
**Universitá degli Studi di Padova**

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

# Componenti software per l'integrazione di telecamere IP in un sistema di home automation

**Relatore**
Ch.mo prof. Carlo Ferrari

**Correlatore**
Ing. Alessandro Marcassa

**Laureando**
Alessandro Costa
matr. 603743-IF

**Anno Accademico 2011-2012**

# Introduzione

QUANTO RIPORTATO IN QUESTA TESI è il risultato dello stage svolto da me presso l'azienda Master Srl Divisione Elettrica di Este (PD) dal novembre 2010 all'aprile 2011. Questa società, che vanta un'esperienza ventennale nella produzione di sistemi e componenti per le installazioni elettriche di bassa tensione, ha iniziato circa cinque anni fa lo sviluppo di un proprio sistema di domotica, chiamato UNA, che da più di due anni è arrivato sul mercato.

I suoi punti di forza sono l'estrema facilità di utilizzo dei software di progettazione, programmazione e gestione dell'impianto e l'apertura verso l'integrazione di sistemi di terze parti. Questo secondo aspetto si collega al lavoro da me svolto che è stato quello di progettare, implementare e testare una serie di componenti software per permettere una prima forma di integrazione con telecamere IP, che garantisse la visualizzazione del video e un livello base di interazione con gli altri elementi che compongono un impianto.

Per il lancio sul mercato del sistema UNA, Master si è appoggiata alla rete commerciale già ben sviluppata nel ramo dei distributori e installatori di impianti elettrici. La maggior parte di essi, tuttavia, ha da poco approcciato il mondo della domotica e non ha quindi ancora acquisito appieno la mentalità adatta a gestire impianti così complessi. Per questo mi è stato chiesto, nell'ottica di mantenere la filosofia base che ha guidato lo sviluppo dell'intero prodotto, di orientare il mio lavoro più verso la semplicità di installazione e la facilità di configurazione e meno verso la ricerca della raffinatezza tecnologica.

Dopo un periodo di test, il software da me sviluppato è stato integrato negli applicativi e rilasciato ai clienti dell'azienda nei successivi aggiornamenti.

## Fasi della tesi

La fase iniziale del mio tirocinio è stata dedicata allo studio approfondito dell'architettura software del sistema UNA, per capire che tipo di struttura logica avrei dovuto adottare in modo da riuscire ad integrare correttamente negli applicativi già esistenti i componenti frutto del mio lavoro.

Oltre a questo, si è resa necessaria una ricerca bibliografica e tecnica sul tema dello streaming video e sulle modalità di funzionamento dei principali modelli di telecamere IP in commercio. Lo scopo era quello di individuare, in assenza di uno standard ufficiale, una modalità di interfacciamento con questo tipo di dispositivi che permettesse la più ampia compatibilità, mantenendo una complessità di implementazione accettabile.

La seconda fase è stata dedicata alla progettazione e allo sviluppo. Ha rappresentato la parte più corposa del mio lavoro e ha dato come risultati due diverse versioni del software. La prima, più semplice, permette soltanto la visualizzazione del video, mentre la seconda, oltre ad una profonda ottimizzazione di quanto già fatto, introduce l'interazione con gli altri elementi del sistema.

## Struttura della tesi

La tesi si sviluppa in cinque capitoli. Nel capitolo 1 introduce il tema dello streaming video, per capire quali sono le tecnologie attualmente disponibili e le motivazioni dietro alla scelta dell'implementazione di una particolare codifica video. Il capitolo 2 presenta il tema della domotica e introduce il sistema UNA dal punto di vista degli elementi fisici che lo compongono. Il capitolo 3 descrive a fondo la struttura del software che esso integra, in modo da chiarire i motivi delle scelte di implementazione prese durante il mio lavoro. Il capitolo 4 tratta a fondo il lavoro da me svolto, scendendo nei dettagli del software da me implementato. Il capitolo 5 trae le conclusioni e delinea i possibili sviluppi futuri.

# Indice

# Elenco delle figure

# Capitolo 1

# Lo streaming video

Lo streaming di contenuti multimediali è una funzione che consiste nell'invio da una o più sorgenti ad una o più destinazioni di un flusso di dati audio/video che viene riprodotto man mano che arriva a destinazione. La comunicazione può avvenire attraverso diverse tipologie di infrastruttura, come ad esempio l'etere, nel caso dei canali TV o radio, o un cavo coassiale nel caso di un impianto CCTV. Il contesto su cui ci si vuole focalizzare, per quanto riguarda questa tesi, è lo streaming video, in particolare su rete TCP/IP, sia a livello locale che via Internet.

Nel mondo dei calcolatori questo genere di applicazioni ha destato attenzione sin dai primi anni di sviluppo, nonostante le scarse capacità elaborative dell'epoca non consentissero di ottenere risultati pratici degni di nota. L'interesse si è concretizzato nei primi anni Ottanta, quando la potenza di calcolo dei PC ha raggiunto un livello tale da permettere di processare contenuti multimediali, ed è esploso negli anni Novanta e Duemila, a causa soprattutto della diffusione a macchia d'olio di Internet, divenuto accessibile anche agli utenti consumer, e della crescita esponenziale delle capacità di elaborazione dei microprocessori.

Al giorno d'oggi vi sono numerose applicazioni che fanno uso dello streaming video, tra le quali possiamo citare la videoconferenza, ossia la comunicazione simultanea attraverso un'infrastruttura comune tra individui situati in luoghi anche molto distanti tra loro, la videosorveglianza, ossia il controllo in locale o in remoto di un'area attraverso l'uso di telecamere, e, in maniera minore, l'online gaming. Ormai quest'attività è diventata molto familiare a chiun-

que abbia un minimo di conoscenze informatiche, grazie alla diffusione di siti web, come Youtube e Megavideo, dedicati esclusivamente al caricamento e alla visualizzazione di filmati da parte degli utenti.

É possibile distinguere due diverse modalità di implementazione di questa tecnica, pensate per diverse tipologie di contenuti:

- *True streaming*, nella quale il contenuto viene ricevuto dal client e riprodotto senza essere memorizzato nell'hard disk. Su questo sistema si basa lo *streaming live*, usato dalle web TV per la visualizzazione in diretta di canali televisivi[1] o di eventi importanti[2], ma anche in altre applicazioni come la videosorveglianza e la videoconferenza.

- *Progressive streaming*, detto anche *progressive download*, nella quale il contenuto richiesto si trova sul server sotto forma di file che viene progressivamente scaricato dal client e memorizzato nell'hard disk. La riproduzione avviene in parallelo al salvataggio e può iniziare immediatamente dopo la ricezione dei primi pacchetti, contenenti i metadati, ossia le informazioni utili al software di turno per poter avviare la visualizzazione anche senza che il file sia completo. Questo sistema si usa nelle piattaforme di video *on demand*, come i siti web nominati in precedenza, e la differenza sostanziale rispetto al true streaming risiede nella possibilità di visionare lo stesso contenuto un numero teoricamente infinito di volte finchè questo risiede sul server e di posizionarsi a piacimento in modo da poter rivederne o saltarne una parte.

## 1.1 La compressione del video

La prima problematica da affrontare quando si ha a che fare con lo streaming video è l'elevato volume di dati richiesto per memorizzare i contenuti, che si traduce in necessità di spazio per la memorizzazione e di banda per la trasmis-

[1] La RAI ha lanciato da qualche tempo il sito www.rai.tv che permette, tra le altre cose, la visualizzazione in diretta delle sue reti.

[2] Come l'insediamento del presidente Obama, trasmesso in diretta sui principali siti di informazione statunitensi.

sione. Considerati i costi elevati di queste due risorse, soprattutto nei primi anni di sviluppo, si è cercato sin da subito di progettare degli algoritmi di codifica in grado di effettuare una compressione efficiente dei dati. Vi sono algoritmi, detti *lossless*, in grado di ricostruire esattamente il filmato originale partendo da quello compresso, e altri, detti *lossy*, in cui questo è impossibile ed il risultato che si ottiene è un video di qualità inferiore all'originale, tanto peggiore quanto maggiore è la compressione che si vuole ottenere. Nonostante possa sembrare una contraddizione, la maggior parte degli algoritmi attualmente più diffusi è di tipo lossy perchè riescono ad utilizzare una quantità inferiore di dati rispetto ai lossless, tale da giustificare qualche piccola imperfezione nella visualizzazione.

La questione della codifica è talmente cruciale che due gruppi di lavoro: il *Video Coding Experts Group (VCEG)* e il *Moving Picture Expert Group (MPEG)*, sono nati, rispettivamente nel dicembre del 1984 e nel maggio del 1988, con lo scopo standardizzare questa procedura. Il primo standard per la codifica di video digitale effettivamente utilizzato è stato *H.261*, rilasciato da VCEG alla fine del 1990, a cui ha risposto MPEG, nel 1993, con l'omonimo *MPEG1* [1]. Per raggiungere gli elevati requisiti di qualità richiesti, queste tecniche utilizzano una codifica *interframe*, che tiene conto cioè della correlazione che esiste tra i fotogrammi, in quanto una semplice codifica *intraframe*, ossia che comprime i singoli fotogrammi separatamente, non avrebbe permesso di centrare l'obiettivo.

