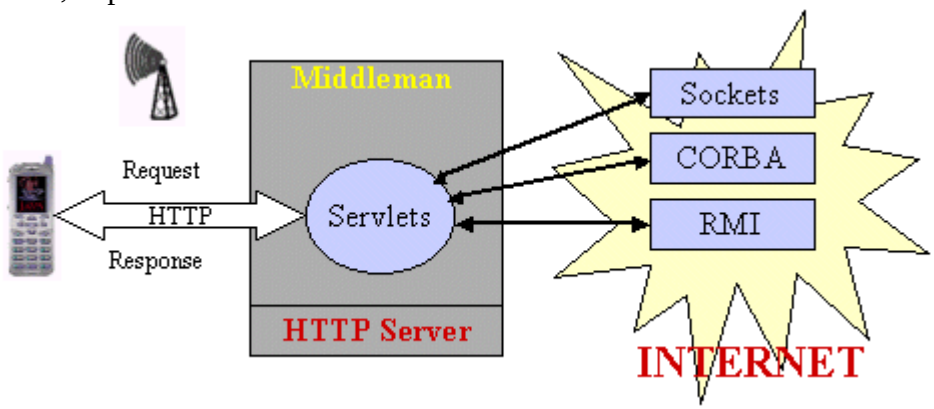

Figura 2.7: Middleman Architecture

2.5.8 Ricezione stream video da telecamere IP

Dovendo adattarsi alle tecnologie del laboratorio domotico e, in particolare, alle telecamere IP di cui è dotato, si è reso necessario ridisegnare la parte di acquisizione video, in quanto l'utilizzo delle *JMF* non prevede, come standard, lo sfruttamento di contenuti multimediali generati da telecamere IP. Dopo un'attenta valutazione si è optato per utilizzare il codec video MJPEG, ricevuto tramite il protocollo HTTP. Allo stato attuale del sistema questa scelta si è rivelata vincente sotto il punto di vista della semplicità d'implementazione, ma ha limitato fortemente alcune possibilità prestazionali e funzionali del sistema stesso. Un analisi più accurata verrà di questi aspetti, comunque, verrà discussa più avanti.

2.5.8.1 MJPEG

MJPEG (acronimo di Motion JPEG) [22] è un codec video nel quale ogni singolo frame del video viene compresso in una immagine JPEG. In questo modo non vi è nessuna compressione interframe, questo fa si che la qualità della compressione sia indipendente dal movimento presente nell'immagine, a differenza della compressione MPEG dove la qualità diminuisce quando il video contiene parecchie sequenze di movimenti veloci. Il bitrate è compreso tra il formato non compresso (come RGB, compressione 1:1, e YUV, compressione tra 1:1.5 e 1:2.5) e l'MPEG (1:100). Flussi dati di circa 29 Mbit/s hanno una elevata qualità, ma hanno una dimensione del file molto grande (paragonabile al Digital Video).

Questo formato è stato scelto sostanzialmente per la sua semplicità di implementazione nell'ambiente Java. Infatti è stato sufficiente rimuovere l'incapsulamento dei singoli frames in formato JPEG per poterli utilizzare direttamente in un pannello.

2.5.9 Adattamento all'EventEngine

La scelta di usufruire delle tecnologie esistenti all'interno del laboratorio domotico ha comportato la necessità di adattare lo sviluppo dell'applicazione alle specifiche predefinite. In particolare, per connettersi all'EventEngine si è dovuto utilizzare una socket sulla porta 1111, in modo da permettere una comunicazione bidirezionale. Inoltre, per facilitare l'integrazione, si è utilizzato lo standard XML per la formattazione dei file di configurazione.

2.6 Possibili scenari di utilizzo

L'intero sistema progettato si colloca all'interno del panorama domotico, in particolare nel settore sicurezza e assistenza.

Vengono qui presentati alcuni casi d'uso attualmente realizzabili con il sistema completo.

- **Security**:

  Intrusione: la presenza di qualsiasi entità all'interno dell'ambiente viene rilevata, quindi nel caso un malvivente riuscisse a forzare l'ingresso e a penetrare in casa, non appena entrerebbe nell'angolo di visualizzazione delle telecamere il proprietario verrebbe prontamente avvisato.

  Intrusione in zone particolari: tramite la configurazione delle telecamere è possibile rilevare il movimento solamente in zone predefinite visualizzabili nell'area coperta dalle stesse. E' possibile quindi rilevare la presenza di movimento nella zona della finestra o in quella della porta del terrazzo.

  Intrusione audio: tramite i sensori audio delle telecamere è possibile rilevare il superamento di un valore di soglia. La rottura di una finestra o di una porta quindi possono far scattare immediate contromisure.

  Monitoraggio: in qualsiasi momento è possibile collegarsi con il telefono cellulare per controllare da remoto le aree coperte dalle telecamere.

- **Safety**:

  Presenza utenza in zona pericolosa: tramite una corretta configurazione dell'EventEngine è possibile sfruttare il sistema di sensori wireless in modo da riconoscere la presenza di persone in aree che possono essere potenzialmente pericolose. Ad esempio se un bambino si trovasse in cucina in prossimità dei fornelli o del forno, il sistema provvederebbe a notificarlo nel migliore dei modi. La possibilità di definire qualsiasi area dell'ambiente sensibile alla presenza di persone incrementa notevolmente gli scenari di applicazione.
  Inoltre ai sensori wireless possono venire affiancate le telecamere, definendo aree specifiche in cui rilevare il movimento, migliorando notevolmente l'affidabilità totale del sistema.

  Monitoraggio: in qualsiasi momento è possibile collegarsi con il telefono cellulare per controllare da remoto le aree coperte dalle telecamere. Per esempio è possibile verificare la condizione del bambino solo in casa, oppure eventuali anomalie all'interno dell'ambiente, come la presenza di fuoco o di acqua.

- **Assistenza**:

  Rilevazione audio: tramite i sensori audio delle telecamere è possibile ricevere una notifica, ad esempio, se viene registrato il pianto del bambino.

  Monitoraggio distanza: tramite il telefono cellulare è possibile controllare visivamente lo stato di una persona malata o di un anziano, contattando immediatamente i servizi di soccorso nel caso di anomalie.

- **Gestione**:

  Tramite il controllo remoto dei dispositivi è possibile controllare in modo basilare (accensione/spegnimento o apertura/chiusura) qualsiasi dispositivo collegato alla scheda relé dell'EventEngine. E' quindi possibile, ad esempio, intervenire sulla valvola del gas nel caso sia stata rilevata la presenza di fumo e quindi di fiamme libere.

  Il ruolo gestionale non si limita solamente agli aspetti legati alla sicurezza, ma può permettere di incrementare il comfort di un ambiente controllando tramite il telefono cellulare le luci, le tapparelle, gli elettrodomestici, l'entertainment…

2.7 Strumenti di sviluppo

Tutti i programmi descritti sono stati sviluppati in *Java* (J2SE e J2ME) su *Windows XP*.
Come ambiente di sviluppo si è usato: *NetBeans 6.0.1*, IDE open source della Sun Microsystems, per il progetto MotionDomus. Eclipse SDK 3.4.2 per lo sviluppo degli applicativi RMI e di tutto il progetto ControlDomus, compreso il client per cellulare. Infine si è utilizzato JDeveloper 2005 Enterprise Edition e Tomcat per la creazione della servlet per il sistema di tracciabilità dell'indirizzo IP.
In combinazione con Eclipse sono stati utilizzati due particolari plug-in che hanno aiutato nel processo di sviluppo:

- EclipseME J2ME Developement Tools for Eclipse (versione 1.7.9): solleva il programmatore dalle problematiche di collegamento tra i Wireless Toolkits e l'ambiene di sviluppo di Eclipse, consentendogli di focalizzarsi solamente sulla parte di scrittura del codice.
- Genady's RMI Plug-in for Eclipse[x] (versione 2.1.0) ha permesso di sviluppare in modo semplice e chiaro una tipologia di applicazioni (le RMI appunto) che, altrimenti, avrebbe richiesto una mole di lavoro decisamente maggiore.

Lo strumento fondamentale utilizzato per lo sviluppo dell'applicazione destinata al dispositivo mobile è stato il *Sun Java Wireless Toolkit for CLDC (WTK)* versione 2.5.2. Esso fornisce tutti i software essenziali per la gestione dei progetti, per la compilazione e il packaging del codice binario, l'emulazione su PC, il controllo di tutti i parametri operativi in fase di esecuzione. Con il proseguimento del lavoro il Wireless Toolkit è stato affiancato dalla nuova *Java Platform Micro Edition SDK 3.0*, sviluppata sempre da Sun Microsystems ed erede proprio del *Java Wireless Toolkit for CLDC (WTK)* versione 2.5.2. Le novità comprendono, tra le altre cose, un set più ampio di emulatori, l'integrazione con la tecnologia Blu-ray Disc Java (BD-J) e un ambiene di sviluppo stand-alone.

Tenendo a mente il fatto che i test su dispositivi reali sarebbero stati eseguiti unicamente su di un Nokia 5800 XpressMusic, si è voluto affiancare alle altre piattaforme anche il Nokia SDK S60 platform for Symbian per eseguire i test su un emulatore specifico per una particolare categoria di dispositivi.

E' doveroso soffermarsi sull'importanza degli emulatori (figura 2.11) e dei test sui dispositivi reali. Le applicazioni desktop possono essere testate durante tutto il processo di sviluppo eseguendole sulla stessa workstation utilizzata per la scrittura del codice o su una macchina dedicata. Nel caso dello sviluppo di applicazioni Java ME tale assunto non è più valido: esse, infatti, saranno eseguite certamente su un sistema operativo differente (spesso proprietario) e con hardware ben diverso da quello di un computer tradizionale. I test, dunque, dovrebbero essere condotti direttamente su dispositivi, al fine di verificare il corretto comportamento dell'applicazione in ogni situazione. Purtroppo i test su hardware reale non sono sempre possibili. Emerge, quindi, l'esigenza di disporre di strumenti software che consentono di effettuare test significativi e attendibili delle applicazioni direttamente su PC senza utilizzare un dispositivo fisico. Tali strumenti, gli emulatori appunto, permettono al programmatore di eseguire gran parte dei test dell'applicazione sul PC e di isolare e risolvere buona parte dei bug eventualmente presenti nel codice. Il livello di fedeltà dell'ambiente emulato può variare a seconda della complessità dell'emulatore ma, per quanto fedele possa essere, non può sostituire il test su dispositivi reali. Quest'ultimo infatti è l'unico oggetto sul quale è possibile riprodurre tutti i casi di utilizzo e le sollecitazioni a cui l'utente sottoporrà l'applicazione.

Figura 2.11: i tre emulatori usati per i test
Wireless Toolkit 2.5.2 (a sinistra)
Java ME platform SDK 3.0 (al centro)
Nokia S60 3rd Edition for Symbian (a destra)

2.8 Conclusioni di capitolo

In questo capitolo l'attenzione è stata spostata sul sistema da realizzare, definendo uno scenario generale di operatività e dei requisiti funzionali che il prodotto finale dovrà rispettare. La trattazione poi è proseguita con l'identificazione delle entità coinvolte all'interno del sistema, descrivendo sinteticamente le loro caratteristiche. In particolare l'attenzione è stata focalizzata sulle scelte software utilizzate per l'implementazione di tali caratteristiche, spiegando il funzionamento e gli obiettivi da raggiungere.

Capitolo 3

ANALISI E PROGETTAZIONE

Il processo di analisi e progettazione si è basato sostanzialmente sulla scelta di costruire l'applicazione passo per passo, partendo da un nucleo base, aggiungendo gradualmente le varie funzionalità durante la fase di sviluppo. Questo approccio ha permesso di testare gradualmente le varie applicazioni, man mano che venivano aggiunte le funzionalità.
L'integrazione delle varie entità è così risultata semplice ed immediata, consentendo quindi di riutilizzare alcune funzionalità, appartenenti alla prima fase del lavoro, anche sul secondo sistema.

3.1 Panoramica

L'applicazione da progettare e sviluppare deve sostanzialmente permettere la connessione di un telefono cellulare (o di un dispositivo mobile) con le webcam e con telecamere di tipo IP. Questa connessione deve poter essere definita manualmente, tramite la specifica decisione dell'utente, oppure in modo automatico, tramite un particolare evento scatenante (rilevazione movimento, presenza in zone pericolose…). Inoltre il sistema deve offrire alcune funzionalità aggiuntive e di supporto. Riferendoci alla prima parte del progetto, ovvero il sistema Motion Domus, le funzionalità aggiuntive sono state individuate nel salvataggio di screenshot e nella trasmissione RTP delle immagini. La seconda parte, chiamata ControlDomus, offre l'interessante possibilità di comandare l'accensione e lo spegnimento di dispositivi tramite il telefono cellulare.

Il contesto in cui si trova l'applicazione è evidenziato nella figura 3.1, dove vengono messe in evidenza tutte le entità coinvolte nel sistema globale.

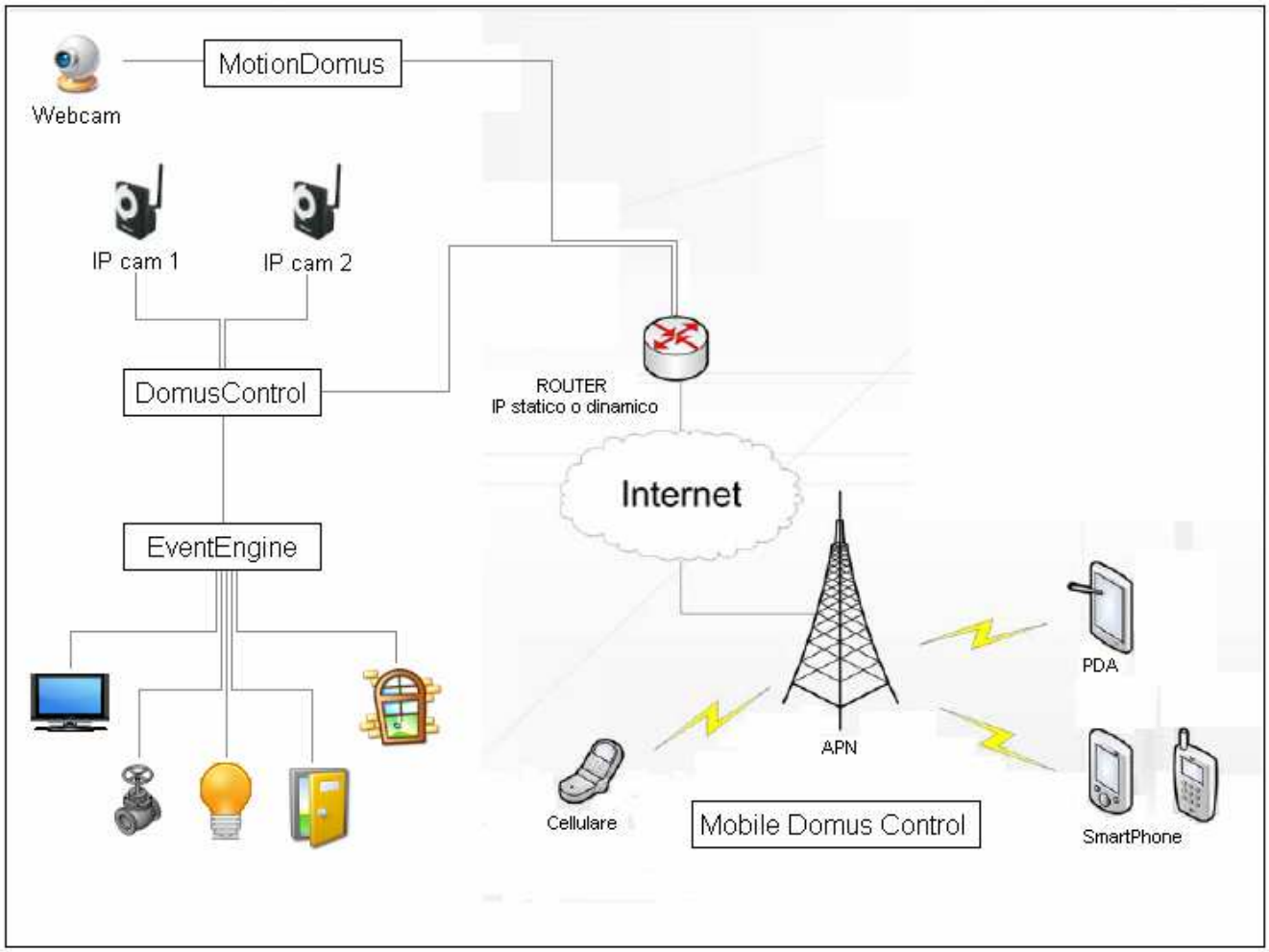

Figura 3.1: panoramica generale del sistema completo

L'impianto è collegato in Internet e deve essere reso accessibile da un qualsiasi terminale mobile, in modo tale da poter visualizzare le immagini sul display cercando di soddisfare le esigenze imposte dal sistema. Prima fra tutte la necessità di operare in tempo reale, esigenza tanto più sentita quanto più è importante l'area sottoposta al monitoraggio. Un altro fattore di notevole importanza nasce dal fatto che il servizio possa essere fruibile in modo (quasi) indipendente dal tipo di dispositivo posseduto; per questo durante la fase di progettazione risulterà necessario utilizzare componenti e moduli "universali" indipendenti quindi dalla piattaforma di sviluppo in uso.

3.2 Architettura del sistema

Il sistema totale si compone di 5 parti differenti, considerati *micro-sistemi* interagenti tra di loro, tramite una specifica gerarchia, la cui sommatoria compone l'architettura generale.

Il sistema totale è formato dalle seguenti parti, già introdotte nel secondo capitolo:

- MotionDomus: l'applicazione lato server che espone l'interfaccia con la webcam e permette l'invio delle immagini al client su dispositivo mobile.
- Sistema di tracciabilità dell'indirizzo IP
- EventEngine
- ControlDomus: questo progetto è a sua volta suddiviso in due parti:

- Domus Control: l’applicazione che espone l’interfaccia con le telecamere IP e con l’EventEngine.
- Mobile Domus Control: il client per dispositivo mobile che consente di collegarsi sia con il MotionDomus, sia con il Domus Control.

3.3 Funzionalità

I micro-sistemi MotionDomus e Domus Control sono stati progettati seguendo un'architettura 3-tier. Questa architettura è composta da:

- Presentation Tier: è il livello superiore dell'applicazione. Il livello presentazione si occupa della parte d’interaccia tramite cui vengono mostrate all’utente le funzionalità del programma.
- Business Logic: è lo strato intermedio ed elabora i dati ed i messaggi in input implementando i requisiti dell'applicazione.
- Data Tier: è lo strato relativo alla memorizzazione e raccolta dei dati.

Durante la progettazione del MotionDomus sono state individuate le seguenti funzionalità da implementare:

- Raccolta delle informazioni su configurazione da file XML
- Interfacciamento con la webcam e creazione dello stream
- Inizializzazione e configurazione del meccanismo di Motion Detection
- Invio dei messaggi SMS di attivazione
- Comunicazione, tramite socket, con dispositivi mobili
- Invio delle immagini in rete tramite una sessione di streaming RTP
- Salvataggio di screenshot

Per il sistema di tracciabilità dell’indirizzo IP funzionalità da implementare sono state le seguenti:

- Autenticazione del client per ambiente Desktop che comunica l’indirizzo IP del PC su cui risiede
- Recupero del proprio indirizzo IP
- Salvataggio dell’indirizzo e delle informazioni di autenticazione all’interno del database
- Recupero dell’indirizzo da parte dell’applicazione per dispositivo mobile

Durante la progettazione del Domus Control sono state individuate le seguenti funzionalità da implementare:

- Raccolta delle informazioni su configurazione, comandi e notifiche da files XML
- Interfacciamento con le telecamere IP e scomposizione degli stream video
- Inizializzazione e configurazione del meccanismo di Motion Detection
- Invio dei messaggi SMS di attivazione e di testo
- Comunicazione, tramite socket, con dispositivi mobili
- Comunicazione, tramite socket, con l’EventEngine
- Ricezione di notifiche dalle telecamere
- Elaborazione e validazione dei dati in ingresso
- Attuazione dei comandi
- Memorizzazione degli eventi occorsi a run-time.

Durante la progettazione del Mobile Domus Control sono state individuate le seguenti funzionalità da implementare:

- Configurazione dei parametri necessari al funzionamento
- Attivazione alla ricezione di particolari messaggi SMS
- Comunicazione tramite socket, sia con MotionDomus, sia con ControlDomus
- Visualizzazione delle immagini
- Raccolta delle informazioni da XML riguardanti i dispositivi controllabili
- Invio dei comandi di accensione/spegnimento

3.4 Diagramma dei Package

In questa sezione vengono descritti i package e i loro contenuti relativi ai progetti.

Il diagramma dei package del MotionDomus ha la struttura descritta dalla seguente figura:

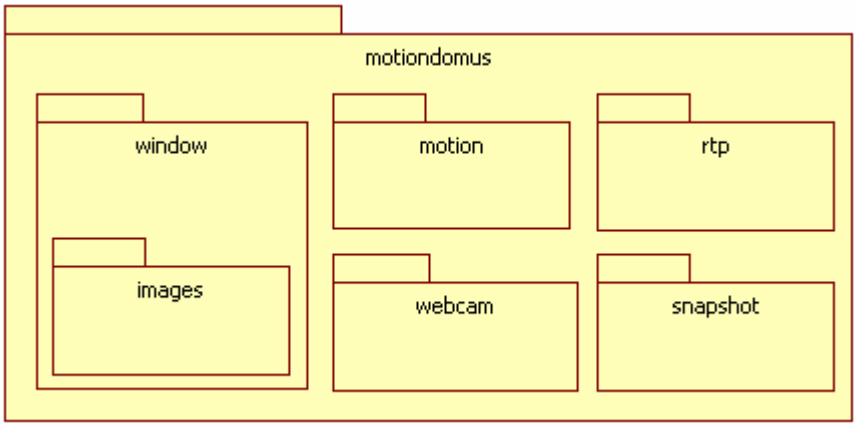

Figura 3.2: diagramma dei package del MotionDomus

All'interno del package `window` sono inserite le classi principali che definiscono il nucleo del sistema. Oltre alla classe `Main`, quindi, trovano collocazione le classi `MainWindow`, `Utility`, `Configuration`, `ConfigureCameraDialog`, `GraphPanel`, `Initialisator` e `AboutBox`.

Il package `window` ne contiene un ulteriore, chiamato `images`. Questo non comprende codice, ma solamente le immagini utilizzate nell'applicazione.

Il package `motion` contiene le classi necessarie al funzionamento del meccanismo di Motion Detection. Queste classi sono: `MotionDetector`, `SimpleMotionDetector`, `DataSourceConsumer` e `DataStreamEmptier`.

Il package `webcam` contiene una singola classe, chiamata `ClientServer`. Questa ha il compito di formattare nel modo più appropriato le immagini ricevute dalla webcam e di inviarle al client, residente su dispositivo mobile, che ne ha fatto richiesta.

Il package `rtp` contiene due classi dedite alla creazione della sessione RTP tramite cui è possibile inviare immagini in rete verso altri computer. Queste classi sono `AVTransmit` e `AddressRtpBox`.

Infine il package `snapshot` contiene la singola classe `SnapshotHandler` che si occupa, alla pressione di un bottone, di catturare e salvare in formato JPEG uno screenshot dello stream video.

