Politecnico di Milano
Facoltà di Ingegneria di Como
Corso di Laurea in Ingegneria Informatica

METODOLOGIE E TECNICHE JINI PER LA PROGETTAZIONE DI AGENZIE DOMOTICHE

Relatore: Chiar.mo Prof. Marco Somalvico
Correlatore: Dott. Francesco Amigoni

Tesi di Laurea di:
Alberto Noseda matr. 622496
Luca Zappa matr. 622436

Anno Accademico 1999 – 2000

Ai nostri genitori

SOMMARIO

Negli ultimi anni abbiamo assistito ad un crescente proliferare nella casa di oggetti elettronici digitali che, in maniera sempre più sofisticata, ci assistono nello svolgimento delle attività di tutti i giorni.

Il Personal Computer è forse il fenomeno più appariscente di questa tendenza, ma non dobbiamo dimenticare i telefoni, le televisioni, i videoregistratori, i forni a microonde, le lavatrici, le lavastoviglie, insomma tutti gli apparecchi elettronici che riempiono le nostre abitazioni.

Il passo successivo all'invasione è quello dell'interconnessione: poiché nella casa sono disponibili un gran numero di oggetti digitali, perché non pensare ad un qualche sistema di comunicazione in grado di far loro scambiare dati e servizi?

La sfida è proprio questa: permettere ai vari dispositivi che ci circondano, di comunicare fra loro.

Lo scopo di questa tesi è quello di dimostrare come si può creare un'agenzia domotica, capace di gestire, ad alto livello, la casa e di effettuare monitoraggio e telecontrollo, utilizzando la tecnologia Jini e le tecniche di intelligenza artificiale. La realizzazione sperimentale si occuperà della definizione globale del sistema (infrastruttura) e di alcune parti specifiche di questo, che serviranno a dimostrare la ricaduta applicativa di questa tesi.

RINGRAZIAMENTI

Desideriamo ringraziare il Prof. Marco Somalvico, per l'opportunità offertaci di lavorare presso l'AIRLAB di Como e per averci consentito di affrontare problematiche di ricerca attuali e stimolanti.

Rivolgiamo, inoltre, un particolare ringraziamento al Dott. Francesco Amigoni per la competenza e la disponibilità dimostrate.

Alberto desidera ringraziare:
La sua famiglia, per averlo sopportato, supportato e spronato.
Piera, Vania, Paolo F, Micaela, Marika, Matty, il Caggia, Paolo S (special thanx), Luca DV, Bric, Cesare, Cmauri, Deni, Massimo, Toni, Elena, Nic, Franz, Ale, Ema, Alessio R, Alessio M, Roberto B, Laura, il Tasso, i componenti della mailing list ItaTrek, i suoi compagni di lavoro in C.D., i suoi studenti di Cantù, i Servizi Generali del Poli e il Beppe, per l'aiuto, il supporto morale, fisico e per i suggerimenti e le idee che gli hanno dato (molte volte inconsapevolmente).
Last but not least, suo "fratello" Stefano.

Luca desidera ringraziare mamma, papà e Daniela che lo hanno aiutato a raggiungere questo importante traguardo con il loro costante e prezioso supporto morale e finanziario.
Ringrazia, inoltre, i parenti e gli amici che sempre hanno creduto in lui, tutti i frequentatori della mailing list Jini-Users per i consigli ricevuti e le persone che l'ambiente universitario gli ha permesso di apprezzare in questi anni di studio.

Il nostro ultimo ringraziamento va al Ceti, senza il quale la ZNS non sarebbe mai esistita (e non solo quella...).

INDICE

INDICE FIGURE, TABELLE E LISTATI

Indice delle figure:

Indice delle tabelle:

Indice dei listati:

1

INTRODUZIONE

"Ogni tecnologia sufficientemente evoluta è indistinguibile dalla magia."
A.C. Clarke

L'area di ricerca di questa tesi è la domotica, ovvero l'automazione della casa (domus); una casa dotata di dispositivi e impianti (agenti domotici) integrati mediante una rete di comunicazione e costituenti un sistema aperto, flessibile e capace di interagire con l'utente in modo diretto ed efficace.

L'agenzia domotica è l'insieme cooperante di più entità, dette agenti, le quali si occupano di gestire ad alto livello l'abitazione domestica.

L'agenzia domotica sostituisce parzialmente l'uomo nel regolare la domus nella quale egli abita [39]. Metaforicamente si può parlare di un castello nel quale il signore (uomo), che lo abita, non deve preoccuparsi di gestire il castello, poiché tale incombenza è demandata alla servitù (l'agenzia domotica) che è coordinata dal maggiordomo; quest'ultimo, tra gli agenti domotici, è quello che svolge le attività di supervisione su tutti gli altri e di interfaccia con l'utente: è tale agente che riceve dal signore i

compiti che la casa domotizzata, tramite l'agenzia domotica, dovrà essere in grado di svolgere. In altre parole, l'agenzia domotica si incarica di espletare una serie di funzioni, rispondendo ai bisogni di chi fruisce dello spazio "domus".

Lo scopo della tesi è di fornire indicazioni su come progettare e sviluppare un'agenzia domotica, utilizzando la tecnologia Jini.