Lo standard attualmente più diffuso, invece, è stato sviluppato attraverso un lavoro congiunto delle due organizzazioni. Il suo nome è *H.264* secondo la nomenclatura di VCEG e *MPEG4 part 10* secondo quella di MPEG. Mentre le altre due tecniche citate in precedenza si limitano a lavorare a livello di pixel, questo algoritmo fa parte della cosiddetta *seconda generazione*. Questo approccio, di cui si era già discusso da tempo evidenziandone l'efficacia [2], introduce tecniche di rappresentazione più avanzate, che tengono in considerazione le modalità con cui l'occhio umano vede. L'intera suite MPEG4 rientra rientra in questa categoria, in quanto adotta un criterio *object-based*, cioè cerca di individuare all'interno della scena del filmato quali sono gli oggetti in movimento [3] [4].

Sebbene in questi anni siano stati fatti enormi passi in avanti per quanto riguarda la codifica video, queste due organizzazioni sono tutt'oggi costantemente al lavoro per cercare di sviluppare nuovi standard sempre più efficienti.

## 1.2 I vincoli temporali

Il secondo aspetto critico da tenere in considerazione consiste nei vincoli temporali che la riproduzione di un video comporta. Perchè la visualizzazione risulti fluida è necessario che il flusso di dati sia continuo e costante. Tuttavia il protocollo IP, sul quale si regge l'intera struttura di Internet, è di tipo best effort e non da garanzie sulla tempistica della consegna dei pacchetti di dati al destinatario. Questo significa che, soprattutto in reti complesse e di grandi dimensioni, il tempo di ricezione è strettamente legato alla congestione della rete stessa e può subire ritardi anche consistenti, compresa la possibilità che i dati non arrivino mai.

Per conciliare questi due aspetti apparentemente incompatibili sono stati sviluppati nel corso degli anni diversi protocolli di rete a livello applicazione[3] in grado di instaurare e supportare una comunicazione real-time in queste condizioni. Nessuno di essi è stato standardizzato ufficialmente da un ente preposto, bensì, come è successo per la suite TCP/IP, sono diventati standard de facto grazie alla loro ampia diffusione. Vediamo ora quali sono i più importanti e qual'è la loro funzione:

- *RTP (Real-time Transmission Protocol)*, presentato per la prima volta nel RFC 1889 [5] e aggiornato nel RFC 3550 [6], che si utilizza per l'effettivo trasporto dei dati. Solitamente si appoggia al protocollo di trasporto UDP, ma può utilizzarne anche altri.

- *RTCP (Real-Time Control Protocol)*, descritto nel RFC 3550 [6], viene utilizzato in combinazione con RTP e il suo scopo è quello di monitorare la qualità della trasmissione e tener traccia dei partecipanti ad una sessione di streaming.

[3] `http://it.wikipedia.org/wiki/Suite_di_protocolli_Internet`

- *RTSP (Real-Time Streaming Protocol)*, definito nel RFC 2326 [7], è utilizzato per instaurare la connessione al server di streaming e per controllare il flusso video. Ciò che fa come prima cosa è chiedere un descrittore del contenuto multimediale desiderato, dal quale è possibile ricavare il tipo di codifica video da utilizzare per la visualizzazione e le modalità di gestione della comunicazione. Sotto questo punto di vista può essere paragonato ad una sorta di "telecomando virtuale" in quanto impartisce al server dei comandi, tra quelli supportati indicati nel descrittore, che permettono di avviare, bloccare, far avanzare o arretrare l'esecuzione.

- *SDP (Session Description Protocol)*, definito nel RFC 4566 [8], delinea il formato dei parametri di inizializzazione di una sessione di streaming. Anche il descrittore del flusso video citato nel paragrafo precedente è strutturato secondo le specifiche indicate da questo protocollo.

É opportuno precisare che, sebbene questi siano i protocolli in assoluto più diffusi per la gestione di flussi video in rete, essi non rappresentano l'unica soluzione possibile. Ad esempio è possibile utilizzare anche HTTP per l'invio dei dati da server a client, come verrà accennato in seguito.

## 1.3 Il settore della videosorveglianza

Tra quelli nominati in precedenza il settore più affine a quanto trattato in questa tesi è quello della videosorveglianza. Nel 1996, anno della commercializzazione della prima telecamera IP da parte di Axis Communications, è iniziata una progressiva migrazione da sistemi di videosorveglianza analogici basati su cavo coassiale a sistemi più avanzati in cui l'interconnessione dei componenti di acquisizione e di visualizzazione si realizza attraverso una classica rete LAN (Ethernet o WiFi).

É evidente come un mutamento di questo genere renda molto più semplice l'interfacciamento con un PC o un qualsiasi sistema di terze parti in grado di comunicare in questo modo. Questo passaggio ha subito un'accelerazione negli ultimi due anni, da quando le aziende più importanti del settore hanno

iniziato a muoversi verso la stesura di uno standard. L'intenzione si è manifestata attraverso la nascita quasi contemporanea di due organizzazioni, ONVIF[4] e PSIA[5], che hanno formulato ciascuna le proprie specifiche riguardanti la gestione della videosorveglianza IP.

L'assenza di uno standard, infatti, ha portato ciascun produttore a sviluppare il proprio sistema chiuso, con modalità di comunicazione tra i componenti e di interfacciamento con l'esterno personalizzate, che rendono molto complessa l'interazione tra piattaforme diverse, vincolando di fatto il consumatore ad un particolare marchio al momento dell'acquisto. É per questo che tra gli installatori vi è ancora una certa diffidenza verso queste novità, alle quali spesso preferiscono tuttora la vecchia ma ben conosciuta tecnologia analogica.

La fiera *Sicurezza 2010* tenutasi nei giorni 17-19 novembre 2010 a Milano, alla quale ho partecipato, è la fiera più importante in quest'ambito e ha dato un'utile indicazione sull'interesse che questi ultimi avvenimenti hanno suscitato nelle aziende del settore. É emersa una forte attenzione verso questi due standard e svariati produttori hanno presentato i loro primi prodotti certificati. Il fatto che tutti i prodotti esposti avessero il marchio ONVIF lascia intuire una forte orientamento del mercato per questo standard, a discapito del rivale PSIA.

Nonostante entrambe le organizzazioni abbiano definito la prima versione delle loro specifiche ormai da più di un anno, i primi prodotti certificati sono stati lanciati soltanto da qualche mese. Questa considerazione, unita al fatto che entrambi i gruppi richiedono l'iscrizione e il pagamento di una quota annuale per poter avere accesso alle specifiche e agli strumenti di certificazione, mi ha convinto a non considerare queste due tecnologie nello sviluppo di questa tesi.

Tralasciando gli standard sopracitati, sebbene ogni azienda del settore abbia sviluppato un sistema personalizzato di gestione e interfacciamento con i propri prodotti, lo streaming video nella stragrande maggioranza delle telecamere IP viene implementato secondo due modalità:

[4] Open Network Video Interface Group, `http://www.onvif.org/`
[5] Physical Security Interoperability Alliance, `http://www.psialliance.org/`

- *Motion-JPEG over HTTP:* il Motion-JPEG, di cui non ho parlato finora, è una semplice codifica video di tipo intraframe che comprime separatamente ogni fotogramma sotto forma di immagine JPEG. Le rappresentazioni dei singoli frame vengono poi concatenate e inviate al client utilizzando il protocollo HTTP. Sebbene non sia molto performante, questa modalità è molto diffusa per la semplicità di implementazione, sia nei dispositivi di acquisizione che in quelli di visualizzazione.

- *H.264 over RTP/RTSP:* questa modalità, invece, utilizza la codifica standard e i protocolli di streaming citati in precedenza. Specularmente alla prima, l'aumento di qualità nella visualizzazione si paga con una maggiore difficoltà di implementazione.

Alla luce di quanto visto, la mia scelta riguardo l'implementazione del protocollo di streaming video è ricaduta sulla prima modalità, essenzialmente per due motivi:

- L'applicazione che mi è stato chiesto di sviluppare non richiede una risoluzione elevata, in quanto non c'è necessità di elaborazione delle immagini. A questi livelli la differenza di qualità tra le due codifiche video è trascurabile.

- L'obiettivo principale del mio lavoro era di ottenere un prodotto software che fosse semplice ma immediatamente integrabile nel sistema UNA. La minore complessità implementativa mi ha permesso di concentrarmi più sugli altri aspetti del progetto, in modo da avere un risultato più solido.

Dopo questa breve presentazione dello streaming video, nel capitolo successivo viene introdotto il contesto all'interno del quale esso si inserisce per quanto riguarda lo stage da me svolto, ossia la domotica e, in particolare, il sistema UNA.

## CAPITOLO 2

# La domotica e il sistema UNA

La domotica, secondo Wikipedia, è *"la scienza interdisciplinare che si occupa dello studio delle tecnologie atte a migliorare la qualità della vita nella casa e più in generale negli ambienti antropizzati"*.

Lo scopo basilare di un impianto domotico è quello di interconnettere una serie di dispositivi all'interno della casa (come luci, prese elettriche, termoregolazione, tapparelle, etc.) realizzando un unico sistema intelligente in grado di monitornarne lo stato ed agire su di essi rispondendo sia agli input dati dall'inquilino ma, soprattutto, anche in maniera autonoma, in base ad una serie di regole di comportamento definite dall'installatore o dall'inquilino stesso. A questo va ad aggiungersi la possibilità di avere nuove modalità di controllo ben più evolute dei semplici comandi a muro, come touchscreen ad incasso, tablet, smartphone e computer. Alcune di esse possono funzionare anche da remoto, permettendo un monitoraggio dell'abitazione anche quando l'utente si trova da tutt'altra parte.