Il diagramma dei package del Domus Control ha la struttura descritta dalla seguente figura:

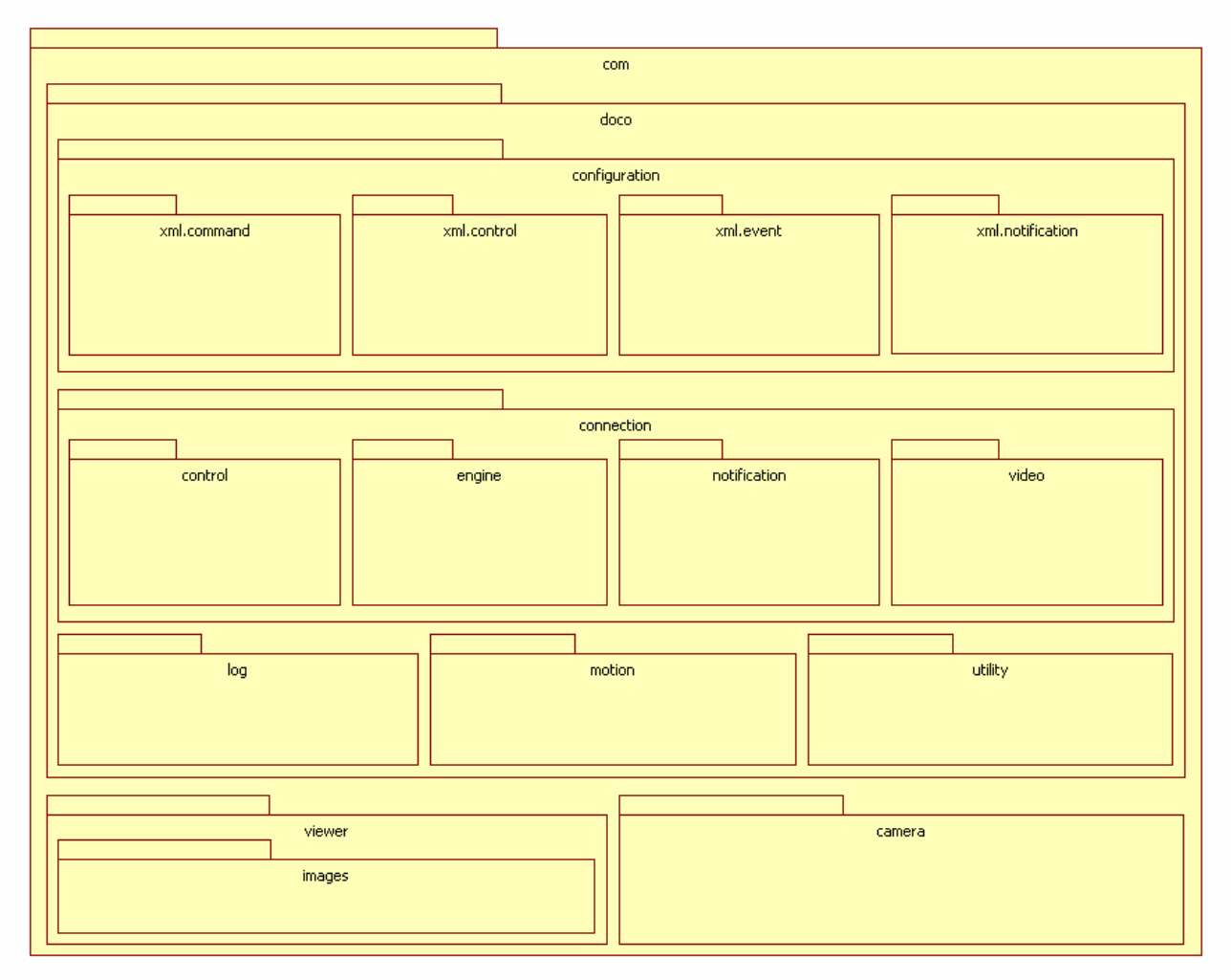

Figura 3.3: diagramma dei package del Domus Control

Tutti i package sono contenuti all'interno di un'unica cartella `com`. Il package che contiene le classi dedicate alla costruzione e alla gestione dell'interfaccia ha nome `viewer` e racchiude le classi `Main`, `MainWindow`, `Blink` e `Options`. Comprende inoltre il package che contiene le immagini utilizzate dall'applicazione. Il package `camera` invece contiene due classi adibite alla connessione con una singola telecamera IP, queste sono `IPCamera` e `MyAuthenticator`.

All'interno del package `doco` sono contenuti 5 package che racchiudono le altre classi dell'applicazione. Essi sono:

- `log`: contiene l'omonima classe Log, che si occupa del meccanismo di log delle azioni
- `motion`: contiene le classi per gestire il meccanismo di Motion. Esse sono MotionDetector e SimpleMotionDetector
- `utility`: contiene un'unica classe Utility, con le funzioni di invio dei messaggi SMS
- `connection`: contiene a sua volta 4 package distinti, per gestire le 4 tipologie di connessione del programma:
    - `control`: con le classi `ControlConnection`, `ControlListener` e `ControlMessage`

- `engine`: con le classi `EngineClient`, `EngineListener` ed `EngineMessage`
- `notification`: con le classi `NotificationConnection` e `NotificationListener`
- `video`: con le classi `ClientServer` e `VideoListener`

- `configuration`: contiene due classi per gestire la configurazione generale dell'applicazione, chiamate `Configuration` e `InitXML`, ed altri 4 ulteriori package, specifici per la lettura dei 4 file xml:

  - `xml.command`: racchiude le classi `Command`, `CommandsParser`, `CommandsSequence`, `Number`, `NumbersSequence`
  - `xml.control`: con le classi `ControlCommand`, `ControlCommandsParser`, `ControlCommandsSequence`
  - `xml.event`: contenente le classi `Event`, `EngineEventsParser` e `EventsSequence`
  - `xml.notification`: con le classi `Notification`, `NotificationsParser` e `NotificationsSequence`.

Infine consideriamo il diagramma dei package dell'applicazione client per dispositivi mobili, descritto dalla figura 3.4:

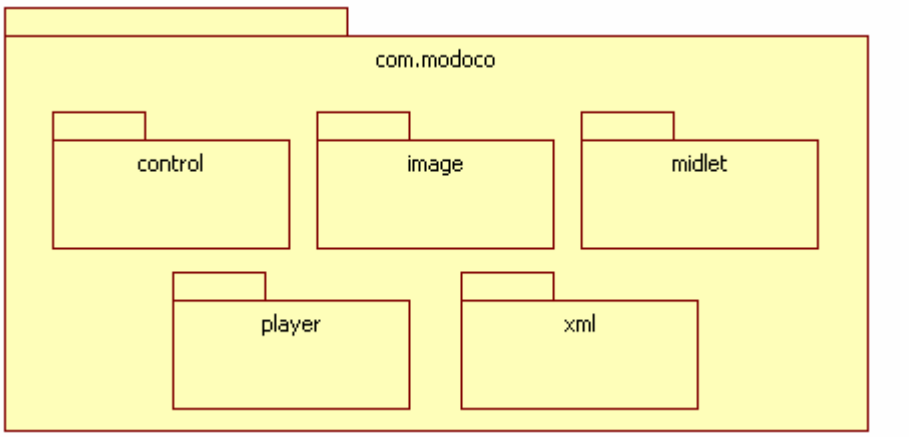

Figura 3.4: diagramma dei package del Mobile Domus Control

La cartella `com.modoco` contiene i 5 package separati che racchiudono le classi dedicate per ogni funzionalità della MIDlet:

- `control`: contiene le classi per la connessione con il DomusControl e la gestione degli elementi modificabili. Sono `ControlConnection`, `ControlMessage`, `ElementStruct`, `MyNode`, `NetworkListener` e `XmlListener`
- `image`: contiene solamente le icone utilizzate dalla MIDlet
- `midlet`: racchiude le classi principali per il funzionamento della MIDlet ovvero `MidletControl`, `MobileDomusController`, `SMSReceive`, `Memory` e `Utility`
- `player`: con le classi per la ricezione e la visualizzazione delle immagini, `VideoClient` e `WebCam`
- `xml`: con le classi per la lettura e il parsing dei file XML, `XMLHandler` e `XMLParser`

3.6 Diagramma delle classi

Verranno ora illustrati i diagrammi delle classi dei due progetti realizzati, assieme ad una descrizione sintetica del contenuto e della funzione di ogni classe. Per esigenze di praticità si è preferito omettere gli attributi e i metodi dai diagrammi, in modo da non appesantire la visualizzazione delle immagini. Inoltre, nel caso del *Domus Control*, si è suddiviso il diagramma delle classi in schemi più piccoli, in base alle funzionalità, in modo da favorire la comprensione.

Il diagramma delle classi del *MotionDomus* è descritto dalla figura seguente:

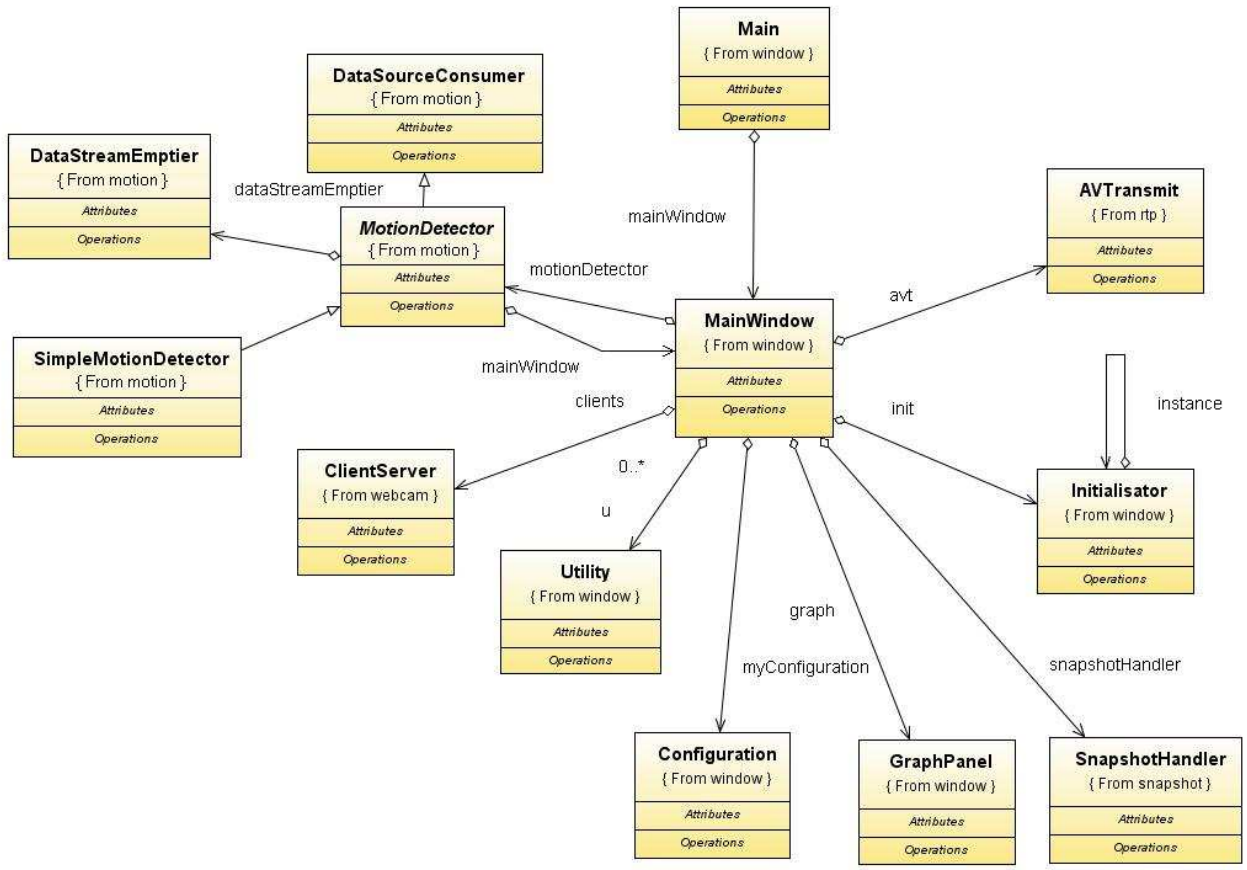

Figura 3.5: diagramma delle classi del MotionDomus

Al centro del sistema c'è la classe `MainWindow`, responsabile dell'interfaccia grafica. Questa richiama le altre classi del sistema in base alle funzioni che deve eseguire. La gestione della configurazione è affidata alla classe `Configuration` mentre per l'accesso alla webcam viene utilizzata la classe `Initialisator`, che espone i metodi per il controllo dello stream video. La classe `GraphPanel` è responsabile della creazione del grafo che visualizzarà i valori di motion, valori definiti dalle classi `MotionDetector` e `SimpleMotionDetector`. Le classi `DataStreamEmptier` e `DataSourceConsumer` si occupano di gestire lo stream video e fornire le immagini per il meccanismo di motion. L'invio delle immagini in rete, alla MIDlet e ad altri PC tramite sessione RTP, è gestito rispettivamente dalle classi `ClientServer` e `AVTransmit`. Infine le classi `SnapshotHandler` e `Utility` hanno il compito rispettivamente di catturare screenshot del video e fornire i metodi per l'invio degli SMS.

Per il caso del *Domus Control*, come detto, si è preferito suddividere il diagramma principale in 4 sotto-diagrammi, in modo da analizzare con più praticità le vaire caratteristiche. La suddivisione tiene conto delle seguenti funzionalità:

- Gestione della configurazione XML
- Motion Detection, Connessione Video e Interfaccia
- Configurazione e connessioni verso l'esterno
- Log

Il diagramma delle classi adibite alla configurazione XML è definito in figura 3.6:

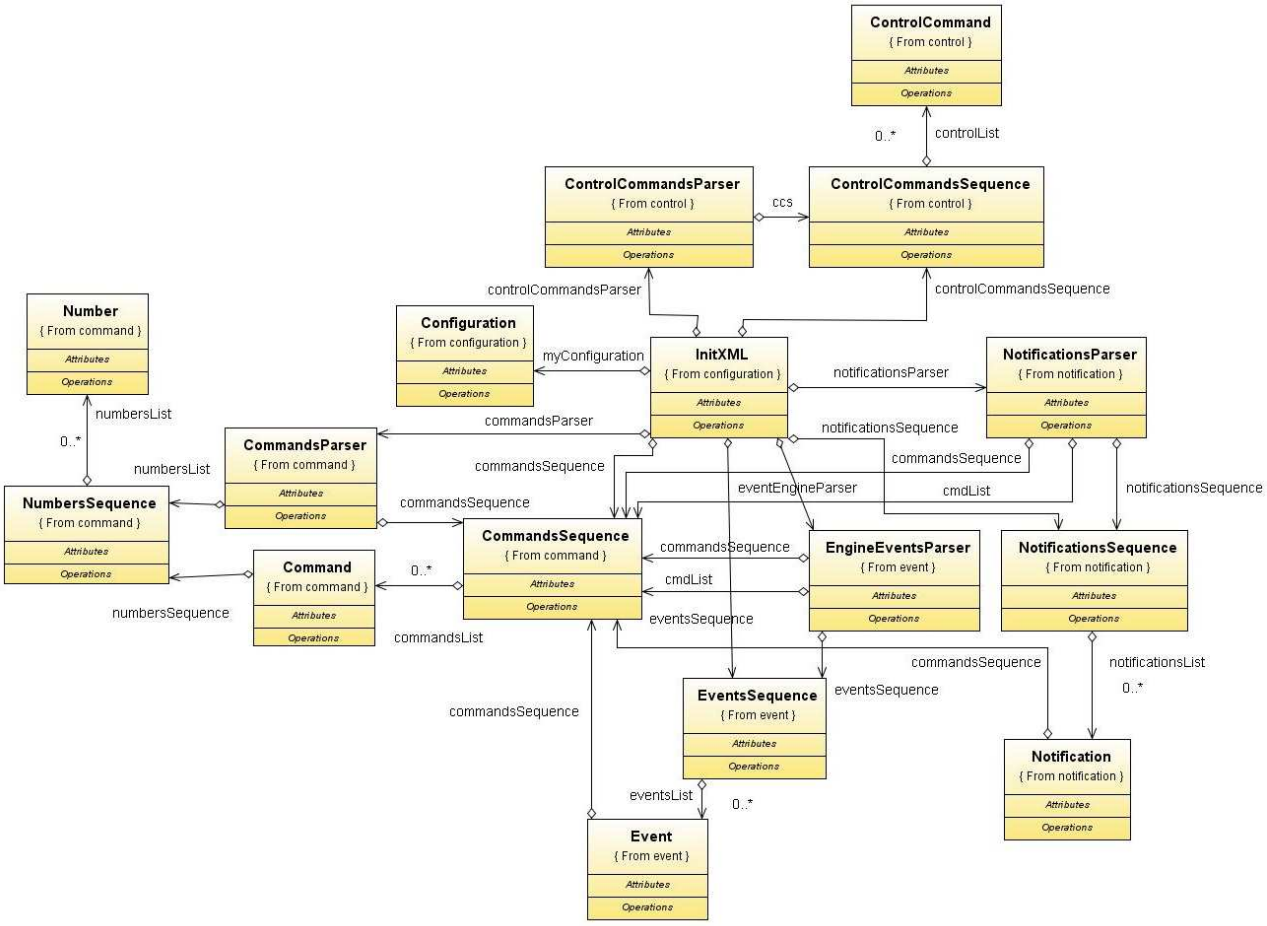

Figura 3.6: diagramma delle classi per la configurazione XML

Il punto di partenza per la gestione dei file XML è la classe `InitXML`, partendo dalla configurazione definita dalla classe Configuration istanzia i 4 parsers per i rispettivi files XML: `ControlCommandsParser`, `NotificationParser`, `EngineEventsParser` e `CommandsParser`. I dati vengono formattati in apposite strutture determinate dalle classi `ControlCommandsSequence`, `NotificationsSequence`, `EventsSequence`, `CommandsSequence` e `NumbersSequence` che a loro volta sono costituite da sequenze di dati elementari definiti dalle classi `ControlCommand`, `Notification`, `Event`, `Command` e `Number`. In particolare queste rappresentano rispettivamente un comando di controllo dei dispositivi, una notifica delle telecamere, un evento generato dall'EventEngine, un comando di risposta e un numero di telefono.

Il diagramma delle classi che rappresenta il meccanismo di Motion Detection, la connessione video e l'interfaccia è il seguente:

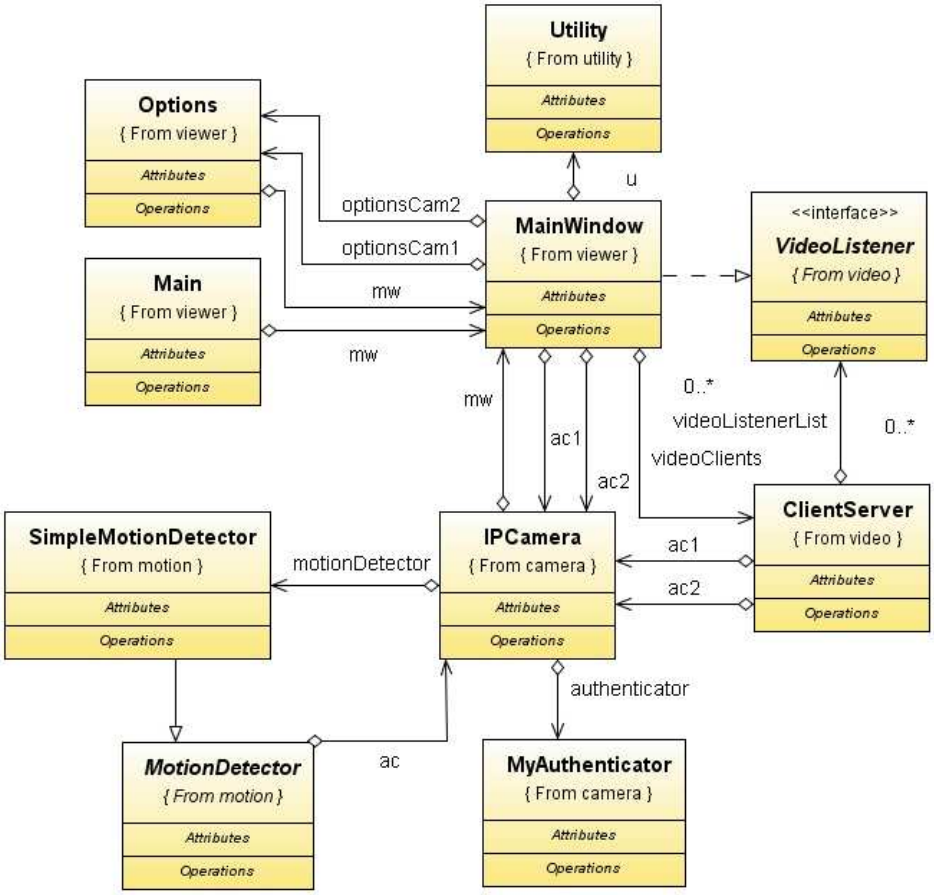

Figura 3.7: diagramma delle classi per il Motion Detection, connessione video e interfaccia

Il punto di partenza è la classe `MainWindow` che crea l'interfaccia ed istanzia le altre classi per le funzioni del software. Dal diagramma è evidente come vengono utilizzate due istanze della classe `IPCamera`, la quale si occupa di ricevere il video da una singola telecamera. Unito a questa ci sono le classi per il Motion Detection, `MotionDetector` e `SimpleMotionDetector`, e la classe per l'autenticazione alla telecamera, ovvero `MyAuthenticator`. L'invio delle immagini alla MIDlet è, analogamente al MotionDomus, definito all'interno della classe `ClientServer`. A questa è ovviamente associata l'interfaccia `VideoListener` che notifica alla `MainWindow` i messaggi scambiati con la MIDlet.

Per quanto riguarda la configurazione e le connessioni con l'esterno il diagramma è dato dalla figura 3.8:

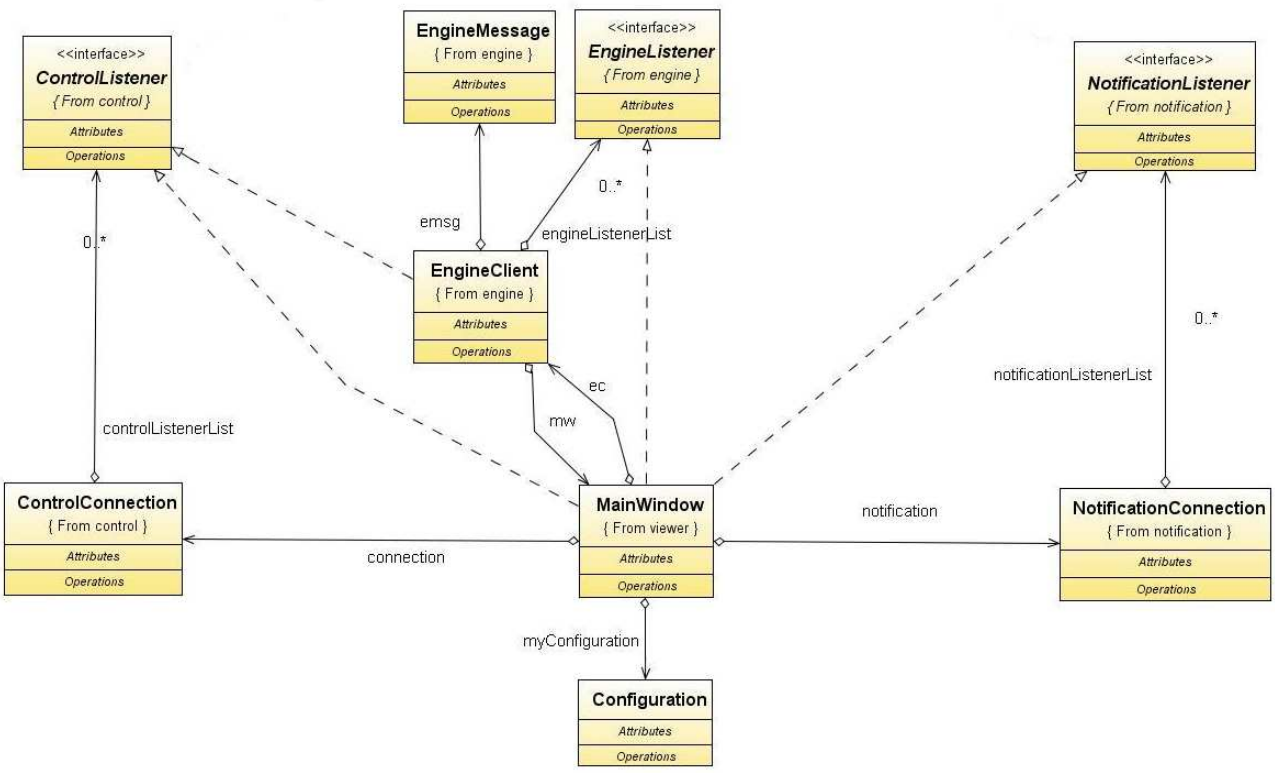

Figura 3.8: diagramma delle classi focalizzato sulle connessioni

La classe MainWindow istanzia tre classi – `EngineClient`, `NotificationConnection` e `ControlConnection` – responsabili delle connessioni con l'esterno. I dati per queste connessioni sono contenuti all'interno della configurazione definita dalla classe `Configuration`. Ogni connessione, inoltre, è associata ad una interfaccia, in modo da notificare alla `MainWindow` lo scambio di messaggi. Queste interfacce sono rispettivamente `EngineListener`, `NotificationListener` e `ControlListener`.