L'automazione domestica, o domotica, è quindi quella disciplina che integra le diverse tecnologie e sistemi presenti in una casa, per offrire un più elevato grado di funzionalità e sicurezza (controllo a basso livello). L'agenzia domotica è il sistema di agenti che cooperano tra loro con lo scopo di consegnare all'utente le funzioni di controllo e di gestione dell'ambiente domestico (gestione ad alto livello).

È opportuno pensare tuttavia al concetto di agenzia domotica, all'interno di un contesto più ampio: l'Ubiquitous Computing. Il termine è stato coniato nel 1988 da Mark Weisser, che immaginava calcolatori embedded collocati nei muri e in qualunque altro oggetto di uso quotidiano [40, 42, 47].

Oggi con tale termine si intende "l'elaborazione ovunque", fatta con l'elaboratore portatile e il telefono cellulare (che probabilmente si fonderanno in un unico apparecchio) dall'uomo, non solo quando si trova in movimento, ma anche da postazioni fisse quali un desktop computer.

Nei prossimi anni, la diffusione delle nanotecnologie e della spintronica (derivata dalla teoria dell'informazione quantistica), presumibilmente porterà dapprima ad avere elaboratori posti attorno ad una persona, negli abiti (wearable computing), successivamente ad avere nanorobot all'interno del corpo umano, i quali formeranno la cosiddetta agenzia antropica.

Nell'immediato futuro, l'agenzia domotica rappresenta il primo passo verso l'ubiquitous computing e, infatti, quale "territorio personale" dell'uomo moderno, la casa si presta ottimamente a tale scopo.

Inoltre, l'ubiquitous computing prevede che l'uomo sia in grado di comunicare e interagire quasi istantaneamente con qualunque punto del globo, quindi anche con la propria casa.

Il nostro lavoro è stato quello di costruire un modello di agenzia domotica, e poi un esempio pratico, utilizzando la tecnologia Jini della Sun Microsystems [16→23], presentata nel 1999, la quale si basa sul linguaggio Java e su TCP-IP (protocolli di comunicazione su Internet) che, utilizzando protocolli di rete, permette di mettere in comunicazione oggetti differenti fra loro (da una lavatrice, ad una stampante fino ad un hard disk) in un'unica rete, detta collaborativa, nella quale non esiste un'unità centrale fornitrice dei servizi, ma dove ognuno mette a disposizione le proprie capacità, per tutti coloro che ne abbiano necessità. Questi servizi sono dinamici, sia per numero, sia per natura.

La nostra agenzia domotica consente di gestire una casa, in tutte quelle funzioni dove una macchina può emulare l'uomo, e di monitorare e controllare i dispositivi, a distanza, e quindi nel complesso la casa, mediante i normali e ormai diffusi mezzi di comunicazione, quali il cellulare, la posta elettronica e il web browser.

L'emulazione avviene tramite la realizzazione di un sistema inferenziale che prevede un unico agente supervisore centralizzato. Il motore inferenziale è stato costruito con Jess, un pacchetto software che permette di realizzare sistemi esperti integrati nell'ambiente Java e quindi anche in Jini.

Esistono, in letteratura e sul mercato, alcuni tentativi di creare una rete domestica con i dispositivi controllabili, ma questo controllo avviene solo a basso livello; in nessun sistema di nostra conoscenza esiste una gestione ad alto livello, come quella pensata per il nostro sistema. Inizialmente si sono studiate le onde convogliate [12] (primi studi nel 1975), ovvero la modulazione del segnale di 50 – 60 Hz della corrente elettrica, utilizzate per accendere e spegnere apparecchi da un sistema

centralizzato, in seguito si sono analizzati altri mezzi trasmissivi [4,12] (linee telefoniche, radiofrequenza, infrarossi) e le loro potenzialità: rete domestica per trasferimento dati, comandi più complessi sui singoli apparecchi, controllo vocale, controllo del consumo elettrico, collegamento diretto ad Internet.

Parallelamente, sul mercato, sono stati introdotti dispositivi "intelligenti", i quali non sono altro che normali apparecchi, con nuove funzionalità aggiunte [3→10]: il forno che si collega ad Internet per scaricare le ricette, il frigorifero che segnala quando manca qualcosa, ecc.. Oppure esistono piccoli sistemi integrati, dedicati ad uno scopo singolo, ad esempio il controllo delle luci o del sistema di allarme complessivo. Solo in questi ultimi due anni [10,13], si sta studiando il modo di integrare tutti i sistemi in uno unico, con tutti i benefici che questo comporta. Tuttavia, al di là di singole realizzazioni ad hoc, non si è visto ancora un sistema che possa integrare, in modo flessibile e dinamico, tutti gli apparecchi, dispositivi e sottosistemi di una casa.

La tecnologia Jini è nata proprio per questo: connettere e far comunicare dispositivi diversi tra loro. Pensata non solo per l'ambito domestico, ma più in generale per una rete di comunicazione globale fra tutti i dispositivi elettronici presenti nella nostra vita di tutti i giorni, la comunità Jini è stata proposta come standard per l'intercomunicazione.

Noi ci siamo serviti di questa tecnologia perché l'abbiamo ritenuta un'ottima base di partenza per la costruzione, non semplicemente di un sistema domotico, ma di un'agenzia domotica.