I vantaggi che derivano dall'uso della domotica sono molteplici. Quello più immediato è la maggiore facilità e velocità di controllo della casa, ma quello a mio avviso più importante è l'ottimizzazione dei consumi dovuta, appunto, al fatto che l'impianto agisce come un'entità unica.

Questo settore fino a pochi anni fa risultava poco conosciuto e riscuoteva un interesse pressochè marginale al momento della progettazione di abitazioni, in quanto veniva visto principalmente come qualcosa di superfluo e costoso, adatto soltanto a clienti facoltosi.

Ultimamente, però, vi è stata un'apertura sempre maggiore da parte dei co-

struttori di immobili e degli installatori, spinti dalla necessità di fornire soluzioni sempre più innovative ai propri acquirenti per poter competere nel mercato edilizio odierno che risente di una forte stagnazione. Si sono dimostrati interessati anche i clienti stessi, sempre più sensibili a tematiche come quelle del risparmio energetico e della produzione domestica da fonti alternative, per far fronte al continuo aumento del costo dell'energia legato alla difficoltà sempre maggiore nel reperire combustibili fossili.

Attualmente il mercato della domotica in Italia è abbastanza frammentato. Al suo interno possiamo trovare colossi del settore elettrico, come BTicino, Vimar e Gewiss, che forniscono sistemi completi. Alcuni di essi possono vantare ormai un'esperienza pluriennale nel settore, basti pensare che il sistema MyHome di BTicino è sul mercato ormai da più di dieci anni. Al loro fianco, però, sono presenti diverse aziende di medie dimensioni, tra le quali può inserirsi la stessa Master, e un insieme di piccole realtà nate dalla passione e dall'interesse dei loro fondatori verso questo settore.

Dopo questo breve excursus sul mondo della domotica in generale, vediamo ora una descrizione del prodotto con il quale Master sta cercando di ritagliarsi la propria fetta di clientela.

## 2.1 La struttura del sistema UNA

I sistemi di domotica attualmente più diffusi sul mercato hanno una struttura a bus che collega tutti i componenti, come sensori, comandi utente e moduli attuatori, ad una unità di controllo, che è l'unico dispositivo intelligente nell'impianto. Il canale di comunicazione, quindi, ne diventa contemporaneamente il punto nevralgico, perchè ogni dispositivo risulterebbe incapace di agire autonomamente senza comunicare con la centrale, e il punto debole, perchè se dovesse rompersi renderebbe inattiva una parte del sistema, o addirittura tutto.

Il sistema UNA, di cui è possibile vedere uno schema in Figura 2.1, adotta un criterio per certi versi simile, ma con alcune differenze. L'approccio a bus rimane, ma la logica di funzionamento dell'impianto è distribuita tra i vari componenti. Questo permette di ridurre sensibilmente il traffico di informa-

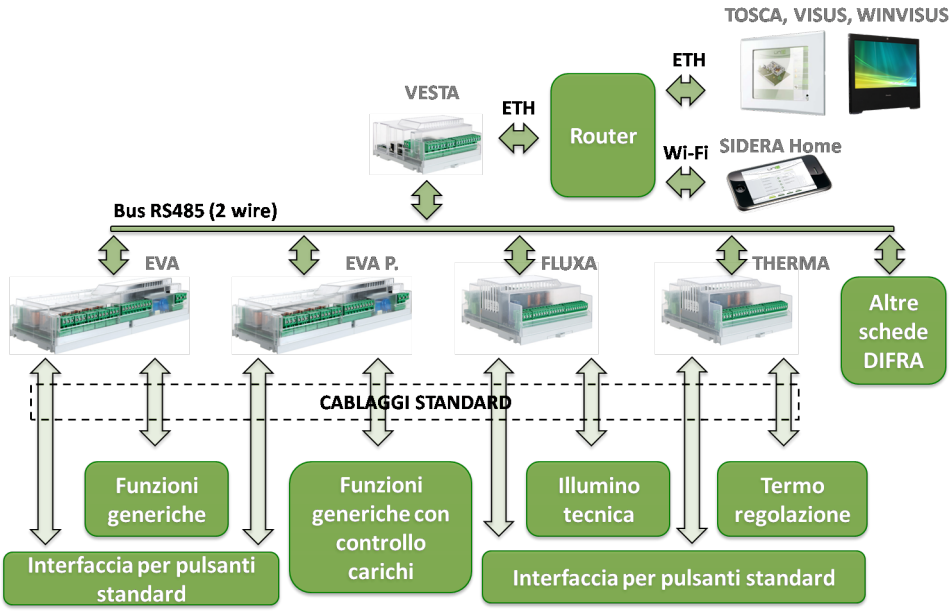


Figura 2.1: Schema della struttura del sistema UNA

zioni che transitano nel canale di comunicazione, tanto da renderlo persino superfluo nelle installazioni di minore dimensione.

Alla base del sistema ci sono una serie di schede, definite “Slave”, diversificate in base alle funzionalità che implementano. Esse sono accomunate dal fatto di possedere una serie di ingressi a contatto pulito (da due a sedici), ossia che richiedono di chiudere un circuito tra un morsetto relativo all’ingresso e un morsetto comune per inviare l’impulso di attivazione alla scheda, e una serie di relè (da due a otto), che rappresentano le attuazioni comandabili dal sistema. Il cuore è un microcontrollore programmabile che permette di definire il comportamento ingressi-relè in base ad una serie di funzioni disponibili, legate alla tipologia di scheda. In ognuna di esse è possibile associare due funzioni ad ogni ingresso, una per l’impulso semplice ed una per l’impulso prolungato. Il numero di secondi che devono trascorrere perchè un impulso venga considerato prolungato è definibile in fase di programmazione.

Il vantaggio principale che deriva da questo tipo di approccio è quello di dare la possibilità all’installatore di montare i comandi che più gli piacciono, senza quei vincoli che altri prodotti dello stesso tipo impongono. Molti disposi-

tivi, inoltre, come sensori d'allarme, sistemi di videosorveglianza o impianti di condizionamento, implementano a loro volta ingressi a contatto pulito e uscite a relè, il che li rende immediatamente interfacciabili col sistema UNA.

Nonostante siano dotate di una porta seriale RS485, queste schede Slave non sono in grado di comunicare tra di loro, tuttavia ciascuna di esse permette di realizzare un impianto in scala ridotta.

Vediamo ora, nell'elenco sottostante, quali sono e che ruolo svolgono all'interno del sistema UNA. Tutti i modelli, salvo alcuni esplicitamente indicati, sono predisposti sia per il montaggio su quadro elettrico che a parete.

- *Scheda Eva*: è una scheda general-purpose dotata di sedici ingressi a contatto pulito e otto relè. Al suo interno, inoltre, è presente un orologio astronomico per permettere di impostare temporizzazioni legate agli orari di alba e tramonto sull'abilitazione, disabilitazione e attivazione delle funzioni. Può essere alimentata sia direttamente a 230V AC, sia a 12V DC.

- *Scheda Eva Power*: è a tutti gli effetti una scheda Eva, con in aggiunta la possibilità di misurare la potenza assorbita da ogni singolo relè, fino ad un massimo di 2,5 KW per singolo canale e 6 KW per scheda. Permette, inoltre, di memorizzare fino a 12 configurazioni di stacco carichi, per ognuna delle quali è possibile definire un ordine con il quale i componenti collegati vengono disattivati se il consumo totale supera la soglia indicata, specificando inoltre i giorni della settimana in cui dev'essere attiva.

- *Scheda MiniEva*: tale e quale alla scheda Eva dal punto di vista della programmazione, ma più piccola. Possiede, infatti, otto ingressi e quattro relè.

- *Scheda MicroEva*: è un dispositivo molto piccolo (meno di cinque centimetri di lato), dotato di due ingressi e due relè. È stato pensato per i casi in cui lo spazio scarseggia, come, ad esempio, le ristrutturazioni. Non integra l'orologio astronomico e può essere alimentata solamente a 12V DC.

- *Scheda Fluxa*: è la scheda specifica per la gestione dell'illuminazione. È dotata di otto ingressi a contatto pulito, quattro ingressi analogici ai quali possono essere collegati altrettanti sensori di luminosità, quattro relè e quattro uscite analogiche 0-10V in grado di pilotare luci dimmerabili.

- *Scheda Therma*: è la scheda specifica per la termoregolazione. Ha le stesse caratteristiche di Fluxa, tranne per la mancanza delle quattro uscite analogiche e per la taratura degli ingressi analogici, pensata per il collegamento di quattro sonde di temperatura.

- *Lettore RFID Difra*: lo scopo di questo dispositivo è la gestione del controllo accessi. Possiede un lettore di badge RFID, due ingressi a contatto pulito e due relè: il primo da utilizzare per l'elettroserratura di apertura del varco, il secondo per un'eventuale luce di cortesia. La sua collocazione, ovviamente, non è in quadro elettrico ma incassata a muro in prossimità del punto d'accesso.

È chiaro che, se si riuscisse a suddividere il sistema che si vuole ottenere in porzioni indipendenti tra di loro, con il solo ausilio delle schede slave, nel giusto numero e della giusta tipologia, si otterrebbe un impianto domotico di base pienamente funzionante. La maggior parte dei clienti, però, richiede funzionalità avanzate o, almeno, comportamenti che debbano agire contemporaneamente su elementi collegati a schede diverse. Qui entra in gioco un secondo livello di controllo, costruito al di sopra delle funzionalità appena descritte, che ruota attorno ad un unico dispositivo: la scheda Vesta.