Infine il diagramma che definisce il meccanismo di Log è rappresentato dalla seguente figura:

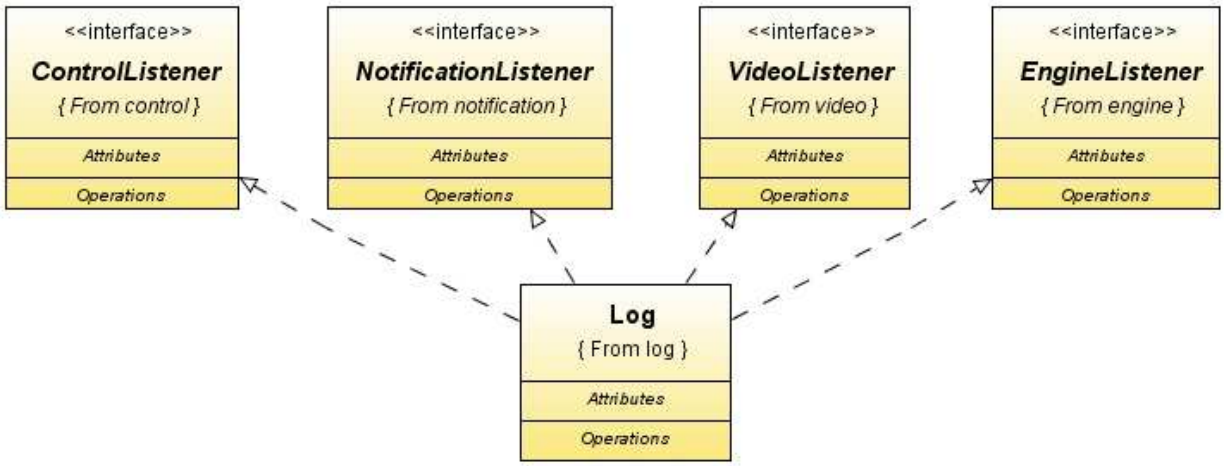

Figura 3.9: diagramma delle classi per il Log su file

La classe `Log` dispone dei metodi definiti dalle interfacce `ControlListener`, `NotificationListener`, `VideoListener` e `EngineListener`, in modo da registrare su file lo scambio dei messaggi.

Infine esaminiamo il diagramma delle classi della MIDlet *Mobile Domus Control*, rappresentato dalla seguente figura:

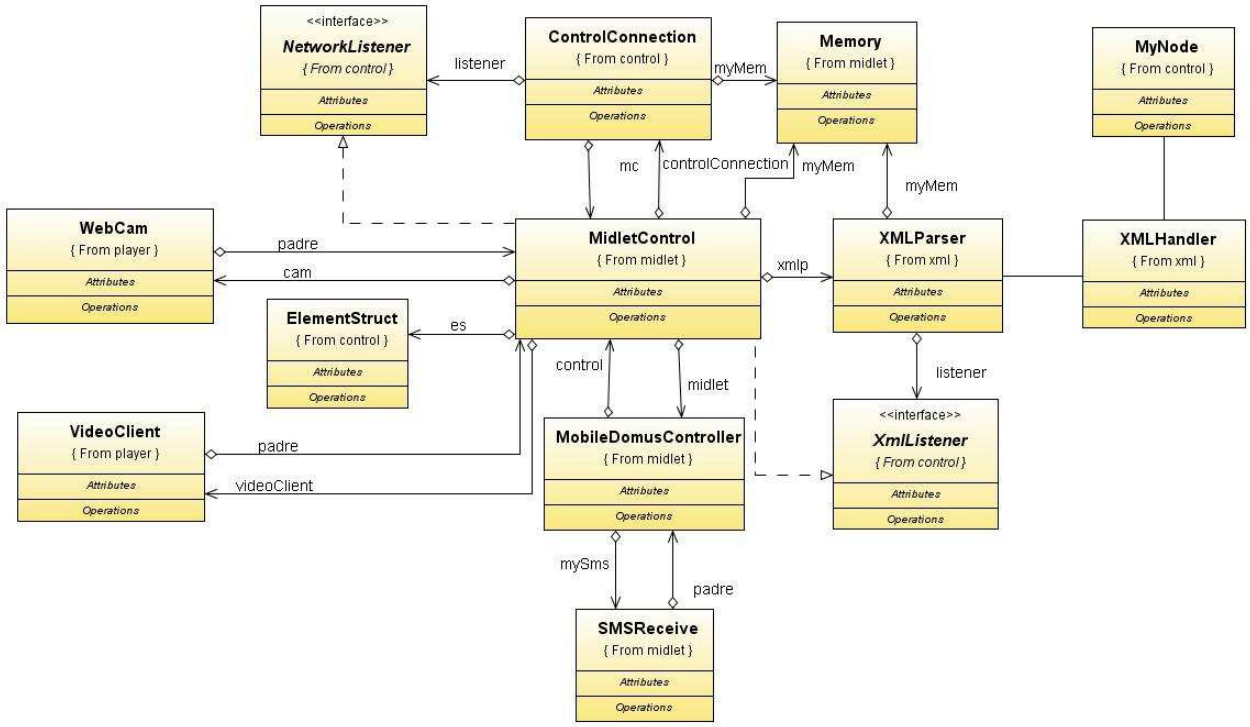

Figura 3.10: diagramma delle classi della MIDlet MoDoCo

Al centro dell'architettura c'è la classe `MobileDomusController`, il punto di accesso della MIDlet. Questa istanzia la classe `SMSReceive` per gestire gli SMS in ingresso e la classe `MidletControl`, che si occupa di gestire la completa applicazione tramite le classi associate. Queste sono, quindi, le classi `VideoClient` e `WebCam` per la connessione e gestione del video ricevuto dal `Domus` Control, la classe `XMLParser` associata a `XMLHandler` e `MyNode` per il parsing del file XML degli elementi, la classe `ControlConnection` per la connessione di controllo e la classe `Memory` per la memorizzazione permanente tramite il RMS. Le interfacce associate a `MidletControl` sono `XmlListener` e `NetworkListener`. Infine la classe `ElementStruct` ha il compito di definire la struttura di un singolo elemento modificabile.

3.7 Conclusioni di capitolo

Questo capitolo si è concentrato maggiormente sulla fase di analisi e progettazione considerando, ad alto livello, l'architettura logica del sistema globale. In particolare si sono definiti i ruoli delle entità coinvolte e le varie funzionalità da implementare.
Infine la trattazione è proseguita con l'analisi dei diagrammi dei package e delle classi, in modo da analizzare, con maggiore dettaglio, l'architettura funzionale delle singole entità e delle loro interconnessioni.

Capitolo 4

MotionDomus

La prima parte del progetto ha preso il nome di *MotionDomus*. Motion ad indicare la capacità di rilevare movimento, Domus invece indica l'ambito di applicazione domestico.

4.1 Requisiti

Requisito indispensabile per poter utilizzare il programma è disporre di una webcam compatibile con JMF[xi]. In generale questa compatibilità è garantita, in particolar modo, su Windows, poiché le webcam sono quasi sempre dotate dei drivers VFW.

Altro requisito essenziale è, ovviamente, l'installazione del *Java Media Framework*. Una volta scaricata la versione appropriata per il proprio sistema dal sito internet[xii], è necessario registrare la webcam tramite il registro delle JMF.

Bisogna quindi eseguire, dal menu d'avvio, il programma JMF Registry presente nel gruppo Java Media Framework. Nella scheda Capture Devices (figura 4.1) è necessario cliccare quindi su Detect Capture Devices ed attendere che il programma completi la scansione del pc alla ricerca di devices compatibili con JMF. Se tutto è andato a buon fine si dovrebbe essere in una situazione simile a quella sottostante:

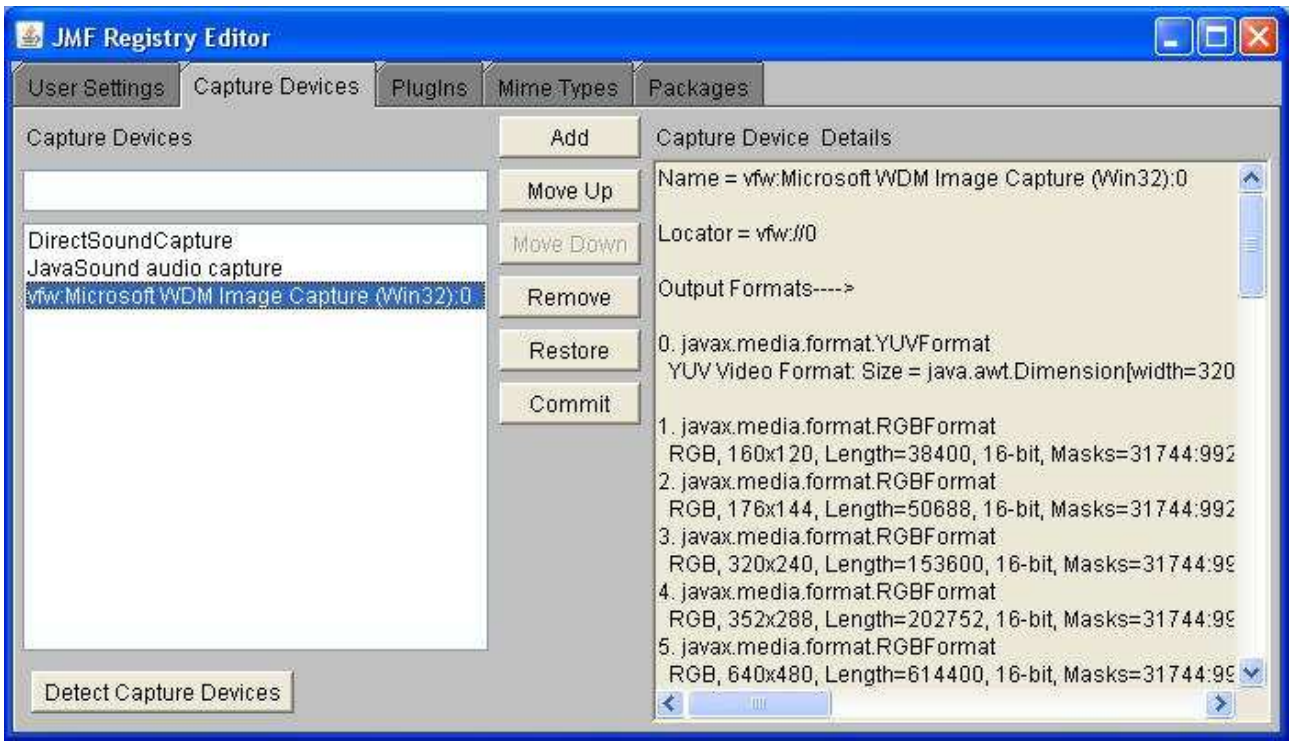

Figura 4.1: la schermata principale del JMF Registry

Nella parte destra sono indicati tutti i formati supportati dal device.

4.2 Funzionamento

Per lanciare il programma è sufficiente digitare `java –jar MotionDomus.jar` all'interno di un terminale. E' importante assicurarsi che nella stessa directory del programma ci sia la cartella `lib` contenente le librerie fornite dal *JMF* e necessarie al funzionamento. In caso contrario il programma genrerà un eccezione. E' anche possibile specificare la directory tramite linea di comando, utilizzando l'opzione `–classpath` del comando `Java`. Al primo avvio verrà creato il file `config.xml` contenente la configurazione e un file di testo chiamato `Surveillance.properties`.

4.3 Interfaccia grafica:

Prima di osservare l'interfaccia grafica principale si può osservare, al primo avvio dell'applicazione, l'apparizione di una finestra di dialogo che permette la scelta del dispositivo e del formato video da utilizzare. Queste preferenze vengono salvate in un file chiamato *`Surveillance.properties`* richiamato ogni qualvolta viene rieseguito il software, in modo da evitare la finestra di dialogo iniziale. La struttura del file è simile alla seguente:

```
#Wed Jun 24 14:28:23 CEST 2009
Device=vfw\:Microsoft WDM Image Capture (Win32)\:0
Format=RGB, 640x480, Length\=921600, 24-bit, Masks\=3\:2\:1, PixelStride\=3,
LineStride\=1920, Flipped
```

Viene creato, inoltre, un file di configurazione in linguaggio XML. Questo file, chiamato *`config.xml`*, contiene i parametri di configurazione:

- **device_index**: l'indice del device registrato nel JMF Registry (default: 2)
- **width**: larghezza del frame video (default: 320)
- **height**: altezza del frame video (default: 240)
- **messaggio**: informazione per il server d'invio SMS (default: Attenzione)
- **numero_telefono**: numero del cellulare che riceverà gli SMS
- **utente**: nome utente per il servizio d'invio SMS
- **password**: password per il servizio d'invio SMS
- **port**: porta su cui il programma ascolterà le connessioni d'ingresso (default: 1983)
- **serverName**: nome del server d'invio degli SMS (default: Mike Server)

Tale file viene richiamato ad ogni avvio del programma. Nel caso non fosse presente viene ricreato con la configurazione di default.

L'interfaccia principale, illustrata in figura 4.2, si compone in due parti (sinistra e destra), ognuna formata da tre pannelli.

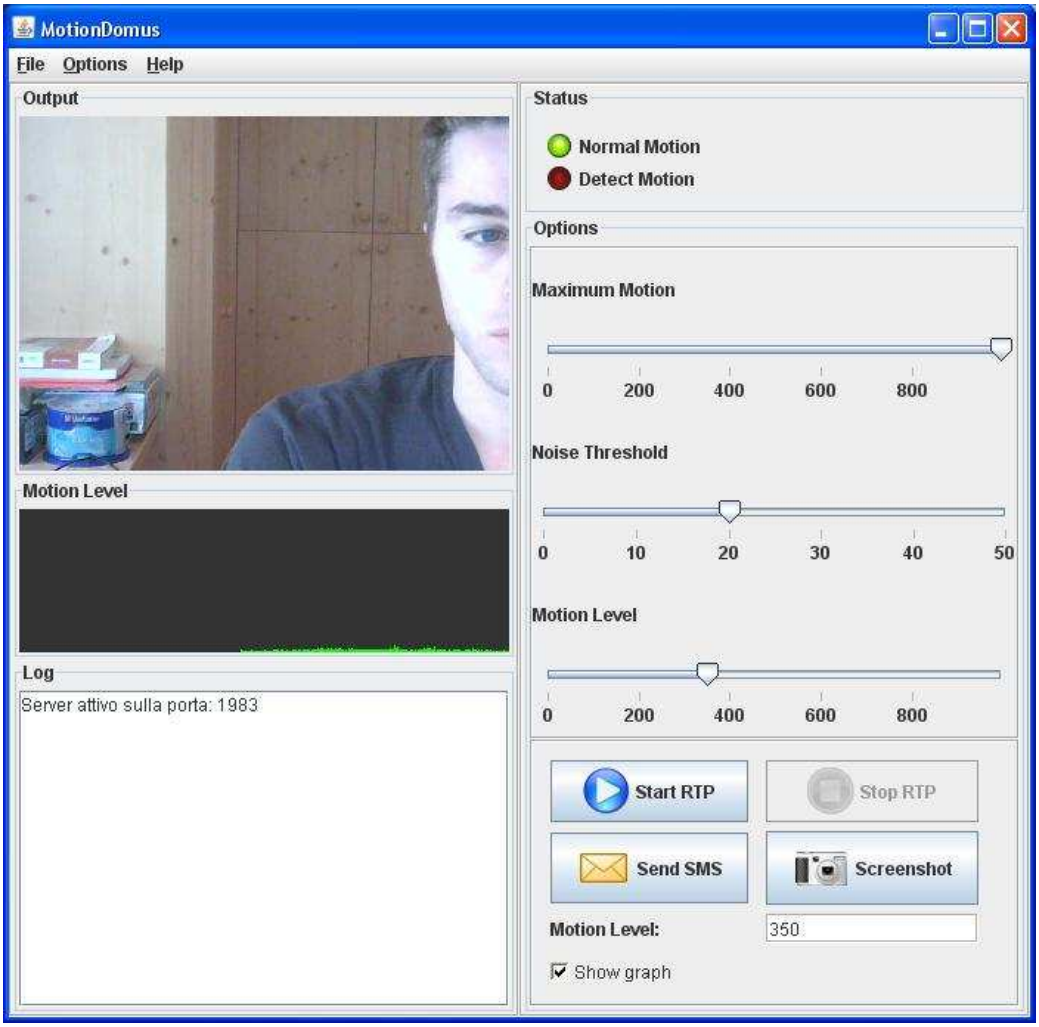

Figura 4.2: l'interfaccia principale del MotionDomus

A sinistra troviamo:

- Il pannello di *Output*, che mostra direttamente le immagini della webcam
- Un grafo che indica il livello di Motion registrato
- Un area di log che visualizza i messaggi informativi per l'utente

A destra invece troviamo:

- Un pannello *Status* che ci indica visivamente, tramite due led, se il livello di motion è nei livelli di guardia oppure li supera
- Un pannello *Options* contenente 3 JSlider che consentono di personalizzare alcuni parametri:
  - *Maximum Motion*: definisce il valore massimo di Motion che può essere registrato. Questo infatti può assumere valori compresi tra 0 e 999.
  - *Noise Threshold*: indica il valore di soglia del rumore. E' importante impostarlo correttamente perché permette virtualmente al meccanismo di Motion di ignorare il tipico rumore di fondo che ogni dispositivo di acquisizione video, ma in particolare la webcam, possiede.
  - *Motion Level*: questa slider permette di impostare il livello di Motion, al di sopra del quale scattano i provvedimenti

- Un pannello contenente i pulsanti di azione, una textBox informativa che comunica l'esatto valore regolato dalla slider del Motion Level ed infine una checkBox per nascondere il grafo dei valori di Motion. I quattro pulsanti comandano rispettivamente:

    - *Start RTP*: l'inizializzazione di una sessione RTP per trasmettere in rete le immagini della webcam. Alla pressione compare una dialog box (figura 4.3) in cui inserire l'indirizzo IP e la porta di destinazione delle immagini.

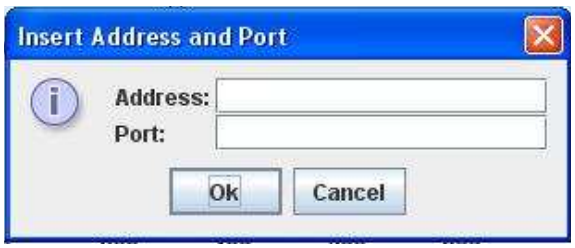

Figura 4.3: la finestra per l'inserimento dei parametri RTP

    - *Stop RTP*: termina la sessione RTP corrente.
    - *Send SMS*: invio manuale dell'SMS di attivazione della MIDlet. E' rimasto nell'interfaccia solamente per scopi didattici, in quanto l'invio degli SMS sarebbe altrimenti determinato dal superamento del livello di Motion.
    - *Screenshot*: permette il salvataggio su file del frame corrente. Alla pressione appare una dialogBox per scegliere il percorso del file di destinazione. Il formato è JPEG mentre la risoluzione è impostata a 320x240.

La barra superiore dei menù consente di raggiungere alternativamente le stesse funzionalità fornite dai pulsanti.

4.4 Caratteristiche

4.4.1 VideoServer

All'avvio, il programma genera un thread che si occuperà di ascoltare eventuali connessioni sulla porta definita nel file di configurazione (default: 1983). Il metodo run associato al thread ha la seguente struttura:

```
public void run() {
    try {
        ServerSocket serverSocket = new ServerSocket(port);

        While(true) {
            Socket socket = serverSocket.accept();
            ClientServer clientServer = new ClientServer(socket);
        }
    } catch (Exception e) {
    }
}
```

In questo modo, tramite socket TCP, il client per telefono cellulare può connettersi e visualizzare le immagini della webcam.

4.4.2 Invio immagini alla MIDlet

L'invio delle immagini alla MIDlet collegata è relativamente semplice ed avviene tramite *DataStreams*. Come si vedrà successivamente, in modo più approfondito nella sezione Protocolli, la MIDlet richiede l'immagine tramite l'invio di un messaggio (in questo caso è semplicemente una lettera – "i"). Alla ricezione di questo, il MotionDomus esegue una serie di azioni per preparare l'immagine all'invio vero e proprio:

- Viene effettuato uno screenshot del flusso video della WebCam
- Il frame così creato viene ridimensionato, tramite l'algoritmo di default, ad una risoluzione pari a 240x120 pixel
- L'immagine così generata viene memorizzata all'interno di un *BufferedImage*, secondo una struttura che prevede 8 bit per ogni componente, quindi in 3 bytes totali (il valore alpha viene ignorato)
- Infine viene codificata, tramite un *JPEGImageEncoder*, in un array di bytes.

L'operazione d'invio dell'immagine è composta, in realtà, da due operazioni distinte. La prima invia alla MIDlet la dimensione dell'array che contiene l'immagine, mentre la seconda comprende la trasmissione vera e propria del frame tramite un array di bytes. In questo modo la MIDlet in ricezione è in grado di conoscere esattamente, in anticipo, la dimensione dei dati che dovrà leggere. Le immagini, infatti, anche se possiedono la stessa risoluzione, occupano in realtà dimensioni diverse. La visualizzazione delle immagini sulla MIDlet è mostrata in figura 4.4.