Nell'ambito dell'intelligenza artificiale, l'agenzia [34] rappresenta la cosiddetta macchina della cooperazione, ovvero quella macchina che emula il comportamento di una comunità di uomini che possono cooperare tra loro. La risoluzione di un problema complesso, implica la scomposizione in sottoproblemi più semplici, ognuno dei quali può essere

risolto da un membro della società, quindi i risultati parziali sono raccolti per ottenere l'obiettivo primario [32].

Un'agenzia domotica, in particolare, permette di integrare tra loro tutti i sistemi presenti in una casa, fornendo un servizio che non sarebbe possibile ottenere dai singoli, con una gestione globale della casa. L'architettura informativa prevede un agente supervisore centralizzato, la cui attività inferenziale consiste nel ricevere i risultati delle rilevazioni, tramutati in fatti, dei sensori e di tutti gli altri dispositivi presenti nella casa e nel decidere quali azioni, o attuazioni, è necessario compiere per gestire determinate situazioni. I fatti inseriti nel motore rappresentano un modello dello stato attuale dell'abitazione.

Poiché il sistema da noi realizzato ha il compito di gestire ad alto livello la casa, non ci siamo occupati delle problematiche riguardanti il controllo di basso livello dei dispositivi, ma li abbiamo simulati via software. La nostra realizzazione si è occupata in primo luogo, di definire l'architettura della nostra agenzia mediante Jini, di risolvere tutti i problemi relativi alla comunicazione verso l'utilizzatore della casa e di ottenere un sistema semplice nell'uso. La parte sperimentale si è poi occupata di gestire gli allarmi relativi a potenziali pericoli che si possono generare in una casa: fughe di gas, perdite d'acqua, fumo, intrusi e controllo del clima. Inoltre è possibile definire scenari particolari di utilizzo, come tutte le azioni da eseguire all'uscita e al rientro del padrone di casa. Questa parte è stata realizzate tramite l'uso combinato di Jini e di un sistema esperto realizzato in Jess.

In aggiunta è stata realizzata la parte relativa al monitoraggio a distanza e al telecontrollo della domus, tramite i mezzi di comunicazione conosciuti. In qualsiasi momento ci si può collegare alla propria abitazione tramite un web browser, la posta elettronica o il telefono cellulare e nel contempo, avere sotto controllo tutti i dispositivi; inoltre in caso di

allarme l'agenzia stessa informa il padrone di casa dell'allarme in corso, in modo che egli possa agire di conseguenza.

La valutazione complessiva del nostro sistema è buona, perché abbiamo ottenuto un sistema flessibile, dinamico, che riesce ad adattarsi alle situazioni e che è in grado di gestire un congruo numero di casi possibili in una casa.

Noi siamo solo all'inizio di un viaggio che si prospetta molto lungo e affascinante. Speriamo di aver posto delle basi solide con la nostra agenzia domotica.

La tesi è strutturata nel modo seguente:

- nel capitolo 2 si mostra lo stato dell'arte della domotica, cosa è già stato realizzato a livello di dispositivi hardware e della loro comunicazione a livello fisico;
- nel capitolo 3 si illustra la teoria delle agenzie dinamiche e non, partendo dall'intelligenza artificiale classica, fino alle ultime realizzazioni di agenzie dinamiche;
- nel capitolo 4 si descrivono tutte le tecnologie utilizzate, con particolare riguardo a Jini, che è il cuore del nostro sistema, a Java che rappresenta il fondamento di tutto, alle Servlet utilizzate per il monitoraggio e telecontrollo via web, e a Jess, che è il motore inferenziale da noi utilizzato;
- nel capitolo 5 si mostra la soluzione logico-concettuale dell'architettura del nostro sistema;
- nel capitolo 6 si descrive il progetto reale, con le sue componenti, i suoi moduli, le scelte effettuate;
- nel capitolo 7 si analizza la realizzazione sperimentale, il suo funzionamento, anche dei singoli moduli e di tutte le funzionalità previste;
- nel capitolo 8 si illustrano le valutazioni del lavoro svolto, le critiche e le direzioni di futuri sviluppi;

- nell'appendice A il lettore trova un glossario con i termini tecnici e gli acronimi utilizzati nella tesi;
- nell'appendice B si riporta il manuale di installazione e uso del sistema da noi sviluppato;
- nell'appendice C si elencano i listati più significativi dell'agenzia domotica.

2

STATO DELL'ARTE

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

2.1 Premessa

Telecomunicazioni, Informatica, Media ed Elettronica di consumo: sono tutti settori che per molto tempo si sono sviluppati in modo separato; oggi stanno convergendo verso un insieme di prodotti e di servizi che cambiano radicalmente il nostro modo di vivere. Dopo avere trasformato il mondo del lavoro e rivoluzionato il settore della comunicazione, stanno entrando nell'universo della casa, non solo attraverso computer e Internet, ma anche modificando e arricchendo le prestazioni degli oggetti di uso comune: frigoriferi, lavatrici, televisori, interruttori, sistemi di sicurezza e per il benessere, che potranno essere attivati e comandati anche senza essere in casa.