Questa scheda non svolge funzioni di automazione, ma rappresenta il centro nevralgico di ciascun impianto evoluto (ad ogni installazione, infatti, ne corrisponde al massimo una). Dal punto di vista hardware è a tutti gli effetti un microcomputer con processore ARM 32 bit a 400 Mhz e 128 MB di RAM. Possiede tre linee seriali RS485, una porta Ethernet, un uscita audio a due canali, uno slot per schede SD fino a 4 GB, un'interfaccia per sensore a infrarossi e una per il modulo radio 434. Gli ultimi due componenti elencati permettono l'utilizzo di classici telecomandi a infrarossi, come quello della TV, o radiocomandi, come quello del cancello di casa, per controllare l'impianto.

Realizzando un bus di comunicazione seriale RS485 tra una delle sue linee e le altre schede slave, quello che prima era un insieme di dispositivi di automazione a sé stanti diventa un corpo unico in grado di esprimere il massimo delle potenzialità.

Vesta, infatti, monitora costantemente lo stato degli elementi connessi al sistema e in base a questo esegue una serie di operazioni su di essi, seguendo le regole impostate dall'installatore o dall'utente nei cosiddetti "Scenari". Questo aspetto verrà approfondito nel capitolo successivo.

La presenza di una porta Ethernet su questo dispositivo è di enorme importanza per due motivi: innanzitutto perchè rende Vesta un gateway di comunicazione che consente, una volta connessa ad una LAN, di mettere in comunicazione un qualsiasi nodo della rete con il sistema domotico, ma anche perchè abilita l'utilizzo delle interfacce evolute.

Il sistema UNA ne mette a disposizione tre, due accessibili dalla rete locale e una accessibile da remoto.

- *Sidera Home*: un'interfaccia web locale che risiede direttamente su Vesta ed è raggiungibile da qualsiasi dispositivo connesso alla LAN dotato di un browser abbastanza aggiornato, come può essere uno smartphone, un tablet, un media center o un PC. Permette di controllare lo stato e comandare tutti gli elementi del sistema e leggere i consumi, nel caso siano state installate una o più schede Eva Power. È disponibile in due vesti grafiche diverse: la versione mobile, pensata per essere utilizzata da dispositivi di ridotte dimensioni, e la versione estesa, pensata per schermi più grandi.

- *Sidera Web*: un'interfaccia web che ricalca la versione estesa di Sidera Home, ma permette il controllo da remoto, attraverso il portale del sistema UNA[1]. Per poterne usufruire l'utente dovrà abilitarla nel menu di configurazione di Vesta e accedere al sito web con le credenziali che gli verranno fornite dall'installatore. Il suo utilizzo non comporta alcun costo aggiuntivo.

[1] `http://sidera.domologica.it`

- *Visus*: un software progettato appositamente per il controllo dell'impianto, scritto in C++, dotato di una grafica più evoluta rispetto alle interfacce web e di funzionalità aggiuntive, come la possibilità di creare nuovi scenari attraverso l'editor integrato e la visualizzazione sotto forma di grafico dello storico dei consumi. Comporta un costo aggiuntivo per l'utente ed si può ottenere in tre modi diversi:

  - *WinVisus*: un'applicazione installabile nel proprio PC con sistema operativo Windows.
  - *Visus USB*: una chiavetta USB autoinstallante contenente una distribuzione personalizzata del sistema GNU/Linux, basata su Ubuntu, modificata per eseguire esclusivamente il nostro software di controllo e pensata per essere installata su comuni pc all-in-one acquistabili sul mercato.
  - *Tosca*: un touchscreen a incasso da 10" progettato e realizzato da Master, sul quale è installato lo stesso sistema operativo GNU/Linux precedentemente descritto.

Tutte e tre le interfacce appena elencate, così come il resto del sistema, vengono programmate dall'installatore che crea una serie di mappe della casa, solitamente utilizzando planimetrie o foto dell'abitazione, sulle quali dispone nel modo più opportuno gli elementi con i quali l'utente può interagire.

Tutto questo viene effettuato utilizzando Lapis, il software di progettazione del sistema UNA, che viene distribuito da Master esclusivamente agli installatori. Attraverso questo applicativo, che verrà presentato più in dettaglio nel capitolo 4, è possibile programmare le schede Slave e la scheda Vesta. Quest'ultima, poi, si occuperà di inviare le configurazioni agli eventuali software Visus collegati all'impianto.

La panoramica di tutti gli elementi hardware e software che compongono il sistema UNA è completa. Nel paragrafo successivo vengono presentate le novità introdotte nella sua struttura durante il mio stage.

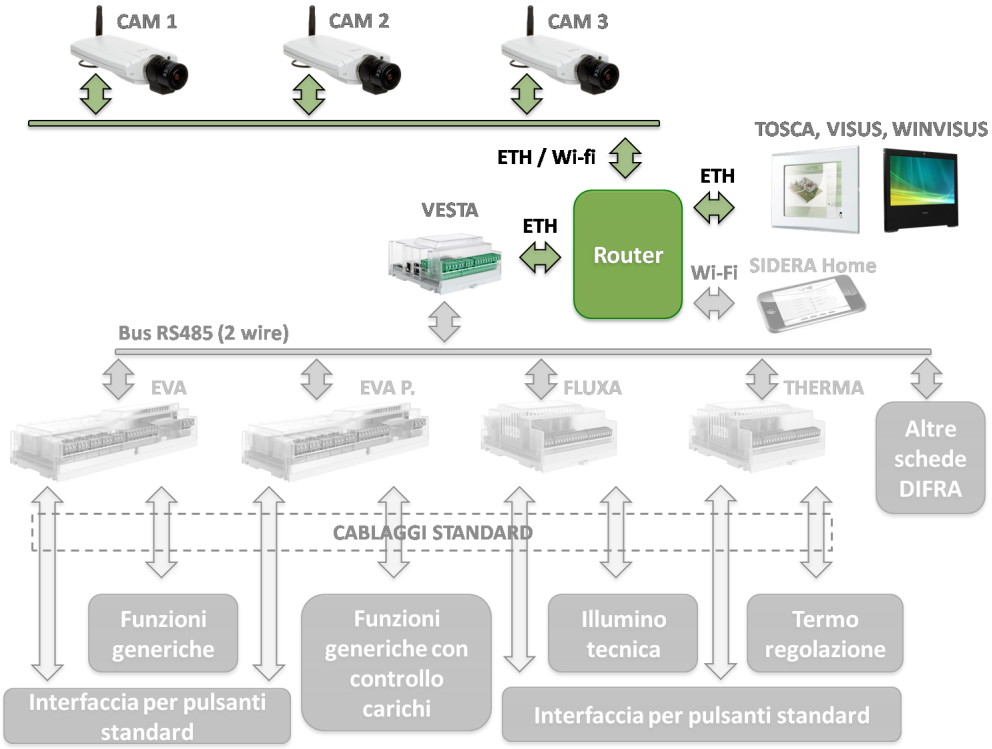


Figura 2.2: Elementi coinvolti nel mio lavoro di sviluppo

## 2.2 L'obiettivo da raggiungere

Come già accennato nell'introduzione, lo scopo del mio lavoro è l'integrazione delle telecamere IP all'interno del sistema. Per questa prima versione, si richiede che la visualizzazione del video avvenga soltanto in locale nelle tre versioni del software Visus.

Dovrà, inoltre, essere sviluppata una prima forma di interazione con gli scenari domotici, ossia fare in modo che uno o più eventi legati ad elementi dell'impianto possano comandare l'apertura del video. Dovrà anche essere possibile, viceversa, inviare dei comandi al sistema senza interrompere la visualizzazione del video stesso.

Per permettere l'integrazione le telecamere IP dovranno essere collegate alla stessa rete LAN dove è presente Vesta. Figura 2.2 da una rappresentazione della nuova struttura del sistema.

Il caso d'uso più esplicativo per descrivere questa funzionalità è quello del videocitofono "simulato": si posiziona una telecamera IP all'esterno in prossimità del cancelletto e si programma il sistema in modo che, quando una persona suona al campanello, appaia il relativo video sullo schermo dal quale, successivamente, sarà possibile aprire la serratura per far entrare l'ospite.

Le modifiche introdotte dal mio lavoro interessano tre elementi del sistema. Innanzitutto il software di progettazione Lapis, che dovrà permettere all'installatore di inserire le telecamere IP all'interno del progetto dell'impianto, configurarne i parametri necessari per la comunicazione e testarne il corretto funzionamento. Anche il software della scheda Vesta verrà aggiornato, in quanto sarà lei a memorizzare le impostazioni relative alle telecamere e ad occuparsi di attivare gli scenari ad esse associati. Per ultimo troviamo Visus, che dovrà integrare la visualizzazione vera e propria del video.

In questo capitolo il sistema UNA è stato descritto dal punto di vista degli elementi che lo compongono, del ruolo che essi hanno e di come interagiscono tra di loro. Nel capitolo successivo viene descritta in dettaglio la struttura software del sistema sulla quale si basa il risultato del mio lavoro.