Figura 4.4: le immagini della webcam visualizzate sulla MIDlet

4.4.3 Rilevazione Movimento

Questo metodo si basa sul confronto dei pixel del frame corrente con quelli del frame precedente. Vengono istanziati quindi due array di interi, entrambi con dimensione pari al prodotto dell'altezza per la larghezza (in pixel) del singolo frame.

```
private int[] previousPixels = new int[width * height];
private int[] latestPixels = new int[width * height];
```

Successivamente viene richiamata la classe `PixelGrabber` sull'ultimo frame. Questa classe prende un'immagine esistente e crea da essa un array di pixels.

```
PixelGrabber pixelGrabber = new PixelGrabber(image, 0, 0, width, height, latestPixels, 0, width);
```

Da questo array viene quindi estratto, tramite un ciclo for, ogni singolo pixel, successivamente scomposto nelle sue componenti. In questo modello, un pixel è di tipo *int* composto dai valori Alpha, Red, Green, Blue (0xAARRGGBB in esadecimale). Alpha rappresenta il grado di trasparenza del pixel (trasparenza totale = 0, opacità totale = 255). Un *int* è un intero a 32 bit, ogni colore è rappresentato da 8 bit (28 = 256); negli 8 bit superiori (da 32 a 24) abbiamo il valore Alpha, negli 8 successivi (24-16) abbiamo il valore Red, poi (16-8) il valore Green e negli ultimi 8, il valore Blue. L'operatore di scorrimento a destra (>>) sposta a destra di un numero specificato di volte tutti i bit appartenenti a un valore: val >> num sposta a destra i bit di val per il numero specificato da num.

```
motionLevel = 0;
    for(int ct = 0; ct < (width * height); ct++) {
        int latestPixel = latestPixels[ct];
       int latestAlpha = (latestPixel >> 24) & 0xff;
       int latestRed = (latestPixel >> 16) & 0xff;
       int latestGreen = (latestPixel >> 8) & 0xff;
       int latestBlue = (latestPixel) & 0xff;
       …
```

Con lo stesso procedimento si recuperano i valori dei singoli componenti dei pixel che compongono il frame precedente a quello che si sta attualmente considerando.
Successivamente i valori trovati vengono sottratti in modulo assoluto e confrontati con il livello soglia di rumore. Nel caso la differenza sia maggiore viene incrementato l'indice di Motion.

```
if ((Math.abs(previousRed - latestRed) > noiseThreshold)
     | (Math.abs(previousGreen - latestGreen) > noiseThreshold)
     | (Math.abs(previousBlue - latestBlue) > noiseThreshold))
       {
           motionLevel++;
       }
```

I valori dell'array `latestPixels` vengono copiati all'interno dell'array `previousPixels`, in modo da essere confrontati con il frame successivo. Infine il valore di Motion viene normalizzato secondo la formula

*motionLevel = (motionLevel * 1000)/noOfPixels*

dove `noOfPixels` rappresenta il numero di pixels dell'immagine (width * height). In questo modo il valore di motion rientra nel range prefissato 0 – 999.

4.4.3.1 Configurazione e comportamenti

Se configurato correttamente, il meccanismo di rilevazione del movimento si rivela molto potente. In caso contrario può assumere comportamenti anomali. In questa sezione ci proponiamo di descrivere i possibili comportamenti del programma nel caso della variazione, tramite le JSlider, dei parametri predefiniti.

La prima JSlider, definita *Maximum Motion*, ha funzione puramente estetica in quanto consente solamente, alla diminuzione dell'indicatore, di accrescere la visibilità dei valori rappresentati sul grafo, senza però modificare il meccanismo di Motion Detection. Questo significa, ad esempio, che impostando come Maximum Motion il valore 100 la scala di rappresentazione del grafo verrà adattata per mostrare i valore compresi tra 0 e 100, mentre i restanti verranno ignorati. Il programma però risponderà comunque nel caso venga registrato un valore di Motion superiore a quello indicato tramite la terza JSlider.

La seconda JSlider è molto importante in quanto determina il valore di soglia di rumore (noise), utilizzato poi per determinare la sensibilità del meccanismo di Motion Detection. Il rumore visivo, proprio come quello acustico, è un fattore spiacevole quanto purtroppo inevitabile. Il rumore elettronico presente nei sensori di acquisizione di fotocamere e telecamere determina la presenza, nel prodotto finale, del fenomeno conosciuto come "granularità". Il rumore visivo riscontrabile nelle immagini è, sostanzialmente, una variazione di luminosità e dell'informazione riguardante il colore. Questo fenomeno aumenta esponenzialmente nei casi di scarsa luminosità ad alti livelli del valore di esposizione (velocità ISO) oppure con una bassa velocità dell'otturatore. Nel nostro caso utilizzando una webcam, a meno di settaggi particolari implementati nei driver del dispositivo, non dobbiamo preoccuparci né dell'esposizione né dell'otturatore, ma dobbiamo tener conto che in un ambiente poco luminoso le immagini avranno la tendenza a presentare una granularità maggiore, aumentando la possibilità che venga erroneamente rilevato movimento.

Per questo motivo è consigliabile mantenere i valori di questa JSlider, almeno superiori al valore 10. L'eccessivo abbassamento del livello di Noise Threshold provoca, infatti, l'evidente sconvolgimento dei valori di Motion (come si può osservare dalla figura 4.5). Con un basso livello di tolleranza è quindi sufficiente una piccola quantità di rumore di fondo nell'immagine per determinare la rilevazione di movimento, nonostante un riscontro visivo dal punto di vista umano non ne riconosca.

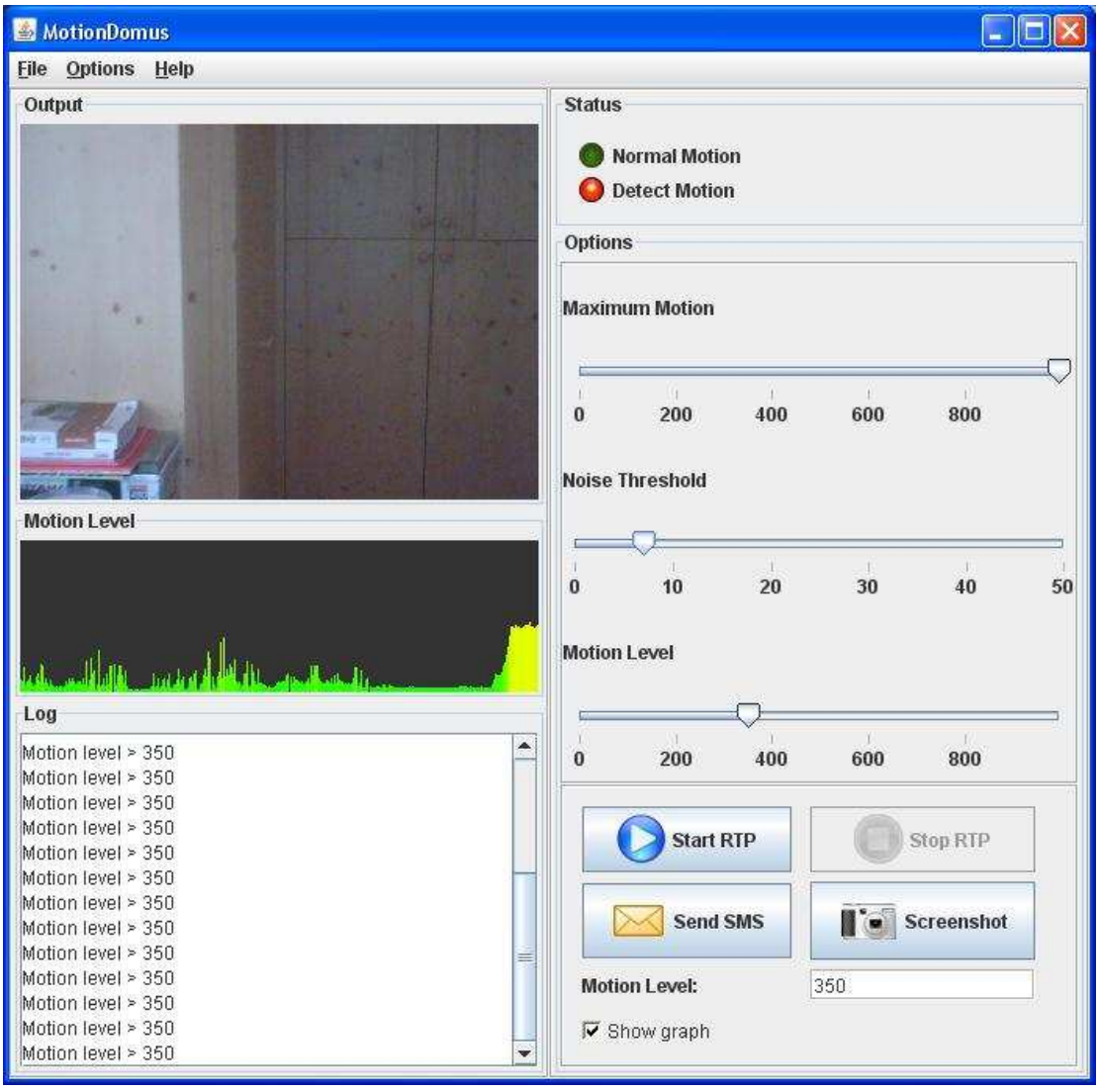

Figura 4.5: effetti indesiderati causati dalla presenza di rumore nelle immagini

La terza e ultima JSlider, chiamata *Motion Level*, imposta il livello soglia di Motion, oltre il quale scattano i provvedimenti. E' opportuno configurare questo valore a seconda delle esigenze dell'utente e dell'ambiente che si desidera monitorare.

4.4.4 Invio SMS

La rilevazione del movimento comporta l'invio di un SMS al numero configurato. L'SMS è di tipo 8 bit, pertanto non viene visualizzato come un normale SMS, ma consente invece di attivare, tramite il push registry, l'applicazione precedentemente registrata e di comunicarle alcune informazioni utili all'inizializzazione (nel nostro caso l'indirizzo IP del pc che ha inviato il messaggio).

Il funzionamento del meccanismo d'invio è molto semplice, in quanto si appoggia su una pagina in php sviluppata da Matteo Chini. Questa, a sua volta, utilizza i servizi forniti da Mobyt[xiii] che consentono la spedizione e la ricezione di messaggi SMS e MMS, attraverso la rete Internet.

I Vantaggi nell'utilizzo dei servizi SMS e MMS sono molteplici:

- Trasmissione dell'informazione in tempo reale
- Invio garantito
- Percentuale di lettura quasi pari al 100% e comunque ben superiore a qualunque altro mezzo di comunicazione
- Comunicazione non invadente
- Mezzo di comunicazione economico, semplice, rapido

Il sistema MotionDomus provvede, tramite la funzione `sendSmsMobyt(String telNumber)`, a creare una connessione di tipo http, in modo da invocare la pagina in php passando come parametri l'indirizzo IP del pc su cui il programma è attivo e il numero telefonico di destinazione del messaggio. Sarà poi questa pagina ad assumersi il compito di formattare correttamente il messaggio, per poi inoltrarlo correttamente al gateway di invio degli SMS fornito da Mobyt. Il messaggio così ricevuto sul cellulare è un istanza di *BinaryMessage* ed è in grado di attivare la rispettiva MIDlet registrata tramite il push registry.

4.4.5 RTP

Come già spiegato è possibile trasmettere a qualsiasi client collegato, tramite sessione RTP, le immagini catturate dalla webcam. Una volta inseriti l'indirizzo e la porta di destinazione verrà comunicato, tramite l'area di log (figura 4.6), l'avvio della sessione RTP. Il pulsante Stop RTP viene abilitato in modo da fermare la trasmissione in qualsiasi momento.

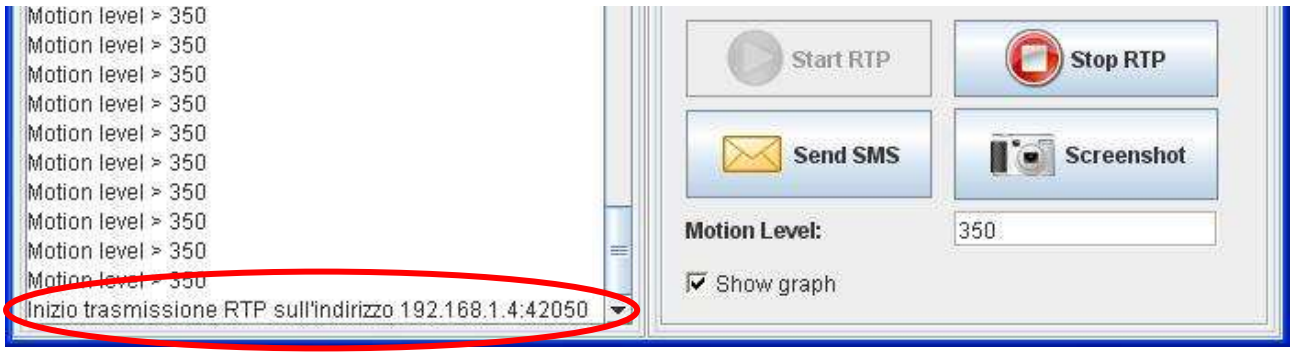

Figura 4.6: notifica della trasmissione RTP

Una volta avviata la trasmissione è possibile collegarsi con un qualsiasi client che supporta tali sessioni per visualizzare immediatamente le immagini. I requisiti sono ovviamente l'IP sorgente e la porta da utilizzare. Le prove sono state fatte utilizzando JMStudio (figura 4.7) integrato con il JMF.

Figura 4.7: visualizzazione delle immagini della webcam su un PC diverso dalla sorgente

Per permettere al software questa caratteristica, il punto di partenza è stata la classe di esempio `AVTransmit2` presente sul sito web del JMF[xiv]. Questa classe è stata opportunamente adattata in modo da richiedere all'utente solamente le informazioni strettamente necessarie come, appunto, l'indirizzo IP e la porta base del PC che riceverà la trasmissione. Questa può essere una qualsiasi porta che non sia già utilizzata per alcun servizio sul proprio computer.

Nonostante la classe supporti la trasmissione di tracce multimediali multiple (ad esempio video ed audio), nel nostro caso viene creata solamente una singola sessione RTP per l'invio del video. Vengono utilizzate, per flessibilità, le RTP Session API invece delle DataSink API. Viene inoltre prestata attenzione speciale alla risoluzione per assicurarsi che le dimensioni del video siano compatibili per la trasmissione RTP. Viene applicato, se necessario, un ridimensionamento in real-time.

4.5 Tracciabilità IP

Il client per telefono cellulare per collegarsi direttamente al MotionDomus necessita di due parametri fondamentali: l'indirizzo IP e la porta del PC su cui risiede. Il secondo valore è statico e può essere scelto utilizzando il file di configurazione, cambiando il rispettivo valore. L'indirizzo IP invece, nella maggior parte delle connessioni casalinghe, è un valore dinamico assegnato dal proprio provider (ISP). Pertanto questo valore potenzialmente cambia ad ogni collegamento. Sorge quindi il problema di rendere accessibile all'esterno un servizio: ad ogni connessione è necessario comunicare al mondo l'indirizzo IP, sempre diverso.

Sono nati quindi numerosi servizi, gratuiti e a pagamento, per venire incontro alle esigenze dell'utente. Si è pensato quindi di provare a costruirne uno analogo, in modo che consentisse all'utente di accedere in modo semplice e veloce all'indirizzo IP del computer che espone il servizio.

E' opportuno ricordarsi di configurare il firewall personale oppure il Windows Firewall, se presente, in modo da rendere disponibile all'esterno il servizio. Bisogna inoltre prestare la dovuta

attenzione alle eventuali falle di sicurezza, mentre è importante disattivare il tutto se non lo si utilizza.

Come già accennato il sistema si compone delle seguenti parti:

4.5.1 Database

Il database utilizzato è un semplice DB MySQL chiamato *users*. E' formato da 3 colonne, chiamate rispettivamente: username, address e password. Tutti e tre i campi sono del tipo *varchar* con lunghezza massima 15 caratteri. Come chiave primaria viene utilizzato il valore username, inoltre l'unico valore null presente può essere il campo address.

Un esempio del Database popolato può essere il seguente:

users.users: 5 records total

username	address	password
gibi	192.168.1.2	gibipwd
paolo	192.168.1.4	paolopwd
marco	87.0.109.242	marcopwd
anna	87.2.106.183	annapwd
mauri	87.2.106.183	mauripwd

Figura 4.8: esempio di database

4.5.2 RMIinterface – interfaccia remota

L'interfaccia remota contiene al suo interno la definizione dei metodi che l'oggetto sul server espone ai client. Un'interfaccia remota deve sottostare ai seguenti vincoli:

- Derivare dalla interfaccia `java.rmi.Remote`
- Definire ogni metodo con la clausola: `throws java.rmi.RemoteException`

La nostra interfaccia ha la seguente struttura:

```
public interface RMImethods extends Remote {
    boolean existsUser(String username) throws RemoteException;
    boolean existsUserPass(String username, String password) throws RemoteException;
    String addressRetrieve(String username, String password) throws RemoteException;
    boolean refreshAddress(String username, String address, String password) throws RemoteException;
    boolean insertUser(String username, String address, String password) throws RemoteException;
    boolean removeUser(String username, String password) throws RemoteException;
}
```

Il significato di ogni metodo è intuitivo.

4.5.3 AddressIndex - applicazione Server

Per l'interfaccia remota definita (nel nostro semplice caso, `RMImethods`) è necessario creare una classe sul server che contenga l'implementazione dell'oggetto remoto vero e proprio. Tale classe deve:

- Derivare dalla classe `java.rmi.server.UnicastRemoteObject`

- Implementare tutti i metodi definiti nell'interfaccia remota (regola comunque sempre valida in Java)
- Tutti i costruttori definiti nella classe devono "lanciare" un'eccezione del tipo `java.rmi.RemoteException`. Si noti che se si decidesse di utilizzare soltanto il costruttore di default, sarà comunque necessario scriverne il codice per gestire proprio la throw dell'eccezione `RemoteException`.

L'implementazione dell'oggetto remoto è stata costruita nella classe `RMImethodsImpl` che richiama, a sua volta, i metodi definiti all'interno di una classe chiamata `UsersHandler`. Quest'ultima rappresenta la vera e propria connessione con il database.

Importante notare il meccanismo di binding dell'oggetto server costruito, nel nostro caso, direttamente all'interno del `Main`:

```
public class Main {
    public static void main(String[] args) {
        try {
            RMImethodsImpl server = new RMImethodsImpl();
            Registry registry = LocateRegistry.getRegistry();
            Registry.rebind("RMIserver", server);
        catch (RemoteException e) {

        }
    }
}
```

Molto importante è l'istruzione evidenziata con la quale, infatti, viene ricercato su localhost il registro RMI e successivamente viene effettuato il bind dell'oggetto server (di tipo `RMImethodsImpl`) con il nome "`RMIserver`".

4.5.4 AddressReveal – applicazione Client

Il compito dell'applicazione client è ricercare l'applicazione server che espone gli oggetti remoti messi a disposizione attraverso RMI ed invocarne, quindi, i metodi opportuni.

Al primo avvio assoluto del programma compare una finestra di dialogo (figura 4.9) in cui inserire i valori userID e password.

Figura 4.9: dialog box in cui inserire i propri valori di accesso

Questi verranno controllati con quelli presenti nel database: se la coppia è già presente allora il programma aggiornerà l'indirizzo corrispondente, se invece non esiste viene effettuato un controllo sul valore username, in modo che sia univoco. Nel caso sia già esistente all'interno del database, viene immediatamente comunicato all'utente di scegliere un valore nuovo. Se invece non ci sono corrispondenze la coppia viene inserita come nuova tupla all'interno del database, assieme all'indirizzo corrispondente, come illustrato in figura 4.10.

users.users: 6 records total

	username	address	password
▶	gibi	192.168.1.2	gibipwd
	paolo	192.168.1.4	paolopwd
	giuseppe	79.11.109.210	giuseppepwd
	marco	87.0.109.242	marcopwd
	anna	87.2.106.183	annapwd
	mauri	87.2.106.183	mauripwd

Figura 4.10: inserimento di una nuova tupla nel database

Ai successivi avvii dell'applicazione i valori inseriti verranno recuperati dal file `RMIclient.properties`, in modo da non doverli digitare nuovamente nei rispettivi campi.

L'interfaccia del programma è molto semplice ed intuitiva (figura 4.11):

Figura 4.11: interfaccia grafica del programma client

E' possibile, tramite script di batch o l'esecuzione automatica, avviare il programma all'avvio di ogni sessione di lavoro con il computer, in modo da aggiornare l'indirizzo non appena il computer viene acceso. E' inoltre possibile utilizzare due comandi manuali per aggiornare l'indirizzo localmente e sul database.

In alto le due textBox informano rispettivamente dell'indirizzo privato e di quello pubblico, a destra invece si trovano i due pulsanti Refresh e Upload. In basso infine si trova l'area dei messaggi che informa delle azioni compiute dal programma.

Degni di nota, nel caso del Client, sono le istruzioni

```
public class InitRMI {
    public rmiinterface.RMImethods init() {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
            Registry registry = LocateRegistry.getRegistry(host)
            Rmiinterface.RMImethods serverMethods = (rmiinterface.RMImethods)
            registry.lookup("RMIserver");

            return serverMethods;
        catch (RemoteException e) {

        }
    }
}
```

L'istruzione in blu si riferisce al *SecurityManager*. Nello sviluppo di applicazioni distribuite con la tecnologia RMI è necessario regolare le azioni eseguite dalle classi remote. Altrimenti si potrebbe inavvertitamente permettere l'esecuzione di codice non sicuro che potrebbe accedere a risorse di sistema private. Questo comportamento può essere evitato installando un security manager. Di default un programma RMI non ne ha uno installato, quindi non sono impostate restrizioni sugli oggetti remoti. Il package `java.rmi` fornisce una basilare implementazione del security manager, esattamente quella utilizzata nel software.
E' necessario inoltre specificare, a runtime, un file contentente le security policy, definendo un valore per la proprietà `java.security.policy`:

```
java -Djava.security.policy=policyfilename
```

Java cerca di default un file policy in *java.home/lib/security/java.policy*, dove *java.home* è la directory in cui la JDK o la JRE è installata. Se non viene specificato, la JVM cerca un eventuale file di policy, definito dall'utente, nella cartella *user.home/.java.policy*, dove *user.home* è la directory home dell'utente.

La sintassi del policy file è descritta nel file *docs/guide/security/PolicyFiles.html* incluso nella documentazione. Nel nostro caso si è voluto garantire il permesso a tutti, tramite le righe:

```
grant {
  permission java.security.AllPermission;
};
```

Le istruzioni in rosso, invece, si occupano di effettuare la ricerca del registro sul server e quindi il lookup dell'oggetto remotizzato denominato `RMIserver`.

Unica istanza:

Per prevenire che venissero eseguite più istanze dell'applicazione, anche inavvertitamente, è stato implementato un sistema semplice ed efficace istanziando una `serverSocket` su una porta effimera (nel nostro caso: 15486). Con questo sistema successive istanze dell'applicazione, trovando la porta già occupata, non proseguirebbero la loro esecuzione.

System tray:

Il programma è pensato per funzionare in background infatti, riducendo a icona il programma (o chiudendolo tramite la classica X), continua la sua esecuzione nella System Tray (figura 4.12), ovvero l'area che ospita l'orologio e le icone dei processi attivi.

Figura 4.12: l’icona del client che comunica l’esecuzione in background

Infatti con il rilascio di Java 6 è possibile utilizzare le nuove funzionalità per l’interazione con la traybar di sistema. L'implementazione fornita dal JDK6 gestisce, in modo del tutto indipendente dalla piattaforma in uso, i principali window-manager, dalla shell di Windows ai più comuni ambienti desktop per GNU/Linux, come KDE e Gnome.

La classe Java che si occupa di interfacciare l'applicazione alla traybar del sistema si chiama `SystemTray`, contenuta nel pacchetto `java.awt`.
Questa classe non può essere istanziata liberamente, infatti è necessario verificare se ci si trova in un ambiente in cui la traybar è supportata utilizzando il metodo statico:

```
boolean isSupported()
```

Una chiamata a questo metodo dovrebbe essere la prima istruzione all'interno del processo d'implementazione. Se il valore di ritorno è false, è opportuno ripiegare su un'altra soluzione. Il metodo descritto garantisce che la traybar sia supportata e comporta, inoltre, una chiamata a `checkPermission()` sul *SecurityManager* di sistema, passando come parametro `“accessSystemTray”`, controllando il permesso d'accesso.

Il metodo che permette di recuperare la traybar di sistema e di restituirla è

```
SystemTray getSystemTray();
```

A questo punto è possibile aggiungere alla traybar oggetti di tipo `TrayIcon`. Questi rappresentano le icone all'interno della tray-area di sistema, ovvero della zona in cui si trovano le icone di stato o di notifica che permettono di accedere ai processi che girano in background. In effetti, tale zona cambia nome e comportamento in base alla piattaforma. Sui sistemi Windows è nota come Taskbar Status Area, per Gnome si chiama Notification Area, KDE sceglie invece un più banale System Tray. Un'icona all'interno della tray-area è un componente AWT completo, in grado di legarsi ai Listeners di eventi di tipo ActionEvent e MouseEvent e di ricevere, quindi, input da parte dell'utente.