Il mercato ha visto, fino ad oggi, prevalere un modello di tipo passivo per l'intrattenimento e l'acquisizione delle informazioni, centrato sull'apparecchio televisivo e su elementi correlati (videoregistratore, ecc…) e sul telefono come elemento interattivo di comunicazione. Oggi questo modello sta cambiando: vengono privilegiati gli aspetti di interazione con l'utilizzatore. Il principale motore del cambiamento è dato dall'esplosivo sviluppo di Internet (e tecnologie correlate); essere connessi alla rete sta diventando un elemento di forte interesse per l'utilizzatore finale ("consumer").

Spesso si sente citare il termine di "casa telematica" [1]: nell'immaginario collettivo questo termine evoca un tipo di abitazione dove una serie di comandi azionati a distanza, tramite un telefono fisso o mobile o tramite Internet, permettono di controllare quanto succede all'interno, dal riscaldamento agli allarmi, all'apertura o chiusura di porte o finestre, all'azionamento di elettrodomestici robotizzati; il tutto collegato a computer che guidano telecamere e comunicano con l'utente; la Domotica non è solo questo: permette anche il telelavoro e il controllo della mobilità dell'utente tramite GSM e GPS.

Buona parte del lavoro svolto sulla scrivania dell'ufficio ha a che fare con la presa in visione di dati, la scrittura di rapporti, la messa a punto di programmi e di appuntamenti; gran parte del tempo è anche speso in conversazioni telefoniche. Tutte queste attività potrebbero essere in realtà effettuate in qualsiasi posto, anche dalla propria abitazione, purché si possa accedere all'insieme di informazioni aziendali che servono. Inoltre il tempo impiegato per recarsi sul posto di lavoro può ammontare anche a un paio d'ore al giorno.

Un'abitazione attrezzata permette quindi anche di effettuare il proprio lavoro collegandosi via web.

L'agenzia domotica permette anche, attraverso un telefono cellulare e un GPS, di offrire alcuni servizi quali: assistenza stradale in casi di emergenza, controllo e diagnosi dell'autovettura in remoto, assistenza sui

percorsi da seguire con informazioni aggiuntive su hotel, negozi, ristoranti, attrazioni turistiche.

Inoltre il collegamento alla rete esterna permette anche servizi finanziari e bancari, servizi pubblici telematici (pubblica amministrazione), commercio elettronico.

Nel paragrafo 2 analizziamo i prodotti presenti sul mercato italiano e non, proseguiamo con considerazioni economico-sociali sulla domotica nel 3. Nel paragrafo 4 illustriamo le aree di possibile automazione di una casa. Nel paragrafo 5 analizziamo nel dettaglio tutte le tecniche trasmissive a livello fisico di rete, mentre nel paragrafo 6 mostriamo la struttura di un rete domestica.

2.2 Prodotti presenti sul mercato

A tutt'oggi, uno scenario di domotica (intesa come automazione a basso livello, e non come gestione ad alto livello) è già possibile. Alla fiera SMAU 2000 a Milano [5,10] è stata infatti presentata una abitazione domotica assemblata da diversi costruttori: BTicino, Merloni, Siemens, Olivetti Lexicon Domustech, Electrolux, Ciao Lab Technologies, Sistema Casa, ecc… i quali per la prima volta hanno dato dimostrazione di un effettivo collegamento funzionale dei sistemi: impianti di climatizzazione, idrotermosanitari, sistemi di regolazione dell'energia e dell'illuminazione, robotica domestica, impianti di sicurezza, home computer, elettrodomestici intelligenti, elettronica di entertainment.

La Merloni [3,8] ha presentato i suoi elettrodomestici Ariston Digital i quali utilizzano la tecnologia WRAP, di sua proprietà, che sfrutta le onde convogliate come mezzo di comunicazione e microcontrollori all'interno di

ogni elettrodomestico, il cui software è basato sulla logica fuzzy. Il pannello di controllo è unico per tutti gli elettrodomestici ed è costituito da un monitor touch screen (Leon@ardo): permette di navigare in Internet, collegare l'utente al centro servizi della Merloni, dove scaricare ricette, ecc…

La BTicino [9] ha presentato i suoi "interruttori" i quali non sono più uno switch fisico per il passaggio della corrente ad ogni nodo elettrico della casa, ma interruttori logici su un bus dati, cui è collegata anche tutta una serie di periferiche aggiuntive quali: telecamere (anche a infrarossi), sensori di posizione, videocitofoni, telefoni. Esistono poi gli attuatori che collegano queste periferiche all'alimentazione elettrica.

La Electrolux ha presentato Screenfridge, frigorifero "intelligente" il quale può pianificare ricette e ordinare cibi, tenendo conto di particolari esigenze dietetiche, delle preferenze e del bilancio famigliare.

In questa casa, basta un solo comando per effettuare automaticamente una serie di operazioni collegate, chiamate funzioni (o scenari) che sono state programmate (dall'uomo, in partenza) tenendo conto delle differenti esigenze della giornata, in base allo stile di vita.