# CAPITOLO 3

# Descrizione della struttura software

In questo capitolo viene descritta la struttura logica degli applicativi sui quali ho lavorato durante il mio stage, cioè Lapis, Visus e Vesta. Vengono esclusi volontariamente Sidera Home e Sidera Web in quanto non coinvolti.

Sono tutti e tre stati realizzati usando il linguaggio C++ e il framework Qt, che è stato scelto in quanto permette una facile portabilità tra sistemi operativi e piattaforme hardware diverse. Questo aspetto è cruciale per ottimizzare lo sviluppo in quanto Lapis viene rilasciato nelle versioni per Windows, Linux e Mac OS, Visus solo per i primi due sistemi operativi, mentre Vesta richiede di essere compilata per architettura ARM.

Questi tre software, pur avendo funzioni diverse e, di conseguenza, interfacce di utilizzo diverse, possiedono una struttura base comune sulla quale si regge il funzionamento avanzato dell'intero sistema UNA e comunicano tra di loro attraverso protocolli basati sullo stack TCP/IP. D'ora in poi userò la parola *sistema* per indicare una generica istanza di uno di questi tre applicativi presente in rete. Il paragrafo successivo presenta nel dettaglio la struttura di base di un sistema.

## 3.1 Struttura di un sistema

La struttura di un sistema è a plugin, cioè è formata da un insieme di componenti che vengono caricati a runtime da un Core. Con il termine componente, nel nostro caso, si intende una porzione di software che risiede in una libreria

separata e implementa una particolare funzionalità. La comunicazione tra di loro può avvenire in due modi:

- Attraverso un sistema di notifiche (o messaggi), gestite interamente dal Core secondo un modello publish/subscribe, nel quale ogni componente dichiara al Core quali sono i tipi di notifiche che desidera ricevere. Durante il funzionamento dell'applicazione, appena un componente invierà un messaggio al Core, questo si occuperà di inoltrarlo, in ordine, a tutti coloro che si sono registrati per ricevere quel determinato tipo di notifica. Questo meccanismo nasconde al mittente quali siano gli effettivi destinatari dei suoi messaggi, permettendo una maggiore flessibilità, in quanto i collegamenti possono cambiare durante l'esecuzione del programma, e una maggiore facilità di gestione del codice, in quanto, dovendosi preoccupare esclusivamente dell'interazione con il Core, è più facile aggiungere o modificare i componenti stessi.

- Richiamando direttamente un componente dall'interno di un altro. Questo approccio ha lo svantaggio di creare una dipendenza diretta tra i due componenti, anche a livello di librerie, per cui il secondo non verrebbe nemmeno caricato dal Core se dovesse mancare il primo, ma ha il grosso vantaggio di semplificare l'implementazione. Questo tipo di soluzione viene solitamente utilizzata quando esiste un forte legame tra i due componenti, oppure quando c'è bisogno di un'interazione diretta che renderebbe troppo dispendioso in termini di tempo il passaggio per il Core.

La struttura comune di un sistema si fonda su tre componenti fondamentali, senza i quali l'intero applicativo, sia esso Lapis, Visus o Vesta, non sarebbe in grado di funzionare. Essi sono il "modello degli elementi", il "gestore dei comandi" e il "gestore RPC", che verranno presentati in dettaglio nei paragrafi successivi.

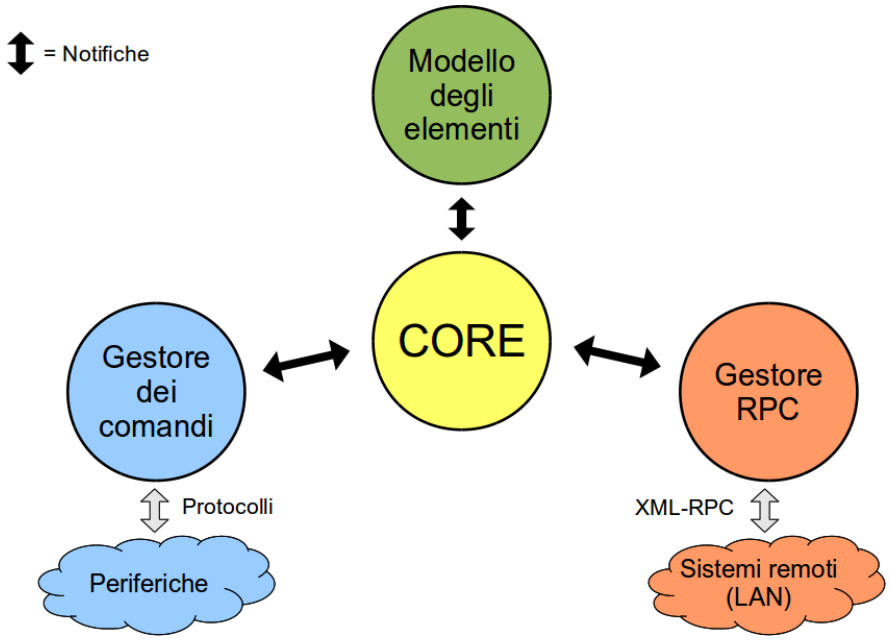


Figura 3.1: Componenti fondamentali nella struttura di un sistema

## 3.2 Il modello degli elementi

Il modello degli elementi è il componente che contiene la struttura logica dell'intero impianto domotico. Ogni sistema connesso al medesimo impianto contiene una replica del modello associato, che si mantiene sincronizzata con le altre attraverso un meccanismo che verrà descritto successivamente. Un elemento è una qualsiasi entità collegata al sistema, sia essa reale, come una scheda Slave o un punto luce, o virtuale, come un pulsante virtuale comandabile esclusivamente da Visus. Essi sono suddivisi in classi, ognuna delle quali definisce gli attributi che determinano lo stato dell'elemento e la sua ontologia, cioè l'insieme di regole che descrive le modalità con cui il sistema può interagire con un elemento di quella particolare classe. Vi sono due tipi di regole:

- le *condizioni* che possono essere valutate sullo stato dell'elemento
- le *azioni* che possono essere applicate all'elemento stesso per modificarne lo stato

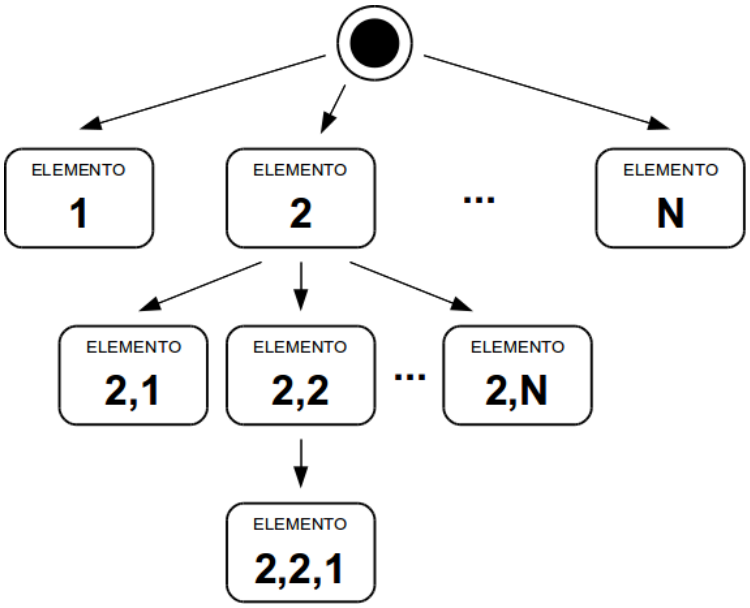


Figura 3.2: Albero degli elementi

Per chiarire quanto detto finora, prendiamo come esempio uno degli elementi più semplici, ossia un punto luce. Il suo stato è rappresentato da un valore booleano che indica se è acceso o spento, mentre l'ontologia racchiude le due condizioni "il punto luce è acceso" e "il punto luce è spento" e le tre azioni "accendi il punto luce", "spegni il punto luce" e "cambia stato al punto luce".

Gli elementi possono essere annidati definendo una gerarchia di tipo padre-figlio. Un esempio è dato dalla scheda EVA, nella quale ogni ingresso e ogni relè sono singoli elementi che hanno come padre l'elemento principale. Questa configurazione porta ad avere una struttura ad albero, visibile in Figura 3.2, nella quale ogni nodo, esclusa la radice, rappresenta un elemento del sistema. Ognuno di essi è identificato univocamente dal percorso che bisogna fare per raggiungerlo partendo dalla radice.

Un singolo elemento comunica con il resto del sistema attraverso dei punti di connessione denominati *porte*. Una porta è un oggetto con tre caratteristiche fondamentali:

- Contiene un valore il cui tipo (intero, floating point, booleano, stringa, etc.) viene specificato al momento della creazione della porta e non può

essere più modificato.

- Può essere di input o di output. La prima categoria identifica le porte che devono essere usate per inviare informazioni all'elemento, la seconda, invece, quelle che devono essere usate per ricevere informazioni dall'elemento.

- Può essere pubblica, cioè accessibile da tutti gli elementi del sistema, o privata, cioè accessibile soltanto dagli elementi genitori nella gerarchia ad albero.

In base alla logica di funzionamento, ogni elemento espone all'esterno una serie di porte. Ogni volta che esso riceve da quelle di input informazioni dal sistema, le elabora, modificando il proprio stato interno, e aggiorna quelle di output per renderlo pubblico. Ritornando all'esempio del punto luce, un elemento di questa classe possiede due porte, una booleana di output che indica lo stato acceso o spento, e un'altra booleana di input che permette di impostare lo stato. In questo caso l'elaborazione interna è banale e consiste nel riportare il valore della seconda all'interno della prima.