All’icona quindi sono associati gli stessi Listeners dei pulsanti, consentendo di eseguire le azioni di *Refresh* e *Upload* direttamente dalla traybar (figura 4.13).

Figura 4.13: operazioni accessibili dalla traybar

Il comando *Exit* è l’unico in grado di chiudere l’applicazione. Viene richiesta la conferma tramite un’apposita dialog. La chiusura comporta l’eliminazione dal database della tupla associata alla coppia *username/password* utilizzata per l’accesso.

4.5.5 AddressServlet

Si tratta di una semplice servlet che, definiti i parametri *username* e *password*, restituisce l'indirizzo IP memorizzato nel database (figura 4.14). Si basa sulla stessa classe `UsersHandler` definita nell'`AddressIndex`, ristretta però solamente al metodo `findAddress` definito come segue:

```
public String findAddress(String username, String password) {
    try {
        Statement stmt = connection.createStatement();

        ResultSet rs = stmt.executeQuery("SELECT * FROM users WHERE username='"+ username +"' AND
        password='"+ password +"'");

        if (rs.next()) {
            return ((String) rs.getObject("address"));
        }
        rs.close();
        stmt.close();
    } ...
}
```

La servlet viene invocata con il metodo doGet, ricevendo i parametri direttamente dall'URL:

```
public void doGet(HttpServletRequest request, HttpServletResponse response) {
    user = request.getParameter("username");
    pass = request.getParameter("password");

    address = usershandler.findAddress(user, pass);
    ...
    response.setContentType("text/plain");
    PrintWriter out = response.getWriter();
    out.println(address);
    out.close();
}
```

E' stata scelta una servlet per ovviare a questo compito, innanzitutto per la facilità di implementazione ed inoltre per la semplicità con cui una MIDlet può invocarla. L'`HttpConnection` della piattaforma J2ME è la tipologia di connessione più diffusa e meglio supportata.

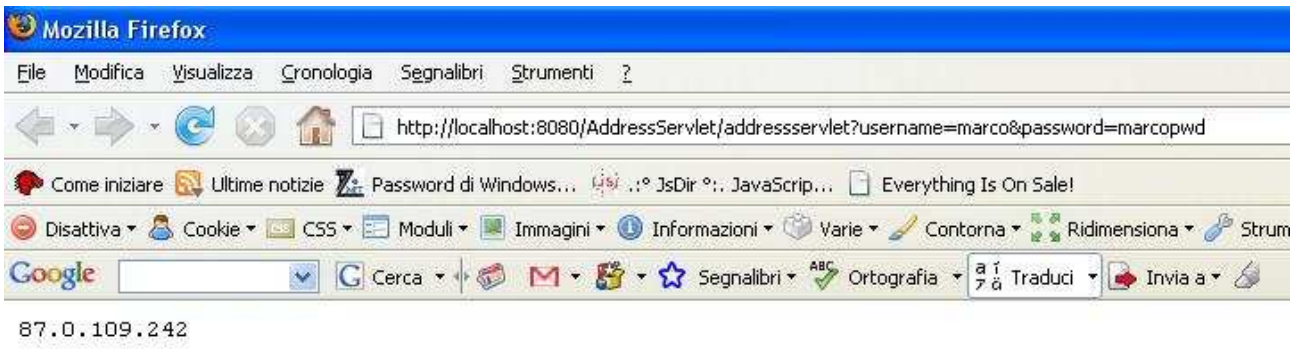

Figura 4.14: funzionamento pratico della servlet

4.5.6 Client Cellulare

Il client su telefono cellulare all'atto di apertura richiede, tramite i due box visibili nella figura 4.15, i due parametri necessari a richiamare la servlet tramite una sessione HTTP, in modo da risalire all'indirizzo. Nel caso i valori non trovassero corrispondenza al posto dell'indirizzo viene salvata la stringa "Address Unknown".

Figura 4.15: i box di testo per l'inserimento dei valori di autenticazione

Il meccanismo di recupero dell'indirizzo è contenuto in una classe chiamata `ServletInvocation`, richiamata non appena l'utente preme il pulsante Login. Questa classe implementa l'interfaccia Runnable in modo da essere eseguita in un thread separato. Nella programmazione di MIDlet, infatti, è buona norma istanziare nuovi thread per le operazioni potenzialmente pericolose, come appunto le connessioni di rete, in modo da evitare possibili blocchi del processo principale che porterebbero forzare l'utente a dover riavviare l'apparecchio.

L'indirizzo recuperato tramite il metodo GET viene così salvato, tramite il meccanismo RMS, all'interno della memoria del cellulare per essere poi richiamato nel processo di connessione.

Il funzionamento dettagliato delle altre caratteristiche del Client per il telefono cellulare verranno descritte ampiamente nell'analisi della seconda parte del progetto.

4.6 Conclusioni di capitolo

In questo capitolo è stata analizzata la prima parte del sistema, la quale ha preso il nome di MotionDomus. In particolare è stato mostrato il funzionamento pratico corredando la trattazione con immagini esplicative e con brevi e rilevanti sezioni di codice, al fine di mostrare l'implementazione. Inoltre materia di questo capitolo è stata anche l'analisi del servizio di tracciabilità dell'indirizzo IP.

Capitolo 5

ControlDomus

Con il proseguimento del lavoro si è aperta la possibilità di usufruire delle tecnologie del laboratorio domotico in via della Cascata a Povo. Si è pensato a come il sistema MotionDomus potesse integrarsi all'interno di tale scenario, in modo da permettere sviluppi più ampi da parte di entrambe le applicazioni.
L'idea di partenza era, quindi, quella di permettere l'iterazione tra il telefono cellulare e le tecnologie preesistenti all'interno del laboratorio. In particolare si è potuto usufruire di due telecamere IP Axis 212 PTZ, in grado di controllare tutto l'ambiente interno, e dell'EventEngine di Enrico Nicoletti che permette di attuare un'astrazione tra lo strato di "sensing" e quello di "intelligence". Questo ha facilitato il processo di sviluppo, in quanto è stato possibile operare su eventi simulati, senza preoccuparsi del meccanismo tramite cui tali eventi venivano generati.

Il lavoro si è quindi svolto in due direzioni:

- Progetto DoCo (Domus Control): essenzialmente un gateway che permettesse la comunicazione tra il telefono cellulare e le tecnologie del laboratorio domotico.
- Progetto MoDoCo (Mobile Domus Control): la vera e propria MIDlet di controllo, l'interfaccia finale per l'utente.

5.1 Progetto DoCo

Il programma è stato scritto da zero, sfruttando e adattando però alcune tecnologie già utilizzate all'interno del MotionDomus. In particolare si è potuto riutilizzare (con piccole modifiche) il sistema client-server tramite cui vengono inviate al telefono cellulare le immagini delle telecamere e il meccanismo di Motion Detection.
Invece per interfacciarsi con le telecamere non è stato possibile usufruire del JMF, in quanto il player non supporta di default il codec MJPEG, ma si è dovuto sviluppare un decodificatore (molto semplice in verità) per rimuovere l'incapsulamento delle singole immagini JPEG.
Oltre al meccanismo, basato su socket TCP, di invio delle immagini verso un telefono cellulare, si sono aggiunti altri 3 sistemi di comunicazione (sempre basati su socket):

- **Control Connection**: una connessione in grado di ricevere comandi predefiniti da un telefono cellulare e di inoltrarli all'EventEngine
- **Engine Connection**: inizializzata all'avvio del software, permette la connessione, sulla porta 1111, con l'EventEngine.
- **Notification Connection**: una connessione in attesa delle eventuali notifiche delle telecamere.

5.1.1 Configurazione

Come nel progetto MotionDomus, al primo avvio dell'applicazione viene creato un file di configurazione nel linguaggio XML. Questo file, chiamato come nell'altro caso `config.xml`, contiene i parametri di configurazione modificabili. Questi parametri sono qui elencati. Alcuni sono differenti rispetto al caso del MotionDomus e pertanto verranno ulteriormente presi in dettaglio:

- **commands**: il percorso del file XML contenente i comandi
- **eventEngineNotifications**: il percorso del file XML contenente le notifiche generate dall'EventEngine e gestibili dal software. Contiene inoltre i comandi associati a tali notifiche.
- **cameraNotifications**: il percorso del file XML contenente le notifiche generate dalle telecamere e gestibili dal software, assieme ai comandi associati.
- **mapCommands**: il percorso del file XML che contiene la corrispondenza tra i comandi che sono ricevuti dal telefono cellulare e quelli inviati all'EventEngine.
- **videoPort**: il numero di porta a cui il cellulare si collega per ricevere le immagini. (default: 1983)
- **controlPort**: il numero di porta a cui il cellulare si collega per inviare i comandi (default: 1985)
- **enginePort**: il numero di porta da utilizzare per collegarsi all'EventEngine (default: 1111)
- **notificationPort**: il numero di porta utilizzato dalle telecamere per inviare le notifiche (default: 1987)
- **telNumber**: il numero telefonico del dispositivo su cui è installata la MIDlet da attivare
- **ipCamera1**: l'indirizzo IP della telecamera 1
- **ipCamera2**: l'indirizzo IP della telecamera 2
- **messaggio**: informazione per il server d'invio SMS (default: Attenzione)
- **utente**: nome utente per il servizio d'invio SMS
- **password**: password per il servizio d'invio SMS
- **serverName**: nome del server d'invio degli SMS (default: Mike Server)

Tale file viene richiamato ad ogni avvio del programma. Nel caso non fosse presente viene ricreato con la configurazione di default.

Prima di passare all'analisi dell'interfaccia è opportuno soffermarci sui dati di configurazione del programma. Questi dati sono contenuti all'interno dei quattro file XML identificati tramite i parametri corrispondenti nel file `config.xml`. Questi quattro file sono rispettivamente `CommandsSequence.xml`, `MappedCommands.xml`, `EventEngineNotifications.xml` e `CameraNotifications.xml`.

5.1.1.1 Descrizione dei comandi

Il file `CommandsSequence.xml` definisce i comandi che possono essere utilizzati in risposta agli eventi che raggiungono il software, indipendentemente dal canale di comunicazione (notifiche telecamere, notifiche EventEngine…). Il file ha la seguente struttura:

```
<?xml version="1.0"?>
<CommandsSequence>
    <commands>
        <command>
            <name>SendSMSCucina</name>
            <content>Entrato in ZonaPericolosaCucina</content>
            <numbers>
                <number>340*******</number>
                <number>339*******</number>
            </numbers>
        </command>
        .
        .
        .
</CommandsSequence>
```

Il tag <*name*> identifica il nome del comando. Il tag <*content*> invece definisce il contenuto dell'SMS che verrà inviato, è buona norma mantenerlo il più sintetico possibile (per rientrare nei

160 caratteri disponibili in un sms). Infine il tag <*numbers*> definisce una sequenza di numeri di cellulare a cui dovrà essere inviato l'SMS corrispondente. L'unico formato accettato è quello classico, ovvero il semplice numero telefonico, senza il prefisso +39 per indicare il paese.

5.1.1.2 Descrizione dei comandi mappati

Il file `MappedCommands.xml` contiene la mappatura tra i comandi che il telefono cellulare può inviare al ControlDomus e quelli che lo stesso ControlDomus deve inviare all'EventEngine per ottenere lo scopo desiderato. La figura 5.1 ne spiega intuitivamente il funzionamento:

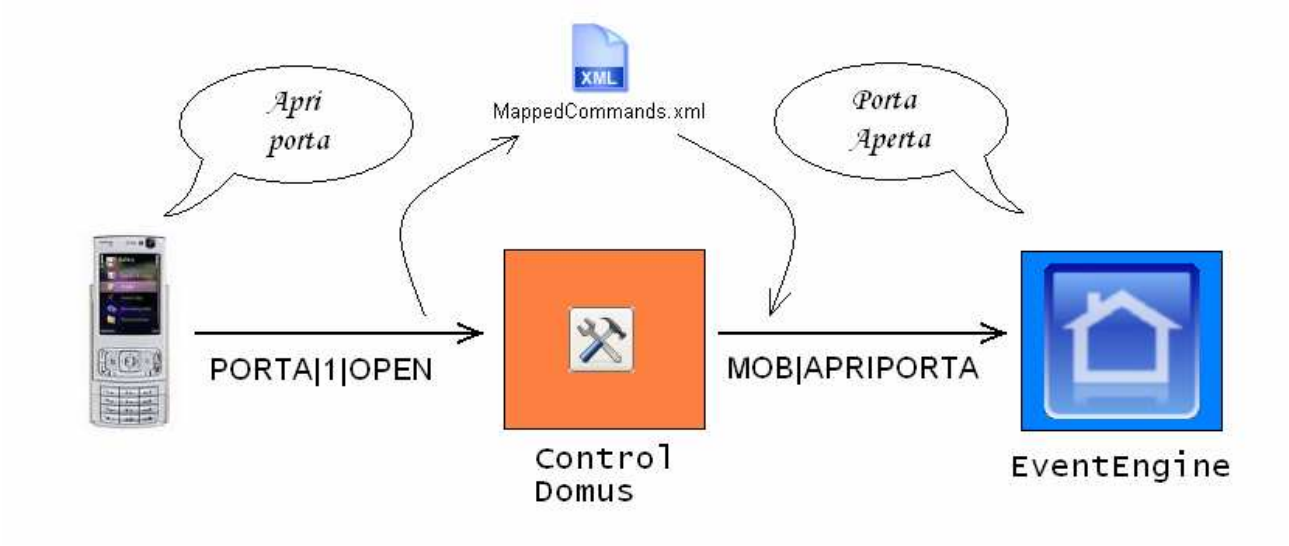

Figura 5.1: schematizzazione dei comandi mappati

La struttura del file XML è immediata:

```
<?xml version="1.0"?>
<MappedCommands>
    <control_commands>
        <element>
            <remote_command>PORTA|1|OPEN</remote_command>
            <event_command>MOB|APRIPORTA</event_command>
        </element>
        .
        .
        .
</MappedCommands>
```

Gli elementi contenuti nel file sono formati da una coppia: il tag <*remote_command*> definisce il comando utilizzato dal telefono cellulare e deve essere identico alle definizioni contenute sul dispositivo stesso, ma che comunque esamineremo in seguito, ed infine il tag <*event_command*> definisce il comando per l'EventEngine. Questo, a sua volta, deve essere identico con il trigger specifico definito nel file `etc.xml` di configurazione dell'EventEngine (consultare la documentazione relativa).

5.1.1.3 Descrizione delle notifiche delle telecamere

Come già accennato prima, le telecamere del laboratorio domotico possono inviare notifiche tramite socket TCP nel caso fossero configurate per rispondere a livelli di Motion superiori alla guardia, oppure a livelli audio elevati. Il file `CameraNotifications.xml` contiene appunto queste notifiche, in modo da rispondere nel modo più opportuno all'eventualità registrata dalle telecamere.

La struttura del file è la seguente:

```
<?xml version=”1.0”?>
<CameraNotifications>
    <notifications>
        <notification>
            <name>Motion</name>
            <description>MotionDetection</description>
            <sender>192.168.188.226</sender>
            <commands>
                <command>SendSMSMotionCam1</command>
            </commands>
        </notification>
        .
        .
        .
</CameraNotifications>
```

La singola notifica è identificata dal tag <*notification*>, il nome e la descrizione rispettivamente dai tag <*name*> e <*description*>. Questi valori dovranno essere identici al formato utilizzato dalla telecamera per trasmettere le notifiche (%e|%E – evento|descrizione). Il tag <*sender*> è invece utilizzato per identificare l’indirizzo IP della telecamera che ha trasmesso la notifica. Infine con il tag <*commands*> si indica la sequenza dei nomi di comandi (definiti nel file *CommandsSequence.xml*) che verranno utilizzati per rispondere a quel determinato evento. In particolare il comando presentato nell’esempio ha questa struttura:

```
    <command>
        <name>SendSMSMotionCam1</name>
        <content>La telecamera 1 ha registrato un eccessivo livello di motion</content>
        <numbers>
            <number>340*******</number>
        </numbers>
    </command>
```

5.1.1.4 Descrizione delle notifiche dell’EventEngine

Prima di descrivere i messaggi che l’EventEngine può comunicare, è opportuno focalizzare l’attenzione su un’altra caratteristica di rilevante interesse che lo stesso EventEngine dispone. Si tratta della disponibilità a configurare nell’ambiente (tramite XML) delle zone sensibili di particolare rilevanza. E’ possibile quindi definire, ad esempio, il perimetro circostante i fornelli della cucina in modo da reagire nel caso venisse registrata la presenza di utenza all’interno. In particolare, allo stato attuale delle cose, si può registrare l’atto di ingresso e di uscita da una zona sensibile, operare una tipologia di reazione interna all’EventEngine ed, eventualmente, trasmettere queste informazioni al ControlDomus o ai dispositivi collegati.
Nelle prove effettuate, oltre alle zone sensibili definite nell’interesse dell’EventEngine, sono state create quattro nuove zone per dimostrare le potenzialità del sistema completo. Queste vengono mostrate nell’immagine 5.2.

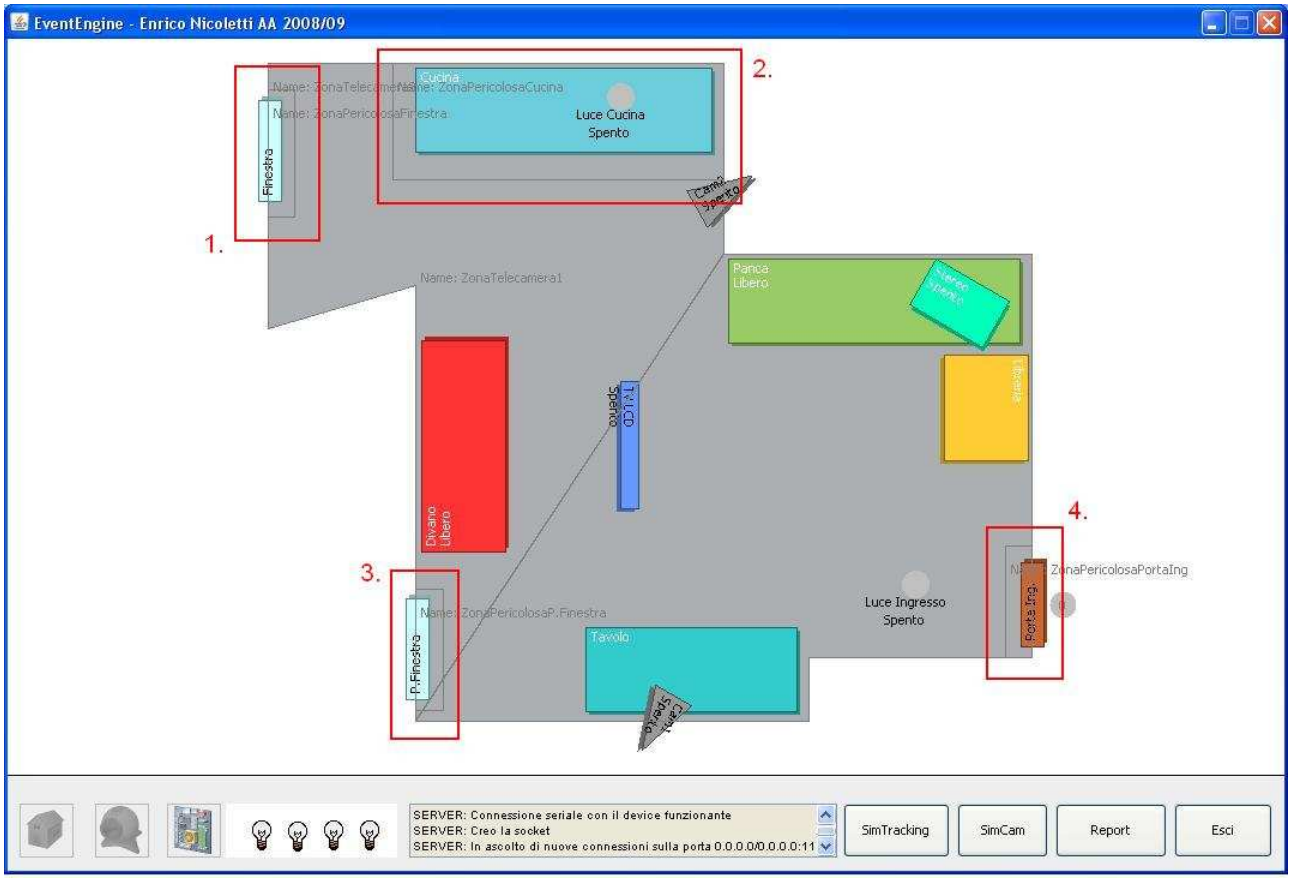

Figura 5.2: zone sensibili d'interesse per il DomusControl definite nell'EventEngine

Le zone definiscono un perimetro di guardia attorno ad elementi sensibili come la finestra (1), la cucina (2), la porta del terrazzo (3) ed infine la porta d'ingresso (4). Il passo successivo è stato quindi quello di associare (nel file `etc.xml` dell'EventEngine) ad ogni zona un evento e quindi un trigger corrispondente. In questo modo, non appena un mote entra nell'area sensibile, l'informazione viene comunicata ai client collegati.
La convenzione utilizzata per comunicare queste informazioni è data dal formato:

trigger_content|attribute|n_mote

trigger_content è definito dal tag <*content*> corrispondente al trigger definito dall'evento. Nel nostro caso può assumere ovviamente quattro valori: ZonaPericolosaCucina, ZonaPericolosaFinestra, ZonaPericolosaP.Finestra e ZonaPericolosaPorta. Il valore attribute indica se l'azione compiuta dal mote è di tipo "enter" oppure "exit". Per semplicità abbiamo impostato solo azioni di tipo "enter". L'ultima sezione riguarda il numero associato al mote. Nell'EventEngine è possibile gestire, in modo simulato, fino a 10 mote. L'ultimo comando consente quindi di personalizzare le notifiche in base al numero del mote.

Se proviamo a pensare ci accorgiamo dell'utilità di questo valore: il mote indossato da un adulto non deve attivare nessuna notifica, nel caso entrasse nella zona sensibile definita, per esempio, attorno alla finestra. Il mote indossato da un bambino, invece, dovrebbe notificare nel modo appropriato se si è avvicinato troppo ad un elemento sensibile.

Con queste premesse possiamo analizzare il file `EventEngineNotifications.xml`. La sua struttura è simile a quelle già osservate:

```
<?xml version=”1.0”?>
<EventEngineNotifications>
    <event_engine>
        <element>
            <name>EntratoInZonaPericolosaCucina</name>
            <content>ZonaPericolosaCucina|ENTER|0</content>
            <commands>
                <command>SendSMSCucina</command>
            </commands>
        </element>
        .
        .
        .
</EventEngineNotifications>
```

Il tag <*name*> identifica il nome della notifica, il tag <*content*> invece definisce il comando completo ricevuto. Per semplicità si è lavorato utilizzando un singolo mote (il numero 0), ma l'estensione a più mote è semplice ed immediata. La sezione <*commands*> definisce i comandi associati all'evento. Come negli altri casi, questi comandi devono trovare la giusta corrispondenza nel file `CommandsSequence.xml`.

E' importante sottolineare come una errata formattazione dei documenti XML impedisce l'avvio del programma. Viene comunque comunicato qual è il file che contiene l'errore di formattazione.

5.1.2 Funzionamento

Per lanciare il programma è sufficiente eseguire il comando `java –jar DomusControl.jar` dal terminale. Al primo avvio del programma verrà creato, nella stessa cartella in cui è presente l'eseguibile .jar, il file `config.xml`. Inoltre il programma ricercherà i file che descrivono le notifiche e i comandi all'interno di quella stessa cartella. Nel caso non fossero presenti il programma interrompe la sua esecuzione comunicando che uno tra i 4 file di configurazione non è formattato correttamente. E' possibile, a questo punto, copiare i file `CommandsSequence.xml`, `EventEngineNotifications.xml`, `MappedCommands.xml` e `CameraNotifications.xml` all'interno della directory del programma o, in alternativa, specificare all'interno del file config.xml i percorsi relativi a tali files, avendo cura di non inserire percorsi che contengono spazi all'interno del namespace.
All'interno della stessa directory verrà creata una nuova cartella, chiamata *Log*, che conterrà i file di log relativi al software. Se non è possibile crearla all'interno di tale percorso il software prova a crearla nella root directory di `c:`.