Gli scenari [5] proposti sono:

- uscita: con un solo comando si può uscire dalla casa spegnendo tutte le luci, abbassando le tapparelle e inserendo il sistema di allarme;
- rientro: al rientro si può trovare il clima ideale le luci accese e la musica in sottofondo;
- cena: tramite Internet si può attivare il forno per trovare le cena pronta all'arrivo a casa, inoltre si possono scaricare da Internet le ricette;

- gas: se si lascia il gas aperto, il sistema lo chiude, avvisa dell'allarme in corso e attiva le telefonate di emergenza;
- intruso: in caso di tentativo di intrusione, le telecamere forniscono direttamente sul televisore le immagini dell'intruso e partono le telefonate di allarme;
- risparmio energetico: gli elettrodomestici lavorano in modo da risparmiare energia elettrica e evitano la simultanea accensione che potrebbe causare uno scatto del contatore per sovraccarico;
- film: mentre si guarda un film, sullo schermo possono essere visualizzati i messaggi da altri elettrodomestici, es. la lavatrice quando ha finito il ciclo di lavaggio;
- mattina: si alzano le tapparelle, si prepara la vasca idromassaggio, si accende il riscaldamento in tempo per il risveglio.

Inoltre una versione più grande [11], funzionale e già abitabile, è presente sempre a Milano, in Piazza Diaz 6.

2.3 Considerazioni economiche e di mercato

Per quanto riguarda discorsi di tipo economico, lo sviluppo del mercato è dato da tre settori [1,5]: home automation, home networking e smart home. Il primo si occupa di automatizzare tutte le funzioni possibili all'interno di una casa, introducendo nuove soluzioni tecnologiche al fine di migliorare apparati e prodotti già esistenti e di fornire nuovi servizi di utilità domestica; il secondo di collegare tutti gli elementi di una casa e di farli colloquiare a livello fisico, siano essi apparecchi intelligenti o non intelligenti, computer oppure elettrodomestici; il terzo di integrare, coordinare e controllare il tutto.

I requisiti di home automation sono:

- integrazione: trasformare la sommatoria dei singoli prodotti, in un sistema omogeneo, rendere i diversi sistemi reciprocamente compatibili, fisicamente e logicamente, uniformare il cablaggio, le tecniche di trasmissione e di comunicazione tra i prodotti;
- flessibilità ed espandibilità del sistema all'introduzione di nuovi elementi nella struttura esistente;
- semplicità di utilizzo per qualunque utente: poiché l'utente medio è la massaia, i controlli devono essere semplici, facilmente comprensibili e deve essere sempre presente il corrispondente dispositivo di controllo manuale;
- continuità di funzionamento anche in situazioni critiche che devono poter essere gestite;
- affidabilità in ogni situazione di utilizzo;
- apertura a diversi produttori.

I fattori di successo possibili sono: velocità di diffusione della cultura tecnologica (a tutti i livelli della società), accettazione vantaggi e benefici, massiccia azione comunicativa, standardizzazione dei sistemi, integrazione dell'offerta e del servizio, unica interfaccia per fornitori e utenti, sistemi ad alta flessibilità e integrabilità, e da ultimo (ma non come importanza) la riduzione dei costi (sia dei singoli dispositivi che della messa in opera dell'impianto) che permetterà la diffusione in gran parte delle case.

2.4 Aree di automazione

L'obiettivo finale del sistema domotico (sistema abitativo integrato con un'agenzia domotica) è il controllo completo di tutti i servizi e la realizzazione di nuove funzioni complesse, possibili solo quando più sistemi semplici sono connessi e gestiti in modo intelligente [39].

Le aree di automazione possibili in una casa sono le seguenti [1]:

- Gestione dell'ambiente (microclima ambientale e requisiti energetici)
- Comunicazione e informazione
- Sicurezza
- Gestione degli apparecchi

Ogni area, a sua volta, è suddivisa in sottoaree più specifiche del settore, che vengono di seguito analizzate.

- Gestione dell'ambiente:
    - distribuzione dell'energia
    - climatizzazione
    - riscaldamento acqua sanitaria
    - illuminazione
    - azionamento remoto di sistemi di apertura e di ingresso

Il controllo automatico, attuato mediante un sistema di sensori e attuatori, consente interventi di termoregolazione domestica, in funzione dei mutamenti ambientali, e la verifica costante dei consumi; la programmazione di tempi e livelli di temperatura è orientata verso il confort desiderato in rapporto al massimo risparmio energetico.

Inoltre si attua un controllo sugli eventuali sovraccarichi di tensione e sull'alimentazione di emergenza.

Nel campo dell'illuminazione, l'esigenza più sentita è quella della qualità dell'illuminazione stessa; il punto nodale è rappresentato dall'interruttore (elettronico) che assume il ruolo di sensore e attuatore locale multifunzionale.

I sottosistemi di quest'area hanno minime richieste di banda di comunicazione.

- Comunicazione e informazione:
    - telefono
    - comunicazioni interne con videocitofoni
    - trasmissione dati per controlli sanitari e telemedicina
    - trasmissione dati per attività lavorativa e istruzione
    - informazioni e svago con televisori e radio

I sottosistemi di quest'area assumono rilevante importanza soprattutto perché la rete telefonica può fungere da supporto fisico per l'integrazione.

Inoltre i sottosistemi di quest'area si distinguono per l'elevata richiesta di banda.