Le regole contenute nelle ontologie degli elementi, che all'utente del sistema vengono presentate come frasi in linguaggio naturale, al momento dell'invio della programmazione vengono mappate sulle porte dell'elemento stesso. Le condizioni vengono tradotte in valutazioni di valori su porte di output, mentre le azioni in valori da inviare su porte di input.

Quanto detto finora interessa la struttura di un singolo elemento. Nel prossimo sottoparagrafo vengono descritte le modalità di interazione tra elementi diversi.

### 3.2.1 Elementi a comando diretto e indiretto

In precedenza si è citata la suaddivisione degli elementi in classi in base al loro funzionamento. Queste possono essere partizionate in due superclassi in base alla modalità con cui il sistema agisce per modificarne lo stato, che sono:

- Elementi a *comando diretto*

- Elementi a *comando indiretto*

Gli elementi a comando diretto rappresentano tutti i dispositivi con i quali il sistema è in grado di interfacciarsi esplicitamente per inviare comandi o leggerne lo stato. Attualmente fanno parte di questa categoria tutte le schede Slave, che comunicano con Vesta utilizzando un protocollo proprietario su bus seriale RS485, le telecamere IP e gli elementi virtuali, per i quali la logica di funzionamento è completamente simulata. Per ogni elemento appartenente a questa categoria viene specificato il *sistema proprietario*, un attributo che specifica l'unico sistema che ha diritto ad effettuare le operazioni necessarie all'aggiornamento dello stato o all'invio dei comandi che, nel caso di elementi associati a dispositivi fisici, comportano una comunicazione vera e propria, mentre, nel caso di elementi virtuali, comportano la simulazione della logica interna.

Gli elementi a comando indiretto rappresentano, invece, i restanti dispositivi, con i quali il sistema non è in grado di comunicare. La caratteristica principale è che il loro stato non è determinabile con assoluta certezza attraverso una richiesta diretta all'elemento stesso, ma dipende da quello di altri elementi.

Queste dipendenze tra stati di elementi diversi del sistema vengono inserite all'interno del modello creando delle *connessioni*. Una connessione è un collegamento logico tra una porta di output di un elemento e una di input di un altro, contenenti lo stesso tipo di dato, tale che quando il valore della porta di output cambia, questo viene riportato su quella di input, la cui elaborazione interna all'elemento destinatario potrebbe portare all'attivazione a catena di altre connessioni. Una connessione tra porte private di due elementi legati da una relazione padre-figlio prende il nome di *route*.

Il punto luce, per tornare all'esempio iniziale, è un classico esempio di un elemento a comando indiretto. Il suo stato acceso o spento non può essere comandato direttamente, bensì dipende da quello del relè al quale è fisicamente collegato. Per questo, perchè l'elemento punto luce sia utilizzabile correttamente, deve essere connesso all'interno del modello al corrispondente elemento relè della scheda Slave corretta.

La descrizione della struttura del modello degli elementi è completa, ma prima di passare a presentare gli altri due componenti fondamentali per il funzionamento del sistema è necessario capire come avviene la comunicazione tra di loro.

## 3.3 Le principali tipologie di notifiche

Il modello degli elementi, il gestore dei comandi e il gestore RPC comunicano tra di loro utilizzando la prima delle due modalità descritte nel paragrafo 3.1, ossia attraverso notifiche inviate al Core.

Le tipologie di notifica che vengono scambiate tra questi tre componenti sono due:

- Notifica *DeviceChangeRequest*: rappresenta una richiesta di cambiamento o di lettura di stato di un elemento a comando diretto. Solitamente viene originata all'interno del modello degli elementi da un comando esplicito effettuato dall'utente tramite una delle interfacce disponibili o dall'attivazione di uno scenario.

- Notifica *DeviceChanged*: rappresenta un cambiamento di stato di un elemento a comando diretto. Solitamente viene originata dal gestore dei comandi, nelle modalità indicate nel paragrafo successivo.

Entrambe queste due tipologie di notifica contengono al loro interno una struttura dati che dipende dal tipo di elemento coinvolto e specifica nel primo caso le richieste e nel secondo i cambiamenti di stato avvenuti.

Dopo questa breve spiegazione delle modalità di comunicazione interne ad un sistema, nei paragrafi successivi vengono descritti i componenti che gli permettono di interfacciarsi con l'esterno.

## 3.4 Il gestore dei comandi

Il gestore dei comandi è il componente che si occupa della comunicazione vera e propria con i dispositivi fisici associati nel modello ad elementi a comando

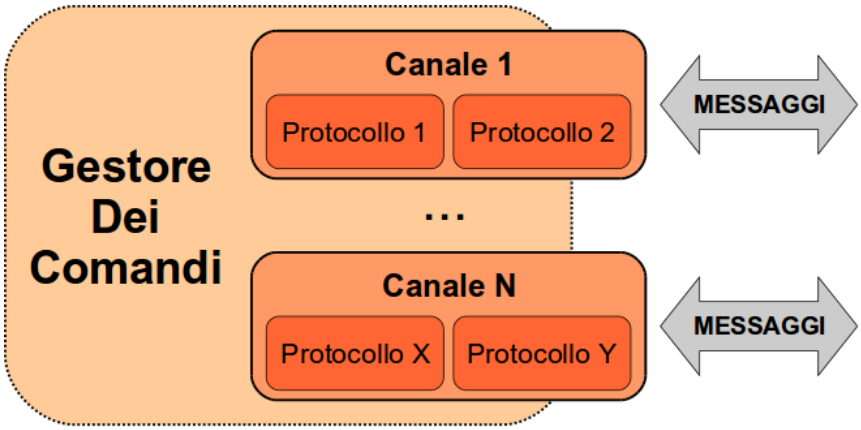


Figura 3.3: Struttura del gestore dei comandi

diretto. Ogni sistema comunica soltanto con quelli dei quali è indicato come proprietario.

Sebbene l'architettura software permetta a diversi sistemi che condividono lo stesso modello logico di spartirsi la proprietà degli elementi a comando diretto, allo stato attuale questo non avviene. È infatti Vesta, che ricordiamo essere una per impianto, il sistema proprietario di tutti gli elementi a comando diretto presenti, concentrando su di sé le comunicazioni con il resto dei dispositivi. I restanti sistemi vengono da essa sincronizzati attraverso il meccanismo descritto nel paragrafo successivo.

Come è possibile vedere nella Figura 3.3, il gestore dei comandi contiene al suo interno l'implementazione del funzionamento dei vari canali di comunicazione accessibili dal sistema. Ognuno di essi, poi, contiene un elenco di protocolli che possono essere utilizzati per comporre e interpretare i messaggi che vi transitano.

Il ciclo di funzionamento del gestore dei comandi è il seguente:

- Riceve dal Core una notifica di tipo DeviceChangeRequest. Se il sistema locale è il proprietario dell'elemento a comando diretto al quale la notifica fa riferimento, allora questa viene presa in carico per essere eseguita, altrimenti viene ignorata.

- Cerca il canale di comunicazione associato all'elemento in questione e vi accoda la richiesta.

- Per ogni richiesta, il canale di comunicazione individua il protocollo associato all'elemento e gli passa la notifica in modo che questo possa tradurla, secondo il formato corretto, nel messaggio che viene poi inviato al dispositivo fisico.

- La risposta al messaggio viene nuovamente passata al protocollo che la converte, secondo la sua logica interna, in una notifica DeviceChanged, in modo che gli altri componenti del sistema possano interpretare gli aggiornamenti di stato dell'elemento.

- La notifica DeviceChanged creata al passaggio precedente viene restituita al gestore dei comandi, che si occupa di inoltrarla al Core.

Attualmente l'unico canale di comunicazione implementato è quello seriale, che invia e riceve informazioni dalle linee RS485 della Vesta. È di tipo master-slave, ossia prevede che vi sia un unico dispositivo (nel nostro caso proprio Vesta) dal quale ha origine la comunicazione, di volta in volta diretta verso uno degli altri dispositivi, che risponderà secondo il protocollo. Inoltre è di tipo request-reply, ossia prevede che ad ogni richiesta debba obbligatoriamente seguire una risposta. L'unico protocollo correntemente implementato legato al canale seriale è il protocollo proprietario Una, che permette al sistema di comunicare con tutte le schede Slave.

## 3.5 Il gestore RPC

Il gestore RPC è il componente che si occupa della comunicazione tra sistemi diversi connessi allo stesso impianto. Essa avviene, come già anticipato nell'introduzione di questo capitolo, utilizzando lo stack TCP/IP.

Quando un sistema A necessita di inoltrare una notifica ad un sistema B, il gestore RPC del primo la converte in un formato XML, che viene inviato, attraverso una socket TCP, al gestore del secondo, il quale effettuerà la conversione inversa. La notifica risultante verrà poi consegnata al Core del sistema B, il quale si occuperà di consegnarla ai componenti adeguati.

Le notifiche che il gestore RPC inoltra sono di due tipi:

- Notifiche di tipo DeviceChangeRequest nelle quali il sistema locale non è il proprietario dell'elemento a comando diretto al quale la notifica fa riferimento, che vengono inoltrate proprio a quest'ultimo. Ogni comando impartito al sistema attraverso l'interfaccia del software Visus comporta la creazione da parte del modello degli elementi sottostante di una notifica di questo tipo, che viene inviata al sistema Vesta associato.