Se tutte le operazioni sono andate a buon fine il programma caricherà l'interfaccia grafica.

5.1.3 Interfaccia grafica

L'interfaccia grafica (figura 5.3) è sostanzialmente divisa in 4 parti:

- A sinistra troviamo i pannelli che mostrano le immagini delle due telecamere, identificati rispettivamente da Camera 1 e Camera 2
- Al centro troviamo due pannelli informativi riguardanti lo stato del programma: nella parte superiore c'è una textArea, chiamata *Log Main*, che visualizza messaggi importanti per l'utente come, ad esempio, quale telecamera ha registrato un alto valore di Motion. Nella parte inferiore invece troviamo delle icone informative, il cui significato verrà chiarito successivamente, nel corso dell'analisi.
- Nella parte destra troviamo due pannelli identici che rappresentano opzioni regolabili per il Motion Detection delle due telecamere. Il meccanismo di rilevazione del

movimento, infatti, è indipendente per ogni telecamera. Questo pannello è formato da: due led, già visti nel caso del MotionDomus, che indicano lo stato di Motion, una checkBox per attivare/disattivare la rilevazione del movimento, due textBox che comunicano rispettivamente il valore attuale di motion (*Current Motion Level*) e il valore di soglia oltre il cui vengono presi provvedimenti (*Threshold Motion Level*). Troviamo, inoltre, un pulsante *Option* che permette di regolare, tramite due slider, la soglia del rumore e il valore soglia di Motion (figura 5.4).

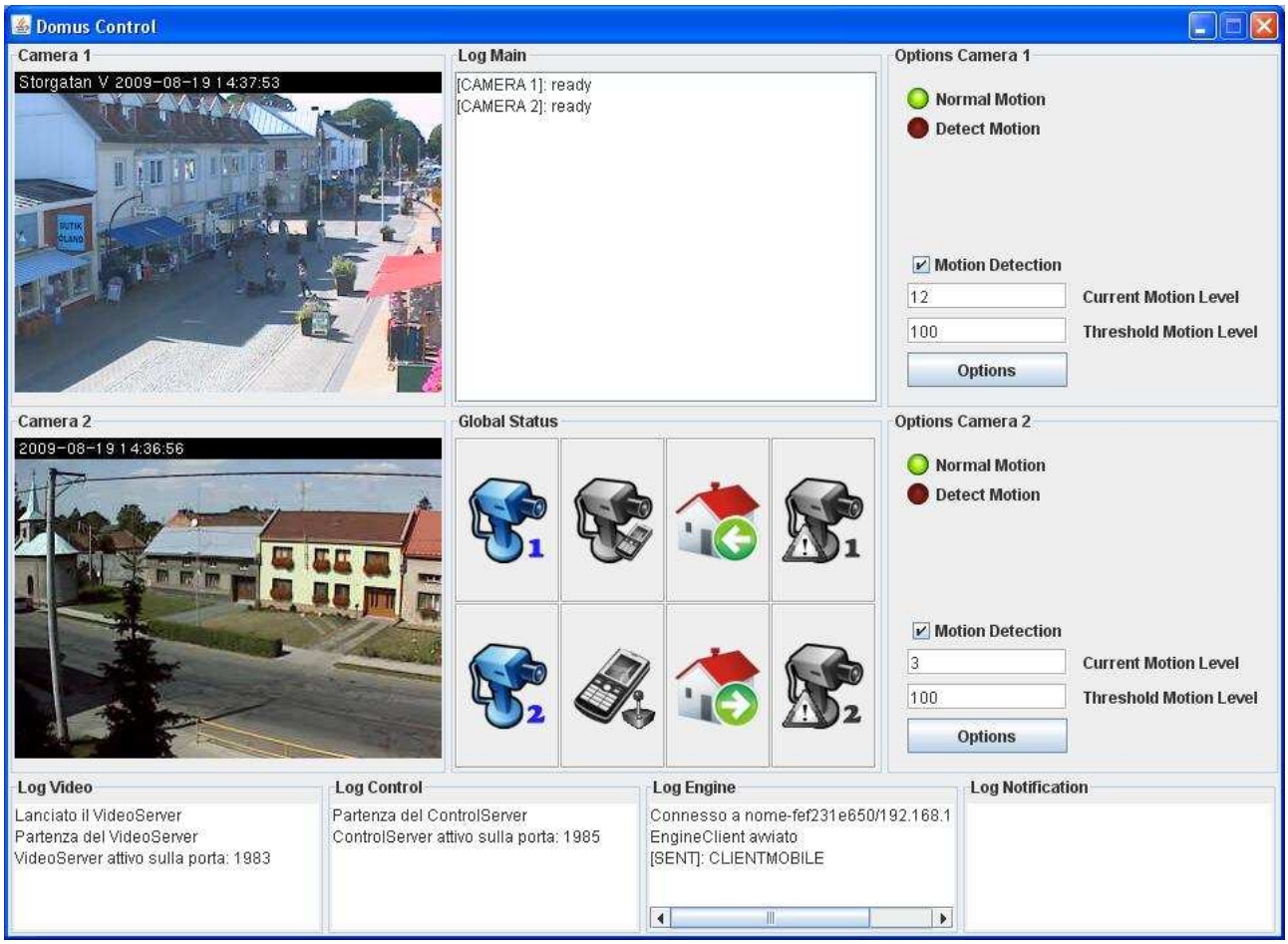

Figura 5.3: l'interfaccia grafica del Domus Control

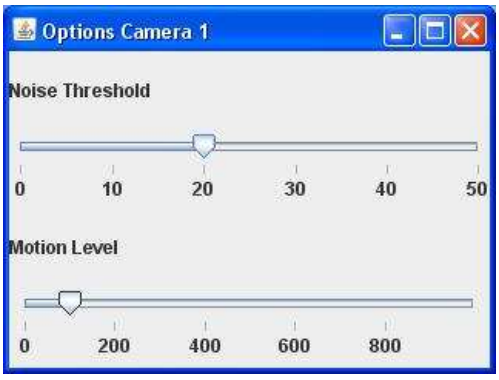

Figura 5.4: opzioni del Motion Detection

- Infine, nella parte inferiore, troviamo quattro pannelli che comunicano lo stato delle connessioni con l'esterno:

    - *Log Video*: comunica lo stato del server Video, ovvero quello che avrà il compito di inviare al cellulare le immagini della telecamera.
    - *Log Control*: comunica lo stato del server di controllo, ovvero quello che accetterà i comandi del cellulare per poi inoltrarli, correttamente mappati, verso l'EventEngine.
    - *Log Engine*: comunica lo stato della connessione verso l'EventEngine, questa connessione viene istanziata immediatamente dopo l'avvio del software.
    - *Log Notification*: comunica lo stato del server delle notifiche delle telecamere.

E' necessario illustrare alcuni dettagli delle quattro aree di log appena definite. Queste aree comunicano i messaggi informativi per l'utente (chiamati impropriamente "di debug") e i messaggi ricevuti e/o inviati da e verso i dispositivi esterni. La convenzione usata indica i messaggi ricevuti come "[RECEIVED]:" mentre quelli inviati come "[SENT]:". Il contenuto di queste aree, inoltre, viene salvato su file tramite un meccanismo di logging (descritto più avanti), per una futura consultazione.

Passiamo ora in dettaglio le icone informative situate nel pannello *Global Status*. Queste sono collegate a tutti i meccanismi di comunicazione con l'esterno e comunicano, in modo visivo, lo stato delle connessioni e i messaggi scambiati. Per indicare che la connessione è attiva diventano colorate (altrimenti sarebbero grigie), mentre per indicare la ricezione o l'invio di un messaggio cambiano il bordo da grigio a rosso tramite un meccanismo di intermittenza denominato *Blink*. Le icone in totale sono 8. Le esaminiamo per colonne, da sinistra a destra:

- La prima colonna mostra lo stato delle due telecamere. Queste due icone non sono state dotate del meccanismo di *Blink*. Si è scelto, inoltre, di non creare il server Video nel caso dell'assenza del segnale di almeno una telecamera. In quest'eventualità viene mostrato nel pannello relativo alla telecamera la scritta "No Signal…"
- La seconda colonna si attiva quando c'è, rispettivamente, la connessione video e la connessione di controllo con il cellulare.
- La terza colonna visualizza rispettivamente la connessione in ingresso e in uscita dall'EventEngine. In realtà la connessione è univoca, ma si è scelto di separare le icone per avere un riscontro visivo dei messaggi in uscita e di quelli in ingresso.
- La quarta e ultima colonna mostra lo stato della connessione con le telecamere, nell'eventualità di dover ricevere notifiche. Per semplicità si è scelto una connessione singola, ciò significa che solo una telecamera alla volta può collegarsi per inviare i propri messaggi.

Ogni icona è dotata di *Tooltip*, in modo da favorirne la comprensione del significato.

5.1.4 Caratteristiche

5.1.4.1 Decodificatore di flusso MJPEG

Le telecamere di cui è dotato il laboratorio domotico consentono l'invio delle immagini tramite due formati: MPEG-4 e MJPEG. E' stato scelto quest'ultimo, in relazione alla sua semplicità di implementazione nell'ambiente Java. Come già accennato, il MJPEG è un codec che incapsula

nello stream i frame video compressi in JPEG. Il decodificatore costruito, quindi, ha solamente il compito di rimuovere le informazioni che non appartengono al frame JPEG e di salvare quest'ultimo all'interno di un oggetto `Image`, dopo aver utilizzato l'apposito `JPEGImageDecoder`.

Prima di osservare nel dettaglio questo meccanismo è importante sottolineare che la classe Java, destinata a questo scopo (`AxisCamera`), è stata costruita sulla base delle telecamere IP modello *Axis 212 PTZ*[xv]. Ciò significa che potrebbe potenzialmente non funzionare con altri modelli e con altre marche di telecamere IP. La semplicità del decodificatore, comunque, permetterebbe, con il cambiamento di alcuni parametri, la compatibilità con tutte le telecamere IP che utilizzano il codec MJPEG.

Il meccanismo usato si compone di una connessione http che richiede, tramite un metodo GET, il video. L'indirizzo utilizzato per la richiesta è, nel nostro caso: http://<ip-camera>/axis-cgi/mjpeg/video.cgi?resolution=320x240. L'argomento "resolution" specifica la risoluzione che deve possedere il video, ovvero 320x240 pixel. Se la richiesta ha avuto successo, il server ritorna un flusso continuo di files JPEG. Il *content type* è di tipo "multipart/x-mixed-replace", inoltre ogni immagine termina con una stringa (<*boundary*>) ad indicare il confine tra un immagine e la successiva. In particolare il flusso ha la seguente struttura:

```
--<boundary>
<image>

<JPEG image data>
--<boundary>
<image>
```

dove il campo <*boundary*> vale, nel nostro caso, "*myboundary*", mentre il campo <*image*> ritornato è:

```
Content-Type: image/jpeg
Content-Length: <image size>
```

Un esempio concreto di questa struttura potrebbe essere:

```
--myboundary
Type: image/jpeg
Content-Length: 15656
<JPEG image data>
--myboundary
Content-Type: image/jpeg
Content-Length: 14978
<JPEG image data>
--myboundary
Content-Type: image/jpeg
Content-Length: 15136
<JPEG image data>
--myboundary
```

La classe `AxisCamera` viene quindi istanziata per ogni telecamera. Estende `JComponent` e pertanto permette di utilizzare il metodo `paint(Graphics g)` e, quindi, di disegnare le immagini ricevute dalla telecamere. Inoltre la classe implementa l'interfaccia `Runnable` in modo da poter essere eseguita in un thread separato. All'interno della stessa viene, inoltre, inizializzato il meccanismo di Motion Detection descritto in precedenza nella sezione MotionDomus.

Nel caso l'accesso all'URL delle telecamere fosse protetto tramite una combinazione *username/password* per prevenire accessi indesiderati dall'esterno, è stata aggiunta una classe `Authenticator` in grado di svolgere questo compito. Non è stata, però, né implementata né testata.

Uno degli sviluppi futuri potrebbe essere l'implementazione di un sistema di ricezione delle immagini basato sul protocollo RTSP, concettualmente più adatto al trasporto di flussi video rispetto al più generico HTTP.

5.1.4.2 Connessioni

La `MainWindow` istanzia un thread separato per ogni Server (Video, Control e Notification), successivamente vengono create le socket lato server (`ServerSocket`) in ascolto sulle rispettive porte definite nel file di configurazione.

La struttura è analoga per tutte e tre le connessioni.

Video Server:

Consultare la sezione relativa nella parte MotionDomus, in quanto il funzionamento è analogo.

Control Server:

La classe che gestisce lo scambio dei messaggi di controllo è chiamata `ControlConnection`. Questa classe istanzia due nuovi thread (`RXThread`, `TXThread`), rispettivamente per l'invio e la ricezione dei messaggi di controllo tramite `InputStream` e `OutputStream`. Dispone dei metodi per notificare ai listeners registrati la ricezione o l'invio di un messaggio.

Notification Server:

La classe `NotificationConnection` si occupa della connessione per le notifiche che le telecamere possono inviare al ControlDomus. La sua struttura è essenzialmente analoga alla classe `ControlConnection`, con una differenza sostanziale: l'unidirezionalità dei messaggi. Infatti la classe deve solamente ricevere messaggi e non inviarne. L'unico thread istanziato è quindi quello di ricezione (`RXThread`).
Inoltre, come la classe `ControlConnection`, dispone dei metodi per notificare ai listeners registrati l'effettiva ricezione di un messaggio.

Una precisazione:
Come già detto in precedenza, si è scelto di non avviare il thread del Server Video nel caso non ci fosse il segnale di almeno una delle videocamere. Di conseguenza anche il thread del Server delle notifiche non viene avviato.

EngineClient:

`EngineClient` è la classe che si occupa di istanziare la socket lato client per la connessione con l'EngineEvent. La porta su cui opera la connessione deve essere di default la 1111, ma attraverso il file di configurazione è possibile cambiarla. Il meccanismo è modellato secondo il funzionamento dell'EventEngine. Per lo stream di input è stato usato un `BufferedReader` mentre per l'output è stato usato un `PrintWriter`. In analogia con le altre connessioni, sono presenti i metodi per notificare ai listeners l'invio e la ricezione di un messaggio.

5.1.4.3 Logging

Il sistema è dotato, come già accennato, di un utility di log dei messaggi per una futura consultazione su file. A seconda della classe che registra il costruttore `Log` viene creato il rispettivo file. Con il sistema completo vengono quindi creati quattro file: `logVideo.log`, `logControl.log`, `logEngine.log` e `logNotification.log`. I messaggi vengono salvati con la seguente struttura:

[Day Month Number Hour:Minute:Second CEST Year] – Message

Dove *Day* indica il giorno della settimana (Mon/Tue/Wed/Thu/Fri/Sat/Sun) e CEST indica il fuso orario dell'Europa Centrale quando è in vigore l'ora legale. Il campo *Message*, invece, definisce il messaggio inviato o ricevuto, formattato esattamente come viene visualizzato nella rispettiva area di log all'interno dell'interfaccia grafica.

5.1.4.4 Invio SMS

Come nel MotionDomus, la rilevazione del movimento comporta l'invio di un SMS al numero configurato. L'SMS è sempre di tipo 8 bit e consente, tramite il push registry, di attivare l'applicazione su dispositivo mobile.

A questo sistema è stato affiancato anche il meccanismo per l'invio di semplici SMS testuali, in modo da notificare all'utente l'occorrenza di un particolare evento. Il servizio si appoggia sempre su una pagina in php che si occupa di effetturare il forwarding dell'SMS verso il gateway di Mobyt. Il funzionamento è analogo al meccanismo già visto per il sistema MotionDomus. Alla funzione `sendSmsMobyt8bit(String telNumber)` è stata affiancata la funzione `sendSmsMobytText(String telNumber, StringContent)` che crea una connessione di tipo http, in modo da invocare la pagina in php passando come parametri il numero telefonico di destinazione del messaggio e il contenuto dello stesso. Sarà poi questa pagina ad assumersi il compito di formattare correttamente il messaggio, per poi inoltrarlo correttamente al gateway di invio degli SMS fornito da Mobyt.

5.1.5 Strutture messaggi

Nonostante tutte le forme di comunicazione previste dal ControlDomus siano effettuate tramite stringhe e array di byte (nel caso dell'invio del video), si è pensato di costruire delle strutture apposite per i messaggi di controllo, scambiati con il telefono cellulare, e per i messaggi scambiati con l'EventEngine. I nomi di queste strutture sono rispettivamente `ControlMessage` ed `EngineMessage`. Attualmente la costruzione è analoga in entrambi i casi e comprende solamente un campo `text` contenente la stringa scambiata, più il costruttore ed i metodi `setter` e `getter`:

```
public class EngineMessage {

private String text;

    public EngineMessage(String text) {
    ...
    }

    public void setText(String text) {
    ...
    }

    public String getText() {
    ...
    }
}
```

In futuro potrebbe essere estesa in modo da ospitare ulteriori campi come, per esempio, un *timestamp*, in modo da incrementare le possibili applicazioni, soprattutto nello scambio di messaggi con l'EngineEvent. Requisito fondamentale dovrà però essere l'implementazione della serializzazione.
Nel caso della comunicazione con la MIDlet, invece, è necessario costruire manualmente i metodi di serializzazione in quanto la configurazione CLDC non comprende meccanismi nativi per questo scopo (come invece accade in J2SE), a causa dell'alto utilizzo di risorse necessarie.

Nel caso delle notifiche recevute dalle telecamere, invece, non avrebbe avuto senso costruire una struttura apposita, in quanto la comunicazione può avvenire solamente tramite stringhe. Queste vengono formattate nel seguente modo:

%e|%E

ovvero il nome dell'evento (%e) e la sua descrizione (%E). Il programma si farà carico di identificare, tramite l'indirizzo IP, quale telecamera ha inviato la notifica.

5.1.6 Protocolli

Con il termine *protocollo* indichiamo un insieme di regole formalmente descritte, definite al fine di favorire la comunicazione tra una o più entità. Nel nostro caso si è cercato di mantenere il più ristretto possibile questo insieme di regole, in modo da facilitare la costruzione di un protocollo ad-hoc partendo da zero quando il sistema sarà, in un futuro che ci auguriamo prossimo, effettivamente omogeneo.

5.1.6.1 Protocollo video

Nel caso dello scambio del video si è estesa la convenzione utilizzata da Michele Mussi. La MIDlet, all'atto della connessione video con il ControlDomus, infatti, invia a quest'ultimo la risoluzione dell'area visibile che conterrà le immagini, tramite una stringa formattata nel seguente modo:

resolution@width@height

Infine per la richiesta delle immagini invia rispettivamente il carattere "j" per il video della telecamera 1 e il carattere "i" per il video della telecamera 2. Poi, come già descritto nella sezione "Invio Immagini" parlando del MotionDomus, l'operazione di'invio è composta, in realtà, da due operazioni distinte. La prima invia alla MIDlet la dimensione dell'array che contiene l'immagine, mentre la seconda comprende la trasmissione vera e propria del frame tramite un array di bytes. Una rappresentazione schematica del protocollo video è mostrata in figura 5.5.

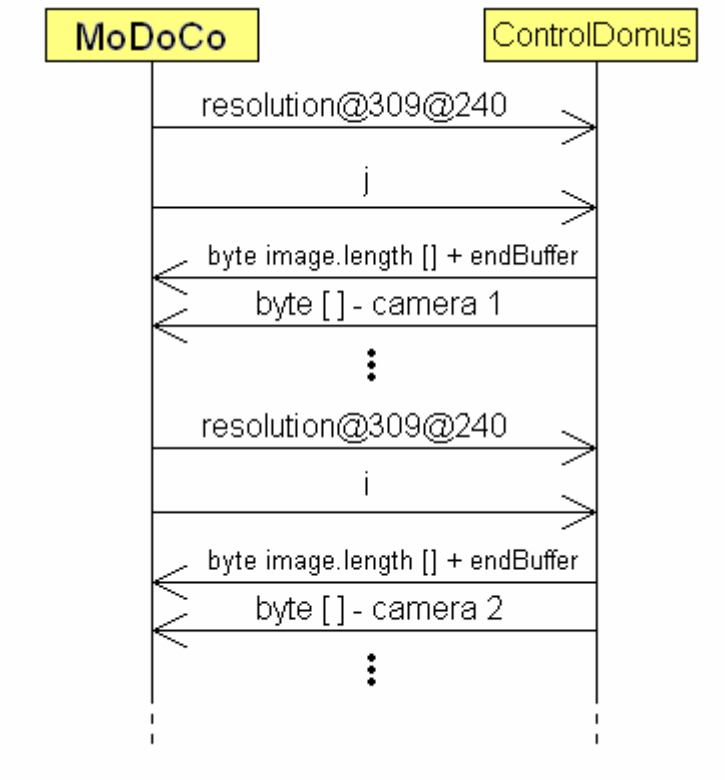

Figura 5.5: protocollo video

5.1.6.2 Protocollo di controllo

Nel caso dell'invio, da parte della MIDlet, dei messaggi di controllo destinati all'EngineEvent l'inizializzazione della connessione viene notificata al ControlDomus tramite un `ControlMessage` contenente la stringa "INITCONTROL". La risposta prevede invece la stringa "INIT_OK" comunicata analogamente tramite `ControlMessage`. Può così cominciare l'invio dei messaggi di controllo da parte della MIDlet. A questi messaggi non corrisponde alcuna risposta. Il protocollo infatti è "stateless", ovvero non ha alcuna informazione sullo stato degli elementi modificati. Ciò significa che l'utente che utilizza il telefono cellulare non può avere la piena certezza che il comando sia effettivamente arrivato a destinazione e quindi sia stato eseguito. Questa scelta è stata dettata dal fatto che sarebbe stato necessario aggiungere un controllo dell'effettiva attuazione del comando anche sull'EventEngine, in modo da comunicarlo al ControlDomus e quindi alla MIDlet collegata. La predisposizione dei thread dedicati all'invio e alla ricezione dei messaggi di controllo sia nel ControlDomus, sia nella MIDlet consente comunque una facile costruzione di un protocollo che tiene traccia dello stato degli elementi modificati. La rappresentazione schematica del protocollo di controllo è visibile in figura 5.6.