- Sicurezza:
    - protezione antifurto, antintrusione, antirapina
    - protezione antincendio, antiallagamento, antiterremoto e da fumo, gas e scariche elettriche
    - telesoccorso e assistenza di persone sole, anziane, disabili o ammalate

L'esigenza di sicurezza rientra tra le principali prestazioni richieste dall'utenza; si divide in "security", ovvero sicurezza personale contro

intrusi, e “safety”, sicurezza globale della casa contro fughe di gas, fumo e allagamenti.

L’agenzia riceve informazioni dalla rete di sensori (fumo, acqua, persone, ecc…) e attua le scelte opportune.

Persone anziane o disabili, utilizzando facili interfacce (vocale, tastiera braille, ecc…) possono gestire la propria abitazione e le apparecchiature presenti in modo molto semplice.

In questa area, le richieste di banda sono variabili, in quanto si prevede la comunicazione solo in caso di allarme, mentre per un utilizzo normale, la banda richiesta è minima.

- Gestione degli apparecchi domestici:
  - lavatrici
  - lavastoviglie
  - frigoriferi, congelatori
  - cucine, forni
  - apparecchi idrosanitari, sauna, idromassaggio

Il settore dei cosiddetti elettrodomestici “bianchi” è quello che è in maggiore evoluzione, con l’introduzione di componenti elettronici, che consentono il miglioramento delle prestazioni, dell’affidabilità e delle funzionalità, oltre alla possibilità della telegestione e telediagnostica di ogni singolo apparecchio.

La richiesta di banda, in questi sottosistemi non è elevata.

Per quanto riguarda la struttura informativa, un elaboratore centrale che permetta di gestire tutte le attuazioni a partire dai risultati di rilevazione, sembra essere la scelta più indicata, anche se una struttura distribuita è più affidabile. L’interfaccia utente deve essere consistente (non deve creare conflitti fra i comandi), di facile impiego (accesso al sistema di controllo da qualsiasi punto della casa) e gradevole [39].

L'interfaccia macchina – macchina deve permettere la totale comunicazione tra tutti i dispositivi dell'ambiente.

2.5 Tecnologie trasmissive

Per la connessione e il networking degli elettrodomestici di una casa, esistono vari mezzi trasmissivi [1,2,4,12,13]: il doppino telefonico TP5, fibre ottiche, cavi coassiali RG-6, che richiedono una messa in posa e quindi l'inserimento "invasivo" di cavi all'interno di una casa; le onde convogliate su linea elettrica (PLC), la radiofrequenza soprattutto nella banda dei 2.4 GHz, i raggi infrarossi, che invece non richiedono nuova posa di cavi.

2.5.1 Linee telefoniche

Le linee telefoniche attuali, costruite essenzialmente per il segnale vocale e ottimizzate per tale scopo, non sono adatte per trasmissione dati ad alta velocità. Le loro caratteristiche di impedenza e attenuazione non possono essere ben controllate.

La prima compagnia che ha cercato di risolvere questi problemi è la Tut Systems Inc. di Pleasant Hill, California [12]. Essa impiega un multiplexing a divisione di frequenza per creare tre canali ognuno dei quali con un differente scopo: servizio telefonico normale (dc – 3400 Hz), segnale ADSL per il collegamento ad Internet (da 25 KHz a 1.1 MHz), networking della casa (5.5 – 9.5 MHz).

Nel canale di network della casa, Tut trasferisce dati a 1Mbit/s con banda tra 5.5 e 9.5 MHz, con 7.5 MHz di frequenza portante. All'interno di

questa banda, Tut usa la tecnologia IEEE 802.3 CSMA/CD, che è solitamente utilizzata per Ethernet, ma incapsula i frame Ethernet in pacchetti più grandi. Per trasferire pacchetti ad alta velocità, Tut utilizza una tecnologia proprietaria di modulazione di impulsi, codificando più bits nello stesso impulso.

La tecnologia Tut è stata utilizzata nella prima specifica 1.0 della rete HomePNA (Home Phoneline Networking Alliance, formata da compagnie quali: 3Com, Advanced Micro Devices, AT&T, Compaq Computer, Hewlett-Packard, Intel, IBM, Lucent Technologies).

Il trasferimento a 1 Mbit/s è lento per la trasmissione di video ad es. MPEG-2 a 2-4 Mbit/s, DVD a 3-8 Mbit/s, HDTV a 19 Mbit/s. Teoricamente è possibile arrivare ad un trasferimento dati nell'ordine di 100 Mbit/s su una linea telefonica, utilizzando porzioni della banda 2-30 MHz. Questa capacità è alla base della seconda versione di HomePNA proposta da Epigram Inc. di Sunnyvale, California e Lucent Technologies Inc. di Allentown.

La release 2.0 di HomePNA, che supporta pienamente la versione precedente, incrementa la velocità di trasferimento dati arrivando fino a 10 Mbit/s utilizzando una banda tra 4 e 10 MHz con frequenza portante a 7 MHz.

La versione 2.0 impiega una modulazione di ampiezza in quadratura con differenza di frequenza (FDQAM), nella quale ogni singola informazione QAM è ripetuta in due regioni di frequenza per aumentare la robustezza del trasferimento dati, nel caso che parte dell'informazione sia attenuata o corrotta.