- Notifiche di tipo DeviceChanged che circolano nel sistema proprietario dell'elemento a comando diretto coinvolto. Queste vengono spedite a tutti gli altri sistemi collegati all'impianto in modo che possano sincronizzare lo stato del loro modello degli elementi per mantenerlo coerente.

Ora che anche la struttura software del sistema UNA è stata presentata, è il momento di descrivere il lavoro da me svolto per arrivare a soddisfare le esigenze dell'azienda. Il tutto verrà trattato nel capitolo successivo.

# CAPITOLO 4

# Descrizione del lavoro svolto

La fase di sviluppo è stata quella che ha assorbito la maggior parte del tempo del mio stage. Il risultato finale è stato un plugin che si integra nei tre software Vesta, Visus e Lapis e fornisce le funzionalità richieste. Ne ho sviluppato due versioni, che verranno presentate e confrontate nei paragrafi seguenti.

## 4.1 Prima versione

Nella prima versione del plugin ho deciso di implementare soltanto la parte di visualizzazione del video, tralasciando volontariamente l'integrazione con gli scenari, in modo da potermi concentrare maggiormente su due aspetti:

- La progettazione della struttura software, in modo che rispettasse i vincoli necessari all'integrazione con il sistema Una. Questo rappresenta un aspetto chiave nello sviluppo di un plugin, in quanto se i componenti da me sviluppati non fossero allineati a queste regole, gli applicativi non li riconoscerebbero e non sarebbero quindi in grado di caricarli ed eseguirli correttamente, rendendoli di fatto inutili.

- L'implementazione della parte di ricezione del flusso video Motion-JPEG, in modo da estrarre correttamente i frame video da visualizzare.

Conformemente all'architettura software presente, nel momento in cui si debba integrare un nuovo dispositivo, il primo passo da compiere è quello di rendere il modello degli elementi conscio dell'esistenza del dispositivo stesso.

Ciò si realizza estendendo secondo le proprie esigenze la classe base Element, implementando, se necessario:

- Gli attributi per la configurazione iniziale dell'elemento, ossia attributi che vengono impostati dall'installatore attraverso Lapis e rimangono fissi una volta che l'impianto è a regime. Nel caso delle schede Slave, ad esempio, essi sono l'indirizzo della scheda, necessario per potervi comunicare sulla linea seriale, e la programmazione ingressi-uscite.

- L'interfaccia con il quale l'elemento intende comunicare con gli altri elementi del sistema, formata da una serie di porte di input e output e dall'ontologia relativa.

- La logica di funzionamento interna.

Su Vesta è sufficiente questo, mentre su Lapis e su Visus è necessario implementare classi ulteriori che si vanno ad inserire nella GUI dei due software, permettendo di interagire con l'elemento vero e proprio.

Seguendo queste linee guida ho creato la classe IpCameraElement. In questa prima versione, priva di integrazione con gli scenari, essa non include né porte, nè ontologie e né logica interna, ma soltanto attributi di configurazione, cioè i dati necessari alla connessione alla telecamera IP associata all'elemento. Come descritto nel capitolo 2, la codifica video da me scelta è Motion-JPEG, che utilizza HTTP come protocollo di trasporto del flusso dati. Il video è quindi identificato da una URL, con un formato del tutto identico a quelle usate per accedere ai tradizionali siti web, che rappresenta il primo attributo di IpCameraElement. Il secondo, invece, è formato dalle eventuali credenziali d'accesso necessarie per il collegamento all'URL precedente, nel caso il web server richieda un'autenticazione.

La parte fondamentale del plugin, ossia la ricezione e decodifica del flusso video, ho deciso di implementarla in un nuovo componente, che ho chiamato MotionJpegHandler. È presente nei software Lapis e Visus e non si integra con la struttura a notifiche vista in precedenza, ma viene richiamato direttamente dall'interfaccia grafica degli applicativi nel momento in cui l'utente richiede la visualizzazione del video.

MotionJpegHandler riceve richieste di visualizzazione dagli oggetti che lo richiamano e restituisce loro i frame del video, o eventuali messaggi d'errore. Le classi che vogliono potervi comunicare in questo modo devono obbligatoriamente implementare un'interfaccia, chiamata MotionJpegViewer, e indicare nella richiesta i parametri di connessione, ossia l'URL e le eventuali credenziali di autenticazione descritte in precedenza.

Il componente internamente contiene un oggetto di tipo MotionJpegStream per ogni flusso video che gli viene richiesto di visualizzare, che viene istanziato nel caso non sia presente. Ciascuno di essi riceve e memorizza in una lista i riferimenti agli oggetti Viewer che intendono ricevere i frame e si occupa dell'intera gestione dello stream associato, eseguendo le seguenti operazioni:

- Nel momento in cui viene creato, stabilisce la connessione con la telecamera effettuando una richiesta HTTP all'URL specificato inserendo, se richiesto, i dati per l'autenticazione.

- In caso di risposta positiva, il web server inizierà ad inviare il flusso dati contenente le immagini JPEG concatenate da un separatore che viene specificato nell'header della risposta dall'attributo *boundary*. L'oggetto MotionJpegStream, a questo punto, separerà i dati ricevuti in corrispondenza delle varie occorrenze del separatore, ricavandone i singoli fotogrammi.

- Ogni volta che viene completata l'estrazione di un frame, questo viene inviato a tutti gli oggetti MotionJpegViewer richiamandone il metodo - setFrame.

- Se il riferimento ad un nuovo oggetto MotionJpegViewer viene passato al MotionJpegStream quando questo è già in esecuzione, ne viene richiamato il metodo setFirstFrame, che permette di inviargli l'ultimo frame ricevuto come primo frame da visualizzare.

- In caso di errore, l'oggetto MotionJpegStream interrompe la connessione e lo notifica sia a MotionJpegHandler che ai MotionJpegViewer. I principali errori che è possibile incontrare sono:

- *Host Not Found*: quando non è stato possibile collegarsi all'indirizzo specificato.
- *Timeout*: quando il dispositivo, dopo aver inizialmente accettato la connessione, smette di rispondere.
- *Connection Refused*: il dispositivo non accetta la connessione. Questo può dipendere da un URL errato o da credenziali di autenticazione errate.
- *Unknown Content*: il contenuto della risposta non è un video in formato Motion-JPEG.

Quando un MotionJpegViewer comunica al MotionJpegHandler di non essere più interessato ad un determinato video per il quale prima si era registrato, il suo riferimento viene rimosso dallo stream associato. Quando un oggetto MotionJpegStream rimane privo di Viewer o comunica un errore, viene distrutto dal MotionJpegHandler, lasciando attivi soltanto quelli che comunicano correttamente con la telecamera IP e per i quali vi sono richieste di visualizzazione.

Come detto in precedenza, lo scopo di questa prima versione è stato quello di verificare la correttezza della struttura software implementata e della decodifica del flusso video. Gli aspetti di integrazione con gli scenari e di ottimizzazione dell'interfaccia di visualizzazione, qui volontariamente tralasciati, sono stati presi in considerazione nella seconda versione, che viene descritta nel paragrafo successivo.

## 4.2 Seconda versione

La seconda versione del plugin da me realizzata è quella definitiva, che copre tutti i requisiti stabiliti dalle specifiche di progetto. La struttura di base riprende quella della prima versione, ma è stata rivista in alcuni punti per migliorarne l'efficienza e integrare le parti mancanti.

Il componente MotionJpegHandler ha subito due importanti modifiche:

- L'utilizzo delle librerie OpenGL, e quindi dell'accelerazione hardware, per il rendering del frame.

- Il multithreading, che consiste in questo caso nell'esecuzione di ogni singolo MotionJpegStream in un thread separato.

Nella prima versione, una volta ricavato un fotogramma dal flusso dati MotionJPEG, questo viene salvato in un oggetto di tipo QPixmap, che è la classe del framework Qt per la gestione delle immagini, e poi viene inviato ai Viewer associati, che lo utilizzano nel modo per loro più opportuno. Queste operazioni, compresa la visualizzazione, sono tutte eseguite dalla CPU, che viene quindi sottoposta ad un discreto carico di lavoro.

Nella seconda versione, invece, il frame corrente del video viene memorizzato in una *texture OpenGL*, ossia un'area nella memoria video, identificata da un numero intero, pensata per contenere immagini in un formato adatto alla visualizzazione. Questo approccio ha richiesto una riscrittura parziale dell'interfaccia, e di conseguenza anche dell'implementazione, dei Viewer su Lapis e Visus, che ora sono forzati ad utilizzare OpenGL per il rendering del video. Il principale vantaggio è quello di avere un'unica texture condivisa tra tutti i Viewer, che viene allocata al momento della creazione del MotionJpegStream e deallocata al momento della sua distruzione, mentre prima per ogni frame era necessario istanziare almeno un oggetto di tipo QPixmap, che veniva poi distrutto all'arrivo del frame successivo.

L'aggiornamento della texture avviene utilizzando una procedura denominata *streaming texture update*, che fa uso di un'altra struttura dati di OpenGL: il *Pixel Buffer Object (PBO)*, cioè un'area nella memoria video nella quale è possibile renderizzare immagini che poi possono essere trasferite ad una o più texture. La sequenza di operazioni effettuate è la seguente:

- Un nuovo frame video viene estratto dal flusso dati proveniente dalla telecamera IP.