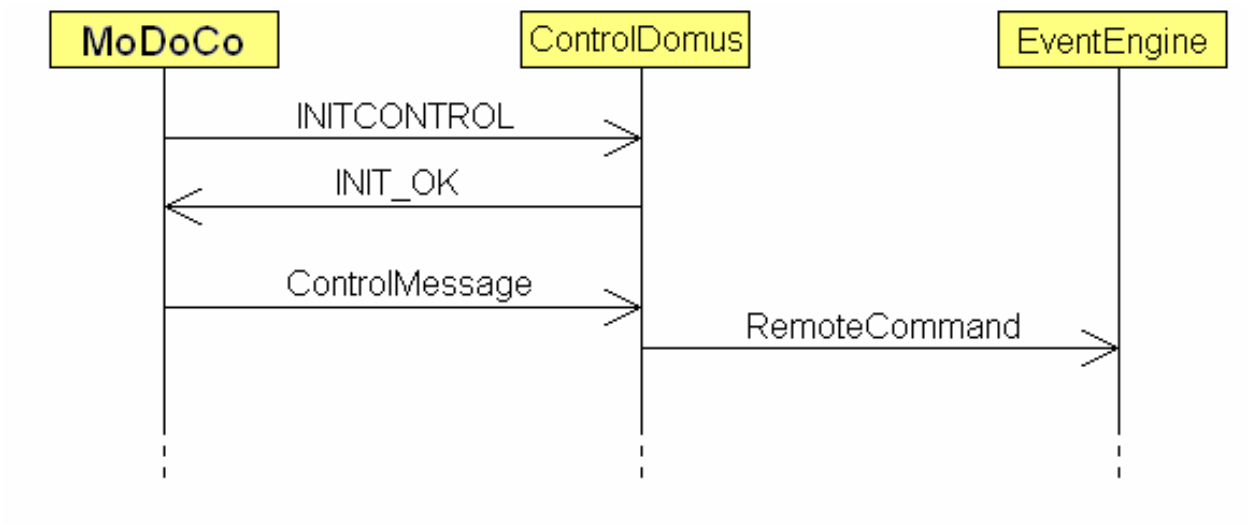

Figura 5.6: protocollo di controllo

5.1.6.3 Protocollo EventEngine

La costruzione dell'EventEngine richiede che gli eventuali client collegati si identifichino tramite una stringa. All'atto della connessione viene quindi inviato il comando "CLIENTMOBILE" per informare l'EventEngine che un client di tipo "mobile" si è appena collegato. Questo meccanismo consente la differenziazione dei messaggi che l'EventEngine potrà ricevere e inviare. Infatti dal ControlDomus si aspetta solamente di ricevere comandi di tipo "remote", formattati secondo la convenzione vista nella sezione XML (es. MOB|ACCENDICUCINA) mentre, per esempio, dal client che gestisce i sensori wireless si aspetta solamente informazioni sulla posizione dei mote. Successivamente a questo comando, la comunicazione è inizializzata in entrambi i sensi. Nel caso l'EventEngine registrasse un errore può inviare il comando "DISCONNECT" in modo da forzare la disconnessione dei client collegati. La struttura del protocollo EventEngine è schematizzata in figura 5.7.

Si noti che attualmente la struttura `EngineMessage` è implementata solamente sul ControlDomus e non nell'EventEngine.

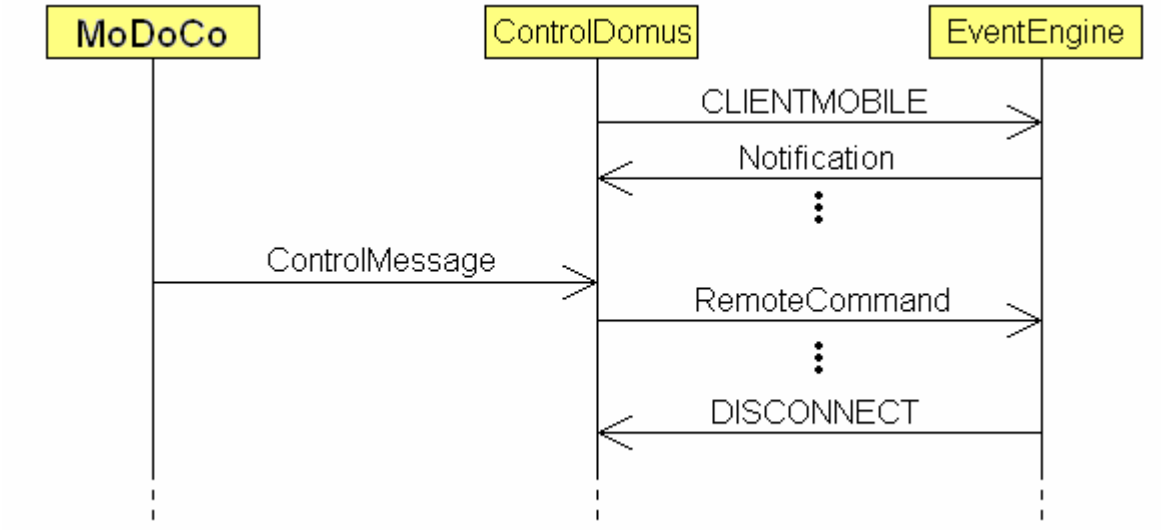

Figura 5.7: protocollo EventEngine

5.1.6.4 Protocollo Notifiche Telecamere

La comunicazione delle notifiche da parte delle telecamere non segue, in realtà, alcun protocollo specifico. Le telecamere si connettono alla socket, inviano la notifica formattata, nel nostro caso, come già visto nella sezione Messaggi, ed infine si disconnettono.

5.1.7 Listeners

Perché listeners?

Il senso di creare un meccanismo di listeners è semplice e chiaro. La classe `MainWindow`, che costruisce l'interfaccia utente, deve intraprendere azioni diverse in base ai messaggi inviati e ricevuti dalle quattro possibili connessioni con l'esterno. Inoltre anche il meccanismo di logging dei messaggi su file deve poter accedere allo scambio di messaggi di ogni connessione. Per questo, le classi che implementano le connessioni, per informare la `MainWindow` (che aggiornerà le rispettive aree di log) e la classe `Log` (che scriverà su file i messaggi) utilizzano i listeners, come miglior soluzione a questo problema.
Infatti, a differenza di altri linguaggi object-oriented, Java non permette l'ereditarietà multipla di classi. Questa sembra essere in generale una saggia decisione, ma in alcuni casi può essere uno svantaggio. Una classe può ereditare solo da un'altra classe. Il meccanismo per compensare questo comportamento è permettere l'implementazione di un qualsiasi numero di interfacce. Le classi sono "estese" e le interfacce sono "implementate". Nella creazione di un listener tutti i metodi dell'interfaccia devono essere implementati.

Le classi `MainWindow` e `Log` (che analizzeremo in seguito) implementano tutte le interfacce disponibili che, come è facile immaginare, sono in tutto quattro: `VideoListener`, `ControlListener`, `EngineListener` ed infine `NotificationListener`. La struttura per le prime tre è identica:

```
public interface ControlListener {
    public void messageReceived(ControlMessage controlMessage);
    public void messageSent(ControlMessage controlMessage);
    public void debug(String sender, String body);
}
```

l'interfaccia `NotificationListener` invece, come è facilmente immaginabile, non definisce alcun metodo `notificationSent`.
La classe `EngineClient` che si occupa della comunicazione con l'EngineEvent implementa a sua volta l'interfaccia `ControlListener`. Infatti questa classe si fa carico, tra le altre cose, di mappare correttamente i `ControlMessage` ricevuti dalla MIDlet con i comandi remoti definiti nel file XML `MappedCommands.xml`.

5.2 Progetto MoDoCo

Il client per cellulare è stato ovviamente esteso, rispetto alla versione utilizzata con il MotionDomus, per permettere l'utilizzo delle nuove funzionalità. E' stata aggiunta la possibilità di scegliere da quale telecamera ricevere le immagini, inoltre si è implementato un meccanismo per gestire alcuni elementi in remoto. Questi elementi possono essere caricati dinamicamente tramite un file XML contenente tutte le informazioni necessarie. Questo approccio consente di personalizzare

completamente gli elementi gestibili dal telefono cellulare, in modo da adattarsi alle specifiche definite dal ControlDomus e, soprattutto, dalla propria abitazione.

5.2.1 Elementi XML

Per poter comandare a distanza elettrodomestici, valvole, impianti, o qualsiasi apparecchio gestibile tramite una scheda a relé, è necessario caricare nella memoria del telefono cellulare la struttura di questi elementi, assieme al loro stato attuale. Con il sistema completo è possibile, infatti, comandare dal telefono cellulare l'accensione/spegnimento, o apertura/chiusura, di dispositivi collegati alla scheda relé tramite l'EventEngine.

Questo file, definito nel formato XML, contiene tutte le informazioni per gestire correttamente diverse tipologie di apparecchi. Al momento sono definiti sei elementi: porta d'ingresso, finestra, luce ingresso, luce cucina, valvola gas, televisione. Di questi sei, solamente a quattro possono essere realmente inviati i comandi di attuazione. La scheda USB a relé collegata all'EventEngine, infatti, possiede solamente 4 relé. In commercio esistono, comunque, schede con un numero di relé più elevato.

Osserviamo ora la struttura del file che definisce gli elementi modificabili. Questa struttura comprende un numero arbitrario di elementi, ognuno identificato dal tag <*element*>. Un singolo elemento possiede sette caratteristiche che ne definiscono la visualizzazione e il comportamento. Il tag <*name*> identifica il nome, il tag <*act_as*> ne identifica la categoria d'appartenenza (per esempio la porta d'ingresso e la porta del terrazzo appartengono alla stessa categoria di "porta"), il tag <*actual_state*> definisce lo stato attuale dell'elemento. Attualmente può assumere due diverse coppie di valori (definite successivamente con i tag <*state*>). Questi valori sono: APERTO/CHIUSO e ACCESO/SPENTO. Per una finestra, ad esempio, non ha senso parlare di acceso o spento, ma solo di aperto o chiuso. I due tag <*command*>, definiti successivamente al tag <*commands*>, stabiliscono i comandi che la MIDlet invierà al ControlDomus per modificare lo stato dell'elemento. Questi comandi devono essere ovviamente identici ai *remote_command* definiti nel file `MappedCommands.xml` del ControlDomus. In caso contrario il comando verrà ignorato.

```
<elements>
    <element>
        <name>Porta Ingresso</name>
        <act_as>porta</act_as>
        <actual_state>CHIUSO</actual_state>
        <states>
            <state>APERTO</state>
            <state>CHIUSO</state>
        </states>
        <commands>
            <command>PORTA|1|OPEN</command>
            <command>PORTA|1|CLOSE</command>
        </commands>
    </element>
    .
    .
    .
</elements>
```

L'unico scopo del tag <*act_as*>, attualmente, è definire la corretta icona associata al dispositivo.

5.2.2 Interfaccia grafica

Prima di osservare in dettaglio tutte le caratteristiche, ci concentriamo sull'interfaccia grafica della MIDlet, illustrata in figura 5.8.

All’avvio della MIDlet possiamo osservare la schermata principale, composta da cinque elementi. Li descriviamo tutti, nonostante il significato di alcuni sia già noto. Il primo elemento, chiamato *Camera*, consente di visualizzare, a schermo intero, le immagini della telecamera definita nel menù *Settings* (di default è la numero 1). L’elemento successivo, chiamato *Control*, consente di visualizzare i dispositivi a cui è possibile modificare lo stato. Al primo avvio assoluto dell’applicazione questo elenco risulterà vuoto. E’ compito dell’utente, infatti, riempirlo caricando il file XML relativo che ne descrive le caratteristiche. Queste verranno poi memorizzate nel RMS in modo da risultare persistenti. Il terzo elemento, come appena detto, consente di recuperare da un URL tale file XML. L’elemento *Settings* permette di inserire i parametri di connessione (indirizzo IP, porta video, porta controllo) verso il ControlDomus e la telecamera da visualizzare (1 o 2). Infine l’elemento *Esci* chiude l’applicazione.

Figura 5.8: interfaccia grafica della MIDlet

5.2.3 Funzionamento

Esaminiamo ora in dettaglio ogni aspetto della MIDlet, corredando la descrizione con immagini pratiche del funzionamento. La spiegazione non prende in esame gli elementi del menù in modo sequenziale, ma piuttosto ne descrive il comportamento come se fosse una normale sessione di utilizzo. La parte tecnica relativa a come queste funzioni sono implementate verrà trattata nel paragrafo successivo.

5.2.3.1 Settings

Il menù *Settings* (figura 5.9) consente di impostare i paramentri di creazione della socket con il Domus Control. I valori che è possibile inserire, quindi, sono l’indirizzo IP della macchina su cui è attivo il Domus Control stesso, la porta del servizio video e la porta del servizio di controllo. E’ possibile inoltre specificare, tramite un `choiceGroup`, da quale telecamera si desidera ricevere le immagini.

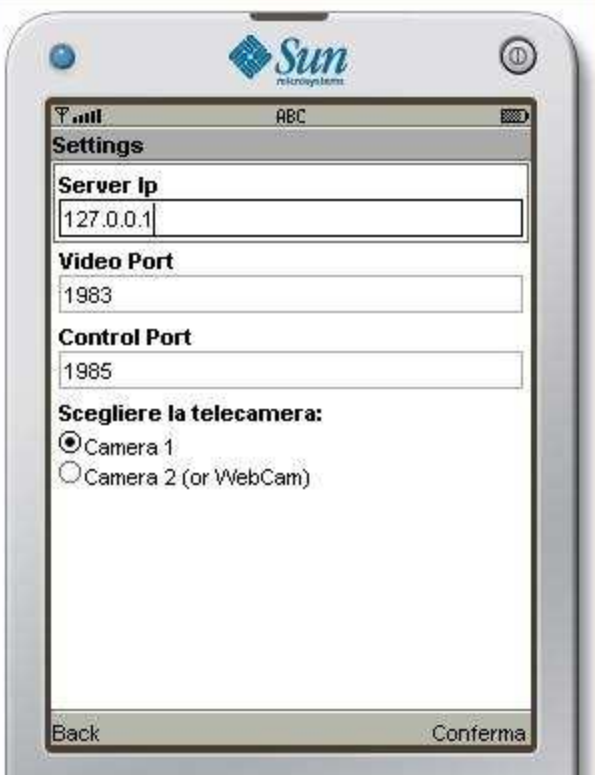

Figura 5.9: menù Settings

Questo client consente, inoltre, di collegarsi anche con il MotionDomus, impostando l'apposito indirizzo IP e la porta video. E' necessario però selezionare la Camera 2. Il meccanismo di richiesta e ricezione delle immagini è sostanzialmente identico, quindi il singolo client può operare con entrambi i sistemi, espandendo ulteriormente gli ambiti di applicazione.

5.2.3.2 Camera

La selezione dell'elemento Camera comporta l'inizializzazione della socket per la ricezione del video, con i parametri indirizzo IP e porta video specificati nel menù *Settings*. L'applicazione è ancora in grado di impostare, alla ricezione di un SMS a 8 bit, l'indirizzo IP salvato al suo interno.

Una volta realizzata la socket vengono inizializzati gli stream di Input e di Output e la comunicazione può cominciare. La figura 5.10 illustra il funzionamento del sistema mostrando l'iterazione tra il dispositivo mobile, rappresentato tramite l'emulatore, e il sistema DomusControl.

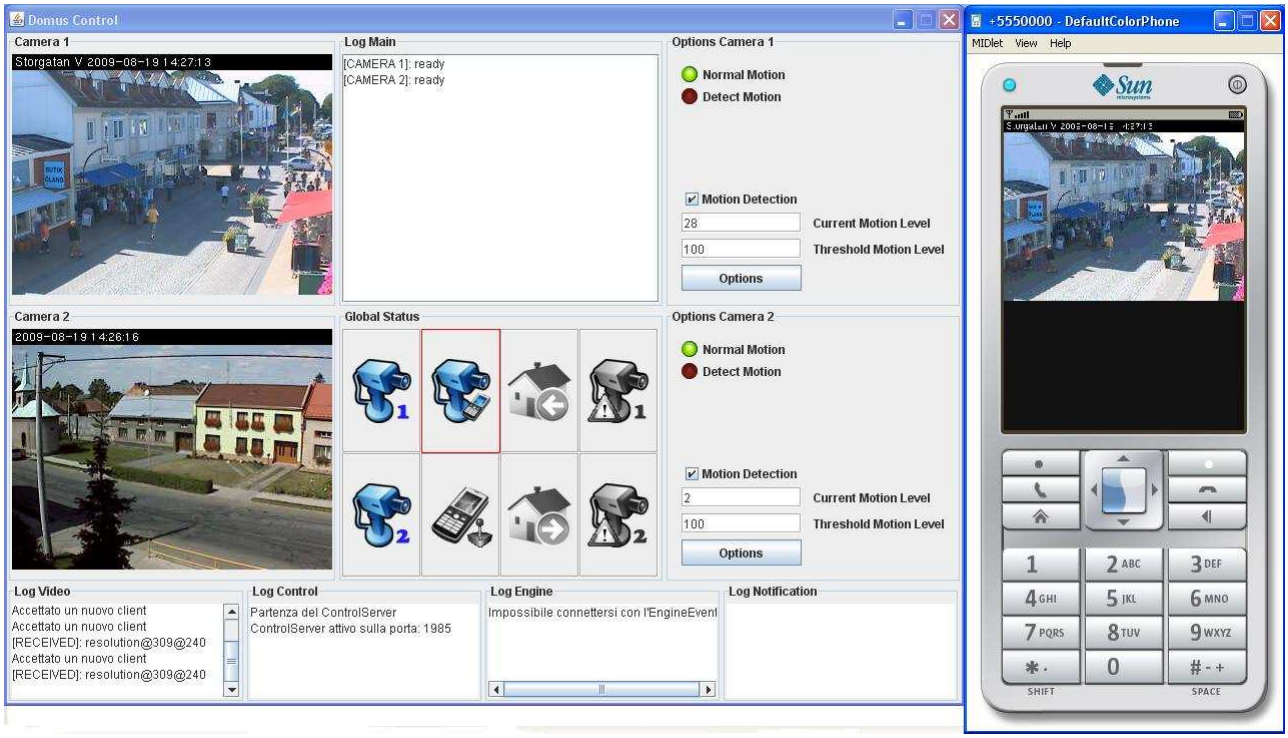

Figura 5.10: il sistema in funzione

5.2.3.3 XML

Tramite la form visualizzata in figura 5.11 è possibile specificare l'URL del file XML usato per descrivere i vari elementi. Un `textField` non editabile consente di informare l'utente se il file è già stato caricato oppure no.

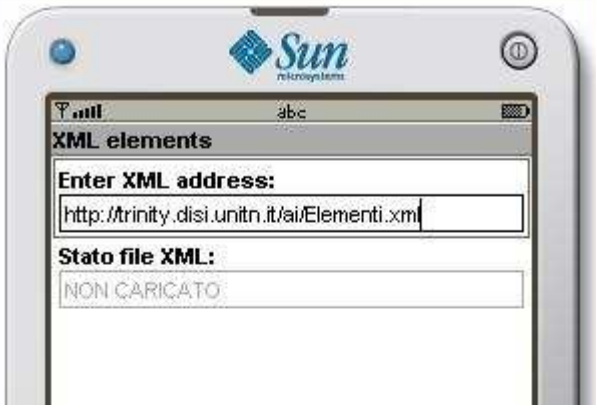

Figura 5.11: la schermata per il download del file XML degli elementi

Il parser che si è scelto di utilizzare è definito nel package `javax.xml.parsers` (JSR 172). Queste API consentono di creare facilmente un parser di tipo SAX, ovvero che non carica in memoria tutto il documento, ma che invece lo carica una riga per volta, richiamando via via delle funzioni definite all'interno dell'applicazione che lo ha istanziato.

Tecnicamente le JSR 172 implementano un sottoinsieme delle JAXP 1.2 (Java API for XML Processing). Essenzialmente, quindi, sono supportate le specifiche SAX 2.0 (Simple API for XML), con l'eccezione delle interfacce SAX 1.0, DOM 1.0 e 2.0 (Document Object Model) ed infine XSLT (Extensible Stylesheet Language Transformation).

Il parser creato, nonostante abbia funzionalità ristrette, è in grado di rilevare correttamente errori di formattazione e di comunicarli all'utente. La validazione dei file XML, non essendo lo scopo finale della MIDlet, non è garantita, in quanto richiederebbe risorse onerose per un dispositivo mobile. Ed inoltre ci si aspetta che sia il produttore a verificare la correttezza del file, non il consumatore.

5.2.3.4 Control

Selezionando dal menù l'opzione Control si possono così visualizzare, a patto di aver caricato correttamente il file XML, tutti gli elementi modificabili da remoto. La visualizzazione è affidata ad una classica lista implicita nella quale vengono visualizzati i nomi degli elementi assieme ad una icona raffigurante una lente d'ingrandimento, per indicare che ogni elemento è selezionabile per essere ulteriormente approfondito (figura 5.12).

Figura 5.12: visualizzazione degli elementi modificabili

La connessione con il ControlDomus viene effettuata solamente su richiesta dell'utente, tramite il comando `Connect`.
La classe `ControlConnection` crea una socket utilizzando l'indirizzo IP, definito per la ricezione delle immagini delle telecamere, e la rispettiva porta di controllo stabiliti nel menù *Settings*. Non appena la connessione è attiva viene notificata all'utente tramite un `Alert`. Può così iniziare la procedura d'invio dei comandi.

La gestione di un singolo elemento, di cui la figura 5.13 riporta un esempio, è molto semplice ed intuitiva. Questa rispecchia pienamente la struttura del file XML che lo definisce. Nella parte superiore troviamo un icona che descrive visualmente l'elemento. Successivamente troviamo il nome ed un choiceGroup che consente di modificare lo stato dell'elemento. Nelle ultime tre righe troviamo, invece, lo stato attuale che può assumere i due valori definiti dagli stati disponibili ed infine la formattazione dei comandi.
Quindi, ad esempio, per aprire la porta è sufficiente selezionare l'opzione APERTO e premere il pulsante associato al comando Set. Il nuovo stato verrà indicato tramite il textField *Stato Attuale*, mentre il comando inviato al ControlDomus verrà mappato per comunicare all'EventEngine la richiesta di apertura della porta. La figura 5.14 mostra visivamente la ricezione del comando di apertura della porta, visibile nell'area di log. Il relé corrispondente viene acceso e la MapArea viene aggiornata visualizzando l'effettiva apertura della porta.

Figura 5.13: schermata per la gestione di un singolo elemento

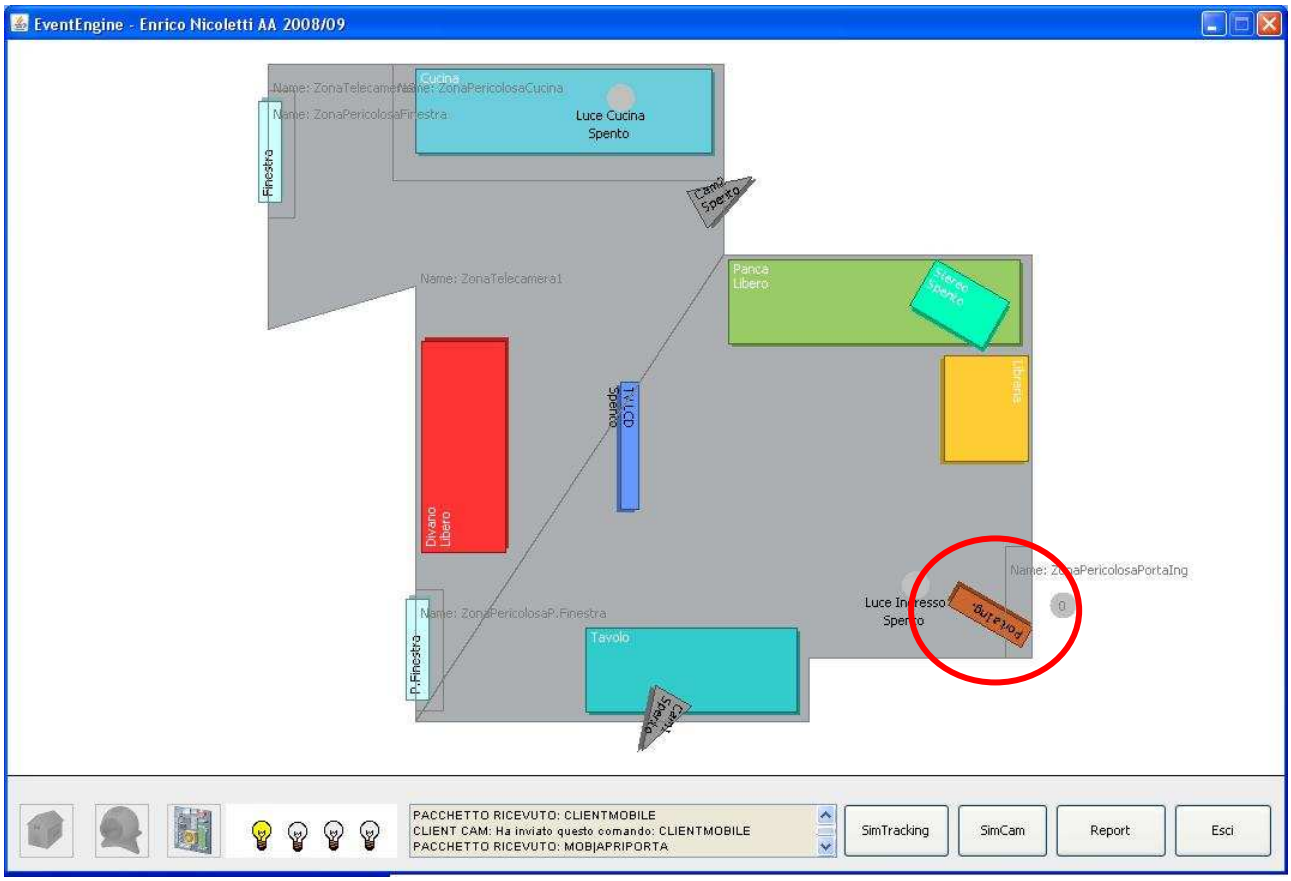

Figura 5.14: riscontro visivo del comando ricevuto dal cellulare

5.2.4 Implementazione

5.2.4.1 Ricezione messaggi

Abbiamo già visto come sia possibile, tramite un attributo specifico nel file JAD, attivare una MIDlet alla ricezione di un semplice SMS. La MIDlet stessa deve fornire, però, i metodi necessari alla gestione di questi SMS. Il meccanismo di ricezione dei messaggi implementato è relativamente semplice e consente di gestire SMS di tipo binario, estraendo da esso le informazioni codificate ovvero l'indirizzo IP del PC che lo ha inviato.