La ITU sta studiando la possibilità che il networking di una casa, via linee telefoniche, possa generare interferenze con VDSL, la prossima generazione di collegamento a Internet ad alta velocità, anche se questa tecnologia sarà pronta non prima di 2 anni; la soluzione attuale è quella di inserire un filtro nell'interfaccia di rete di ogni apparecchiatura.

2.5.2 Onde convogliate su linea elettrica

Sebbene le linee telefoniche non siano disponibili in ogni casa, le linee elettriche lo sono, quindi già da anni sono stati compiuti sforzi per far comunicare diverse aree della casa tramite onde convogliate.

Il primo di questi sistemi è X-10 [4,12], uno standard sviluppato nel 1976 e largamente diffuso negli Stati Uniti. La produzione è effettuata prevalentemente dall'azienda X-10, ma esistono altre aziende che sviluppano prodotti con questo standard semplice e di basso costo. Consiste in una scatola di controllo e in vari moduli che si connettono direttamente con la linea elettrica, e ogni apparecchio può essere connesso a un modulo che ha un suo codice (0 – 255, più che sufficienti per le esigenze di un'abitazione). Dal pannello di controllo oppure da un computer ad esso collegato, è possibile mandare segnali di accensione/spegnimento ad ogni modulo (impulsi).

Un impulso di controllo consiste in un picco di 120 KHz lungo al massimo 1 millisecondo (uno logico), la sua assenza è lo zero logico. La sincronizzazione avviene mandando gli impulsi nell'intorno (200 microsecondi) dello zero della frequenza della corrente alternata (60Hz negli USA).

Inoltre ogni modulo può essere messo in polling dall'unità di controllo e indicare il proprio stato. La velocità di trasmissione arriva a 50 bit/s.

L'inconveniente di un sistema di questo tipo è che ci sono problemi se il bit rate aumenta oltre i pochi bits per secondo: rumori, interferenze, attenuazione, variazioni di impedenza, riflessioni per impedenza.

CEBus (EIA 600) [4,12] è uno standard aperto che dà le specifiche per utilizzare le linee elettriche, ma anche TP5, cavi coassiali, infrarossi e radiofrequenza.

Il sottolivello MAC del livello 2 del CEBus implementa un CSMA/CDCR. Questo permette ad ogni dispositivo sulla rete di accedere al media in

ogni istante; tuttavia un nodo che vuole mandare un pacchetto dati, deve prima “ascoltare” che non ci siano altri pacchetti in quel momento sulla linea e solo quando la linea è libera, si può mandare il pacchetto (questo è molto simile al protocollo IEEE 802.3).

CEBus inoltre specifica un linguaggio (CAL, Common Language Application) che permette alle varie unità di comunicare tra loro. Ogni unità è definita come un “contesto” in CAL; uno stereo oppure un VCR sono esempi di contesti. Ogni contesto è poi diviso in oggetti (esempi di oggetti sono il volume, il controllo degli alti, dei bassi, ecc…). I segnali di comando possono identificare il contesto e attivare funzioni sugli oggetti, come alzare il volume dello stereo. I ricettori attaccati ad ogni unità ricevono questi comandi e li attuano.

Utilizzando la tecnica OFDM, simile alle tecniche di modulazione per la radiofrequenza, è possibile mandare una grande quantità di dati sulla linea elettrica. Questa tecnica risolve il problema delle riflessioni multiple, che è la maggior causa di interferenza nelle comunicazioni su linea elettrica. In teoria è possibile trasmettere fino ad una velocità di 100 Mbit/s, secondo la compagnia Intellon Inc. di Ocala che implementa questa tecnologia.

La Intelogis Inc. di Draper, Utah, [12] utilizza una tecnologia che trasmette i dati in una banda di frequenza superiore a quella dove si trova il massimo rumore: Plug-in PLX (che è conforme allo standard CEBus CAL), che usa una combinazione di DSMA e CTP. DSMA opera in modo simile alla votazione tra più nodi in una rete Ethernet. Un nodo, quando entra nella rete per la prima volta, “sente” il passaggio di altri pacchetti sulla linea e manda i propri pacchetti solo se è permesso farlo.

Una volta che tutti i nodi si conoscono tra loro, uno schema dinamico di distribuzione di token, è instaurato. Questo evita collisione tra più nodi, evita di fare continuamente votazioni tra nodi e aumenta l’effettivo throughput. La versione corrente arriva fino a 350 Kbit/s, ma la prossima è previsto che arrivi a 2 Mbit/s.

Sempre nel campo delle onde convogliate la High Tech Horizon di Angelholm, Svezia, [12] ha commercializzato una scheda di comunicazione per PC. Avendo un modesto bit rate di 2400 bit/s, questa scheda può essere utilizzata per mandare e ricevere messaggi e dati tra PC e periferiche; inoltre è dato in dotazione un software di riconoscimento del linguaggio, per cui è possibile "parlare" al computer da ogni punto della casa.

Altre compagnie si sono riunite in un'unica organizzazione: HomePlug Powerline Alliance [15], che utilizza una tecnologia sviluppata dalla Intellion e prevede l'installazione di componenti in ogni computer o apparecchiatura che si vuole collegare. La tecnologia utilizzata trasmette dati sopra la banda di 4 MHz (dove c'è la massima interferenza nella rete elettrica a 60Hz) fino alla banda di 21 MHz utilizzando canali di comunicazione multipli, in modo dinamico: in pratica si determina di volta in volta qual'è il canale meno rumoroso e a degradazione minore. La velocità di trasmissione arriva a 8 MBit/s e il suo utilizzo è quello di poter condividere informazioni tra computer nella stessa casa e utilizzare lo stesso accesso ad Internet.