- L'immagine è JPEG, un formato compresso che non è direttamente leggibile da OpenGL, e viene per questo espansa.

- L'immagine decompressa viene successivamente trasferita nel PBO.

- Una volta che il trasferimento è completo, il contenuto del PBO viene copiato nella texture. Quest'ultima operazione viene svolta interamente dalla GPU in maniera asincrona.

Il PBO, come la texture, è uno per ogni MotionJpegStream. Viene istanziato al momento della sua costruzione e deallocato al momento della sua distruzione.

Come detto in precedenza, ora ogni flusso video viene gestito in un thread separato che, per la precisione, effettua le prime tre operazioni. L'ultima viene invece eseguita dal thread principale dell'applicazione, che riceve dal primo un puntatore al PBO e da istruzione alla GPU di copiarlo nella texture. Questa suddivisione è data dal fatto che il contesto OpenGL è legato al singolo thread per cui, se la texture fosse creata nel thread interno al MotionJpegStream, non sarebbe visibile a quello principale, che contiene l'interfaccia grafica, rendendo di fatto impossibile la visualizzazione del video.

Anche la classe IpCameraElement ha subito delle modifiche importanti, con l'introduzione delle parti relative all'integrazione con gli scenari nel modello degli elementi, cioè le porte e l'ontologia. Quest'ultima è composta da due regole:

- La condizione *"azione attivata"*. All'interno di IpCameraElement, con il termine azione si intende un comando identificato da una stringa che viene mostrato direttamente nella finestra di visualizzazione su Visus, in modo che l'utente possa interagire con il sistema mentre guarda il video. Questa condizione viene attivata appunto quando l'utente seleziona un'azione dall'interfaccia. Per ogni telecamera IP l'installatore può definire fino a sei azioni diverse, che poi però deve associare con opportuni scenari ai comandi veri e propri che vuole vengano effettuati.

- L'azione *"mostra video"*, che permette di comandare all'interno di uno scenario l'apertura della finestra di visualizzazione su Visus o Lapis. Per evitare, nel caso vi siano diversi sistemi collegati all'impanto, una saturazione della banda dovuta alle connessioni simultanee alla stessa telecamera IP, l'apertura della finestra viene preceduta da un messaggio per-

sonalizzato che viene specificato dall'installatore al momento della creazione dello scenario. In questo modo la visualizzazione inizia soltanto nel software in cui l'utente conferma di volerlo fare.

Oltre alle regole dell'ontologia appena elencate, ad IpCameraElement sono state aggiunte una serie di porte, per poterle mappare come accennato nel capitolo precedente. Pur essendo un elemento a comando diretto fisicamente connesso al sistema, le regole dell'ontologia non richiedono comunicazione con il dispositivo per essere eseguite o valutate, per cui IpCameraElement riceve direttamente le notifiche DeviceChangeRequest e le trasforma in notifiche DeviceChanged secondo la sua logica interna, come fosse un elemento virtuale. Non avviene, invece, nessuna interazione con il gestore dei comandi, ossia il componente che normalmente svolge questo compito.

Le porte inserite nell'elemento sono tutte pubbliche. L'azione "mostra video" viene mappata su due porte di input, la prima di tipo stringa chiamata *"Message"* e la seconda di tipo intero chiamata *"HandleVideo"*. Quando uno scenario esegue l'azione, questa imposta nella prima porta il messaggio personalizzato che viene mostrato all'utente per chiedergli se intende o meno visualizzare il video, e nella seconda il valore 1, che scatena i seguenti eventi:

- il sistema locale crea una notifica di tipo DeviceChangeRequest, nella quale inserisce il messaggio letto da "Message", che, attraverso la struttura descritta nel capitolo quattro, arriva al sistema proprietario dell'elemento.
- la notifica viene intercettata dall'istanza di IpCameraElement e convertita in una notifica DeviceChanged che viene reinviata al Core.
- questa risposta viene inviata a tutti gli altri sistemi, compreso quello dal quale era partita la comunicazione. Essi, se ne sono in grado, mostrano la finestra di richiesta che, se confermata, apre la visualizzazione vera e propria del video.

La condizione "azione attivata", invece, viene mappata su altre due porte, entrambe di tipo stringa, chiamate rispettivamente "ActivateAction" e "Actio-

nActivated". Quando l'utente decide di lanciare un'azione attraverso l'interfaccia di Visus, vengono eseguite le seguenti operazioni:

- L'interfaccia imposta come valore di "ActivateAction" il nome dell'azione selezionata. Viene creata una notifica di tipo DeviceChangeRequest contenente la stringa letta dalla porta, che, come nel caso precedente, raggiunge il sistema proprietario.
- Qui l'elemento IpCameraElement ricopia il nome contenuto nella notifica all'interno della porta "ActionActivated" che, essendo quella sulla quale è mappata la condizione "azione attivata", lancia gli scenari collegati.

È evidente che le necessità di ottimizzazione del funzionamento e di integrazione con il resto del sistema hanno aumentato complessità di questa seconda versione rispetto alla prima. Il paragrafo successivo effettua un confronto tra le due, elencando le varie migliorie apportate in termini di prestazioni.

## 4.3 Confronto tra le due versioni

Paragonando le due versioni del plugin da me sviluppate, la seconda, oltre ad essere più completa, risulta migliore in alcuni aspetti. Il primo, molto importante, riguarda l'utilizzo del multithreading nella gestione dei flussi video, che porta notevoli benefici nell'esecuzione delle applicazioni Lapis e Visus su piattaforme multi-core. L'utilizzo dell'accelerazione hardware attraverso le librerie OpenGL è il secondo grosso vantaggio, che porta tre miglioramenti degni di nota:

- A fronte di un leggero aumento della complessità nell'integrazione della visualizzazione del video su Lapis, si ha una notevole semplificazione su Visus, la cui interfaccia grafica è già interamente realizzata con OpenGL.
- Entrambe le interfacce risultano più fluide durante la riproduzione del video, mentre la prima versione, nella quale la decodifica e la visualizzazione sono completamente a carico della CPU, soffre qualche fastidioso rallentamento.

- L'eventuale scaling del video per adattarlo alla finestra di visualizzazione è interamente a carico della scheda grafica e quindi risulta notevolmente più efficiente.

Questo conclude la descrizione del lavoro svolto durante il mio stage, che ha portato alla scrittura di questa tesi. Il prossimo e ultimo capitolo ne trae le concusioni e delinea possibili sviluppi futuri del plugin.

# Capitolo 5

# Conclusioni e sviluppi futuri

Alla conclusione del mio periodo di stage, i responsabili di Master sono rimasti soddisfatti del lavoro svolto in quanto la versione del plugin consegnata loro, cioè la seconda, rispettava tutti i requisiti definiti nelle specifiche.

Prima del rilascio definitivo ho svolto un periodo di test e debug che, come in ogni software, ha portato alla luce qualche piccolo malfunzionamento che è stato prontamente corretto. Tuttavia l'architettura di base del plugin si è rivelata stabile e non ha mai necessitato di modifiche strutturali legate ad errori di progettazione.

Nonostante le interfacce di Visus e Lapis attualmente permettano di visualizzare un solo stream video in un solo punto per volta, il componente sottostante MotionJpegHandler è stato pensato per dare la possibilità sia di gestire più flussi contemporaneamente, sia di visualizzare lo stesso flusso in diverse parti dell'interfaccia.

La versione finale del plugin è stata inserita negli aggiornamenti dei software del sistema UNA rilasciati a luglio 2011 ed è attualmente utilizzata in diversi impianti, nei quali non sono stati riscontrati problemi degni di nota.

I possibili sviluppi futuri nell'integrazione delle telecamere IP sono orientati in tre direzioni:

- L'implementazione di codifiche più efficienti, sia dal punto di vista dell'occupazione di banda, che della qualità del video, come H264.
- Il miglioramento dell'interfaccia di Visus per permettere la visualizzazione di più stream contemporaneamente, come succede nei software di gestione delle telecamere a circuito chiuso.

- L'integrazione della visualizzazione nelle interfacce web del sistema UNA, ossia Sidera Home e Sidera Web.

# Bibliografia


[1] D. Le Gall, “MPEG: a video compression standard for multimedia applications”, *Communications Of The ACM*, v. 34, no. 4, aprile 1991.

[2] M. Kunt, A. Ikonomopoulos e M. Kocher, “Second-Generation Image-Coding Techniques”, *Proceedings Of The IEEE*, vol. 73, no. 4, aprile 1985.

[3] R. Talluri, K. Oehler, T. Bannon, J. D. Courtney, A. Das e J. Liao, “A Robust, Scalable, Object-Based Video Compression Technique For Very Low Bit-Rate Coding”, *IEEE Transactions On Circuits and System For Video Technology*, vol. 7, no. 1, febbraio 1997.

[4] F. Pereira, “MPEG-4: A New Challenge For The Representation Of Audio-Visual Information”, *keynote speech at Picture Coding Symposium 1996*.

[5] RFC 1889, “RTP: A Transport Protocol for Real-Time Applications”, gennaio 1996.

[6] RFC 3550, “RTP: A Transport Protocol for Real-Time Applications”, luglio 2003.

[7] RFC 2326, “Real Time Streaming Protocol (RTSP)”, aprile 1998.

[8] RFC 4566, “SDP: Session Description Protocol”, luglio 2006.