Abbiamo visto che il lifecycle di una MIDlet generica è determinato da tre metodi fondamentali:

- `startApp()` – per default l'esecuzione della MIDlet comincia da questo metodo
- `pauseApp()` – viene richiamato quando la MIDlet deve sospendere temporaneamente la sua esecuzione
- `destroyApp()` – viene richiamato quando la MIDlet ha terminato la sua esecuzione e deve quindi liberare le risorse da essa utilizzate

Quando un terminale riceve un SMS destinato ad una specifica porta, l'AMS esegue l'applicazione registrata tramite quella stessa porta, se presente, incaricandola di gestire l'evento. Viene quindi richiamato il metodo `startApp()` della MIDlet associata che, nel nostro caso, ha la seguente struttura:

```
public class MobileDomusController extends MIDlet {
    ...
    public void startApp() {
        if (midletPaused) {
            resumeMIDlet();
        } else {
            initialise();
            startMIDlet();
        }
        midletPaused = false;
    }

    public void initialise() {
        SMSReceive mySms = new SMSReceive(this);
    }
}
```

L'istruzione `resumeMIDlet()` definisce i comportamenti della MIDlet nel caso la sua esecuzione riprendesse in seguito alla permanenza nello stato *Paused* forzata, ad esempio, dalla ricezione di una chiamata. L'istruzione `startMIDlet()`, al contrario, è il vero punto di partenza per l'esecuzione della MIDlet e viene chiamata in seguito al metodo `initialise()`. Quest'utlimo, infatti, istanzia una nuova classe, chiamata `SMSReceive`, incaricata di ascoltare la connessione SMS su una specifica porta e di gestire i messaggi ricevuti. La classe `SMSReceive` implementa le interfacce `MessageListener`, in grado di notificare la ricezione di un messaggio, e `Runnable` in modo da essere eseguita in un thread separato.

```
public class SMSReceive implements Runnable, MessageListener {
    MesssageConnection connection;

    public SMSReceive() {
        try {
            connection = (MessageConnection) Connector.open("sms://:5000");
            connection.setMessageListener(this);
        catch (Exception e) {
            e.printStackTrace();
        }
```

```
        Thread thread = new Thread(this);
        thread.start();
    }

    public void run() {
        ...
    }
}
```

Le righe evidenziate permettono, utilizzando l'interfaccia `MessageConnection`, di ricevere gli SMS sulla porta 5000 e di registrare su di essa un listener che informerà l'applicazione nel caso di ricezione di un messaggio. In quest'eventualità, un thread separato si prende carico di scomporre il messaggio, controllando il tipo (`Binary` oppure `Text`), in modo da estrarre il payload che contiene l'indirizzo IP del PC che lo ha inviato. Questa informazione viene poi memorizzata in modo persistente all'interno della memoria del cellulare in modo da essere sempre raggiungibile dall'applicazione.

5.2.4.2 Ricezione immagini

La ricezione delle immagini da parte della MIDlet è effettuata tramite `DataStream` all'interno della classe `VideoClient`. Come per ogni azione che richiede l'uso della rete, viene istanziato un thread per aprire la socket di connessione con il Domus Control.

```
public class VideoClient implements Runnable {
    public void run() {
        try {
            StreamConnection sc = (StreamConnection) Connector.open("socket://"+host+":"+port,
            Connector.READ_WRITE, true);
            DataInputStream is = sc.openDataInputStream();
            DataOutputStream os = sc.openDataOutputStream();
        }
    }
}
```

In particolare l'istruzione in rosso crea una socket bidirezionale (READ_WRITE), utilizzando i parametri indirizzo e porta specificati da `host` e `port`. Il flag booleano che completa l'istruzione sottolinea la capacità del codice di gestire le eccezioni di timeout.

Una volta stabilito il collegamento viene istanziata la classe `WebCam`, responsabile dell'utilizzo delle funzioni di invio e ricezione esposte da `VideoClient` e, soprattutto, della primitiva `paint` per disegnare le immagini sul display. La classe `WebCam`, infatti, oltre ad implementare l'interfaccia `Runnable`, estende la classe `Canvas`, utilizzata per la grafica a basso livello. Il metodo `run` della classe `WebCam` è strutturato nel modo seguente:

```
public class WebCam extends Canvas implements CommandListener, Runnable {
    public void run() {
        Thread s = new Thread(new refresh(nCamera));
        s.start();
        VideoClient.send("resolution@" + height + "@" + width);
    }
}
```

Viene istanziato un nuovo thread che si occupa del refresh delle immagini ed inoltre viene inviata al Domus Control la stringa che identifica la risoluzione del display del terminale mobile. La comunicazione può così cominciare. La MIDlet richiede le immagini di una delle due telecamere (o della webcam), in base alla scelta nel menù Settings. Il metodo `run` della classe interna `refresh` si occupa così, di inviare il carattere chiave specificato dal protocollo, di ricevere i dati e quindi di disegnare l'immagine a tutto schermo tramite il metodo `repaint()`.

```
class refresh implements Runnable {
    public void run() {
```

```
        while (stato) {
            try {
                VideoClient.send(“i/j”);
                Image image = VideoClient.receiveImage();
                Repaint();
            }
        }
    }
}
```

5.2.4.3 Memoria

La gestione della memoria utilizzando, come abbiamo già visto, il RMS è gestita totalmente dalla classe `Memory`. Offre tutti i metodi per la gestione delle informazioni per la connessione, come il salvataggio e il caricamento dell’indirizzo e dei numeri di porta, e i metodi per la gestione degli elementi che è possibile controllare: caricamento, salvataggio, eliminazione e recupero dei nomi. Le informazioni relative ad ogni singolo elemento, per essere utilizzate con maggiore chiarezza, sono state inserite all’interno di una struct apposita, chiamata `ElementStruct`, con la seguente struttura:

```
public ElementStruct();
    String name;
    String actAs;
    String actualState;
    String state1;
    String state2;
    String command1;
    String command2;
}
```

Come si può vedere la costruzione è naturalmente identica a quella definita dal file XML degli elementi. L’attributo che funge da chiave primaria per identificare univocamente la struct è ovviamente la stringa `name`.

5.2.4.4 Listener

Come già sottolineato più volte all’interno della tesi, in una MIDlet le operazioni rischiose, come il networking, devono essere effettuate all’interno di thread separati da quello principale. In caso contrario l’esecuzione dell’applicazione verrebbe bloccata non appena la richiesta di comunicazione venisse inizializzata. Questo comportamento, testimoniato dai numerosi test effettuati con l’emulatore, non è tollerabile in un contesto reale. Pertanto si è reso necessario utilizzare funzioni di *multithreading*, in modo da rendere parallela l’esecuzione senza bloccare il terminale. In questo modo l’utente potrebbe ancora utilizzare la MIDlet mentre questa, in background, esegue le azioni di networking. Si è preferito però impedire questa libertà visualizzando, durante l’utilizzo della rete, un oggetto standard, simile ad una progress bar, chiamato Gauge. Una volta che la MIDlet è connessa o ha terminato le operazioni di networking, viene comunicato l’esito tramite un’*alert*.
Il meccanismo tramite cui i thread figli comunicano al padre lo stato della connessione è affidato ai *Listener*. La classe principale della MIDlet, ovvero `MidletControl`, implementa, oltre all’interfaccia `CommandListener` per la gestione dei comandi da tastiera, anche le interfacce `NetworkListener` e `XmlListener`. La prima si occupa di notificare l’avvenuta connessione della MIDlet con il Domus Control per l’invio dei comandi di controllo ed ha la seguente struttura:

```
public interface NetworkListener {
    public void connected(boolean state);
}
```

La seconda, invece, notifica l’avvenuta ricezione del file XML degli elementi. Ha la seguente struttura:

```
public interface XmlListener {
```

```
    public void XmlReceived(String state);
}
```

In questo caso il parametro utilizzato è una stringa, in modo da coprire una casistica più ampia rispetto al classico vero/falso. In questo modo è possibile notificare alla MIDlet, non solo l'avvenuto caricamento, ma anche se c'è stato un errore di parsing a causa della cattiva formattazione del documento.

5.3 Conclusioni di capitolo

Questo capitolo ha analizzato la seconda parte del sistema, denominata ControlDomus. La trattazione è stata effettuata separatamente sulla parte orientata ad ambienti Desktop, chiamata DomusControl, e sulla parte orientata a dispositivi mobili, chiamata Mobile Domus Control. L'analisi si è svolta, analogamente al 4° capitolo, analizzando il funzionamento pratico utilizzando immagini e sezioni di codice e concentrando, di volta in volta, l'attenzione sui dettagli implementativi.

Conclusioni

Lo scopo di questa tesi era dimostrare che è possibile, con le tecnologie già in nostro possesso, integrare mobile computing e networking, realizzando un sistema di video streaming per dispositivi mobili semplice, efficace e a costi contenuti. Tramite una semplice webcam ed un telefono cellulare è stato possibile costruire un tale sistema, utilizzando tecnologie software differenti, dimostrando le potenzialità che potrebbe avere all'interno del settore della video-sorveglianza.

L'integrazione in ambito domotico ha permesso, inoltre, di estendere il sistema fino a comprendere funzionalità atipiche per un telefono cellulare, come il controllo degli elettrodomestici, le quali saranno prevedibilmente di utilizzo comune nell'immediato futuro. Controllare la propria casa da qualsiasi luogo, infatti, può apparire come uno scenario avveniristico, ma come dimostrato è possibile, già al giorno d'oggi, con tecnologie molto comuni. Infatti il sistema ControlDomus riesce, pur nella sua semplicità e incompletezza, a controllare alcune tipologie di dispositivi da remoto con il semplice ausilio di un telefono cellulare ad un costo limitato.

L'analisi e lo sviluppo di un programma a carattere domotico dimostra la grande copertura interdisciplinare di questa scienza e, inoltre, come l'informatica può legarsi a contesti differenti nonostante, a prima vista, un'associazione sia difficilmente immaginabile.

Il risultato comunque più sorprendente, a mio avviso, è la consapevolezza che le apparecchiature che ci circondano possono essere utilizzate per scopi differenti da quelli che normalmente si possono pensare. Le webcam possono così diventare parte integrante di un sistema di video-sorveglianza e un telefono cellulare di vecchia generazione può, tramite le notifiche via SMS, diventare un efficace cercapersone.
L'underground informatico è in continuo fermento alla ricerca di nuovi utilizzi per i dispositivi elettronici, frutto della modifica o dell'integrazione, in modo da rivelare funzionalità innovative o sorprendentemente inaspettate. La sfida è portarle alla luce del grande pubblico.

Problemi riscontrati e aperti:

I problemi riscontrati nell'arco dell'intero processo di sviluppo hanno rallentato considerevolmente il raggiungimento del traguardo finale, come d'altronde è lecito aspettarsi. I più rilevanti sono stati indubbiamente la comprensione e l'uso dell'architettura RMI e la difficoltà nell'utilizzo di una piattaforma software, ovvero J2ME, con funzionalità e comportamenti differenti rispetto alla distribuzione standard J2SE. In particolare, la difficoltà nell'uso degli emulatori ha comportato la necessità di utilizzare funzioni specifiche all'interno della MIDlet per adempiere ai compiti di debug. Inoltre il controllo dei thread, argomento complesso già in ambiente desktop, è ulteriormente reso difficile dalle politiche dell'AMS di gestione della rete, forzando il programmatore ad alcuni accorgimenti speciali per mantenere l'esecuzione lineare della MIDlet senza bloccare il dispositivo hardware. Inoltre, come già sottolineato, il comportamento della MIDlet su dispositivi reali può differire da quello registrato sugli emulatori. Infatti le fasi di test hanno mostrato occasionalmente funzionamenti anomali a dimostrazione del fatto che una MIDlet, nonostante le ridotte dimensioni e funzionalità rispetto ad un applicativo in ambiente Desktop, richiede una fase di test altrettanto esaustiva.

Sviluppi futuri:

Molti sono gli sviluppi che il sistema potrebbe offrire in futuro, sia tramite espansioni software che hardware.

In primo luogo, l’utilizzo di ulteriori sistemi di sensori eterogenei combinati all’EventEngine potrebbero mettere a disposizione nuove categorie di eventi gestibili dal sistema completo, allargando gli orizzonti di applicazione. Inoltre, come già accennato, sarebbe possibile estendere le strutture dei messaggi scambiati tra l’EventEngine e il Control Domus, includendo nuove informazioni, come per esempio il timestamp. Sarebbe così possibile modellare ulteriormente i meccanismi di risposta, sia per quanto riguarda l’EventEngine stesso, sia per i sistemi collegati, tra cui, ovviamente, il ControlDomus.

Un aspetto che andrà sicuramente sviluppato è la sicurezza delle comunicazioni. Al momento l’EventEngine non è in grado di verificare l'identità dei client connessi. Pertanto si è scelto di non implementarla nemmeno nel collegamento tra il Domus Control e il dispositivo mobile.

Per quanto riguarda la configurazione del software sarà necessario sviluppare un’interfaccia grafica per la generazione dei file XML, in modo da renderla più rapida e intuitiva. Le scelte potrebbero ricadere su un’applicazione stand-alone o su un componente aggiunto all’interno del sistema stesso.

Infine anche lo sviluppo del sistema d’interfacciamento potrebbe ulteriormente essere soggetto a sviluppi. In particolare sarebbe conveniente migrare dal protocollo HTTP al più consono RTSP, certamente più adatto allo streaming video. In questo modo il software potrebbe configurare automaticamente le opzioni della telecamera per adattarla a vari scenari, modificando in tempo reale i parametri necessari. Inoltre sarebbe possibile controllare gli spostamenti di ogni singola telecamera, nel caso queste funzionalità siano supportate, in modo da focalizzare l’attenzione su particolari dettagli.
E’ importante, inoltre, ricordare che il software è modellato sulle telecamere modello Axis 212 PTZ, ma semplicemente modificando alcuni parametri nel codice sarebbe possibile adattarlo alle telecamere IP delle maggiori case produttrici, in modo da garantire una perfetta compatibilità.

NOTE

i. Definizioni rilevanti di domotica:

- "l'insieme delle tecniche e degli studi teso ad integrare nell'abitazione ogni automatismo in materia di sicurezza, gestione energetica, comunicazione…" [23]
- "disciplina che si occupa dell'integrazione dei dispositivi elettronici, degli elettrodomestici e dei sistemi di comunicazione e di controllo che si trovano nelle nostre abitazioni" [24]
- "insieme delle tecniche che trasformano un'abitazione con tutte le sue apparecchiature e i suoi (sotto)sistemi in un edificio intelligente, auto-pilotato nel quale il confort degli abitanti costituisce il centro delle preoccupazioni." [25]
- "l'utilizzo dell'informatica per integrare tutti i sistemi a comando elettrico in un unico sistema di gestione globale." [26]

ii. Électricité de France (EDF): http://www.edf-bleuciel.fr/accueil-141001.html

iii. Enerpoint S.p.A.: http://www.enerpoint.it/operatori/fotovoltaico-italia.php

iv. Istituto Superiore per la Prevenzione e la Sicurezza sul Lavoro: http://www.ispesl.it/osservatorio/BancaDati.asp

v. http://www.repubblica.it/2007/08/sezioni/tecnologia/cellulari/mezzo-mondo-cell/mezzo-mondo-cell.html

vi. DynDNS: http://www.dyndns.com/

vii. http://famoso.disi.unitn.it

viii. Una possibile alternativa alle JMF può essere il progetto open-source **FMJ (Freedom for Media in Java)**. Compatibile con le JMF sotto il profilo delle API, propone un singolo framework che può essere usato per catturare, riprodurre, processare e trasmettere dati multimediali su più piattaforme.
http://fmj-sf.net/

ix. A partire dalla versione J2SE 5.0 il programmatore è stato sollevato dal compito di creare lo Skeleton e lo Stub necessari alla tecnologia RMI. Qualora si utilizzasse una versione più vecchia di JRE bisognerà, invece, avvalersi del comando *rmic* per produrre queste due classi.

x. RMI plug-in for Eclipse: http://www.genady.net/rmi/index.html

xi. http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/formats.html#Capturers

xii. http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/download.html

xiii. Mobyt progetta e realizza piattaforme per la fruizione di servizi di messaggistica su telefonia mobile: http://www.mobyt.it/

xiv. http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/solutions/AVTransmit.html

xv. Axis 212 PTZ specifications: http://www.axis.com/files/datasheet/ds_212ptz-v_34051_en_0812_lo.pdf

BIBLIOGRAFIA

[1] Burkhardt, François; *Crossing, rivista di architettura e tecnologia*

[2] Definizione di domotica: http://it.wikipedia.org/wiki/Domotica

[3] Simonazzi, Marco; *La domotica: dalla casa elettrica alla casa elettronica*, Energia in Casa, AEM 1991

[4] Lori A., Golini A., Cantalini B., Bruno P., Citoni F., Paganelli F. (a cura di), CNR-IRP, *Atlante dell'invecchiamento della popolazione. Vent'anni di evoluzione nelle province e nei comuni italiani*, Roma, 1995.

[5] IEEE 802.3 stadard: http://www.ieee802.org/3/

[6] Système sur courant porteur X-10: www.x10.com

[7] The Official Bluetooth Technology: http://www.bluetooth.com/bluetooth/

[8] KNX Association: http://www.knx.org/

[9] EIBA – European Installation Bus Association: www.eiba.com

[10] BCI – Batibus Club International: http://www.batibus.com/

[11] Universal Mobile Telecommunications System: http://www.umtsworld.com/

[12] J2ME – Java Micro Edition: http://java.sun.com/javame/index.jsp

[13] Enrico Nicoletti, *Progetto e realizzazione di una piattaforma di programmazione per applicazioni domotiche*, tesi di laurea, 9-11, anno accademico 2008 – 2009

[14] JMF: http://java.sun.com/javase/technologies/desktop/media/jmf/

[15] WMA: http://java.sun.com/products/wma/index.jsp

[16] Enrique Ortiz, *The MIDP 2.0 Push Registry*, http://developers.sun.com/mobility/midp/articles/pushreg/, 2003

[17] Eric Giguere, *Databases and MIDP, Part 1: Understanding the Record Management System*, http://developers.sun.com/mobility/midp/articles/databaserms/, 2004

[18] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, http://tools.ietf.org/html/rfc3550, July 2003

[19] C. Huitema, *Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)*, http://tools.ietf.org/html/rfc3605,Microsoft, October 2003

[20] Marco Altese, *RMI: metodi a distanza*, http://java.html.it/articoli/leggi/1767/rmi-metodi-a-distanza/, 2006

[21] Qusay Mahmoud, *Advanced MIDP Networking, Accessing Using Sockets and RMI from MIDP-enabled Devices*, http://developers.sun.com/mobility/midp/articles/socketRMI/, 2002

[22] L. Berc, W. Fenner, R. Frederick, S. McCanne, P. Stewart, *RTP Payload Format for JPEG-compressed Video*, http://tools.ietf.org/html/rfc2435, October 1998

[23] « Ensemble des techniques et des études tendant a intégrer à l'habitat tous les automatismes en matière de sécurité, de gestion de l'énergie, de communication, etc », Dizionario Larousse, 2002

[24] Il portale italiano dell'home automation, www.domotica.it

[25] « Ensemble des techniques qui transforment une habitation avec tous ses appareils et ses (sous)systèmes en un bâtiment intelligent, auto-piloté dans lequel le confort des habitants constitue le centre des préoccupations. », Bits & Bytes Home Système, www.bitsbytes.be

[26] « utilisation de l'informatique pour intégrer tous les systèmes a commande électrique dans un seul système de gestion centrale », Bits & Bytes Home Système, www.bitsbytes.be

RINGRAZIAMENTI

Dicono che l'importante è il viaggio e non la destinazione. Dicono che la meta è solamente lo stimolo per partire. Guardando indietro mi accorgo che è veramente così. Le tappe della mia vita, per quanto importanti, rappresentano solamente una minima parte di me stesso. Ciò che veramente ha plasmato il mio Io, con tutti i pregi e i difetti, sono le esperienze maturate nel corso degli anni, le scelte che ho effettuato, le persone che ho incontrato e, ovviamente, gli errori che ho commesso.
Ora, a pochi passi da una meta molto ambita, mi trovo nella condizione di scegliere una tra le tante strade che si spalancano davanti ai miei occhi. La laurea è, in realtà, un altro punto di partenza per nuovi territori. E se da una parte ciò che è nuovo incute timore, dall'altra c'è la volontà e una bruciante curiosità di scoprire cosa riserva il futuro, con la consapevolezza che il viaggio continua…

Desidero quindi ringraziare tutti coloro che hanno preso parte a questo mio lungo percorso universitario, senza di loro queste che state leggendo sarebbero solamente mere pagine bianche.

Innanzitutto il professor Paolo Giorgini per avermi offerto la possibilità di intraprendere quest'esperienza e quindi tutte le persone che, in un modo o nell'altro, hanno dato il loro contributo a questa tesi: Michele Mussi, Sebastiano Perisi, Alex Vagin, Mattia Daldoss, Matteo Chini ed infine Enrico Nicoletti. Tutti quanti hanno dimostrato disponibilità e pazienza nei miei confronti e per questo mi sento in dovere di ringraziarli particolarmente.

Un grazie speciale è riservato ovviamente ai miei genitori, che mi hanno sempre appoggiato in questi anni, a mia sorella Anna e naturalmente ai miei meravigliosi coinquilini: Lorenzo e Luca, non sarò loro mai abbastanza grato per avermi offerto fedele compagnia durante tutto questo viaggio. Ringrazio anche tutti i miei amici della sala studio, con i quali ho passato momenti meravigliosi in questi anni universitari: Alessandro, Claudio, Marco, Jessica, Francesca e infine l'immenso Daniele Furlan, per cui nutro la più profonda stima. Spero di contare sulla loro splendida compagnia anche negli anni a venire.
L'ultimo ringraziamento, ma non in ordine d'importanza, va alla mia ragazza Veronica che, nonostante sia entrata nella mia vita da poco tempo, è riuscita a donarmi nuova linfa per proseguire il percorso. E' risaputo, infatti, che un viaggio in compagnia è molto più piacevole rispetto a quello solitario.

Ci sarebbero ancora altre persone importanti da ringraziare, ma preferisco farlo in sede separata… Sono sicuro che loro preferirebbero così.