Al di là delle promesse di ogni singolo venditore, il consumatore deve ancora vedere un vero prodotto che permetta alta velocità di trasmissione sulle linee elettriche. Inoltre non sembra ci sia un grande sforzo per rendere i singoli prodotti interoperabili tra loro.

2.5.3 Radiofrequenza (wireless)

La radiofrequenza è la tecnologia a maggior sviluppo attuale, per la trasmissione di dati e voce; in molte situazioni, la radiofrequenza dà una

soluzione conveniente ed economica per la costruzione di una rete (si veda la tabella 2.1), in una casa o in piccoli uffici.

Tecnologie wireless per la rete domestica

Nome	Applicazione	Caratteristiche	Banda di frequenza (GHz)	Tipo di modulazione	Bit rate (Mbit/s)	Organizzazione di specifica	Agenzia di certificazione
802.11 FH	rete dati wireless	disponibilità di crittografia	2.4	FH[1] (Frequency Hopping)	2	IEEE[2]	WLIF (Wireless LAN Interoperability Forum)
802.11 DS			2.4	DS (Direct Sequence)	2	IEEE[2]	WLIF
High-speed 802.11	LAN wireless ad alta velocità	Broadband	5	DMT/OFDM (Discrete Multitone/Orthogonal FDM)	6-54	IEEE[2]	WECA (Wireless Ethernet Compatibility Alliance)
			2.4	DS[1]	11	IEEE[2]	WLIF e WECA
HiperLAN-BRAN[3] (Broadband Radio Access Networks)	LAN multimediale ad alta velocità	Supporta voce, dati, video; può coesistere con i 2.4 GHz; certificabile in USA e in Europa	5	GPSK (Gaussian Phase-Shift Keying)	24	ETSI (European Telecommunications Standards Institute)	
DECT (Digital Enhanced Cordless Telecommunicationts	Voce e dati per case e piccoli uffici	Integra voce e dati	1.88-1.90	GFSK (Gaussian Frequency-Shift Keying)	1.152		
Shared Wireless Access Protocol	Comunicazione wireless per case e piccoli uffici	Basso costo	2.4	FH[1]	2	HomeRF Working Group	
Bluetooth	Standard per la radiofrequenza	Basso costo, corto raggio, supporta solo voce e dati	2.4	FH[1]	1	Bluetooth Consortium	

FDM = Multiplexing a divisione di frequenza
[1] Differenti tipi in diverse parti dello spettro
[2] Nessuna agenzia ha ancora certificato IEEE 802.11, ma la WLIF sta lavorando a questo con l'Università del New Hampshire
[3] BRAN è succeduto ad HiperLAN come nome per lo standard di rete a radiofrequenza ETSI

Tabella 2.1: Tecnologie wireless per la rete domestica

Le infrastrutture attuali di radiomobile (TACS/ETACS e AMPS, prima generazione; GSM/DCS/PCS, seconda generazione) hanno un servizio

dati a velocità molto basse, inutilizzabile per una rete domestica [42]; si sono studiati quindi standard alternativi, specifici.

Gli standard sono: HomeRf, Wi-Fi (IEEE 802.11B) e Bluetooth [2,12,13].

HomeRf (sostenuto da IBM, Motorola e Proxim) usa la "frequency hopping" e viaggia a 2 Mbit/s.

Wi-Fi (sostenuto da Cisco System, 3Com e Lucent Technologies) usa la "direct sequence" dei dati, viaggia a 11Mbit/s ed è più costoso di HomeRf.

Bluetooth (sostenuto da Intel, Ericsson e Nokia) non riguarda solo la casa, ma si propone come standard di radiofrequenza a 2.4 GHz (cui operano anche gli altri) per la connessione di cellulari, computer, palmari, portatili.

IEEE 802.11 permette sia l'utilizzo di DS, sia di FH, e con entrambe le tecniche si arriva ad un massimo bit rate di 2 Mbit/s; tuttavia con una nuova versione che utilizza la tecnologia CCK, si può arrivare a 11 Mbit/s. Il problema di questo standard è che è stato inizialmente pensato per la telefonia cellulare e ha overhead molto alti.

Utilizzando la tecnologia LST che permette di spedire fino a 50 – 100 Mbit/s nella stessa banda dei sistemi a 11 Mbit/s, si può pensare di utilizzare IEEE 802.11 per i video e HDTV. Utilizzando invece DMT assieme a FDM si può arrivare a 6 – 54 Mbit/s.

Queste due ultime soluzioni sono ancora in fase di studio.

HomeRf, rifacendosi alle specifiche SWAP, usa il FH nella banda di 2.4 GHz. Per aumentare la velocità di trasmissione bisognerebbe aumentare la larghezza di banda da 3 a 5 MHz per il FH; questo porterebbe a 11 Mbit/s, ma tutto è attualmente in mano alla FCC.

ETSI supporta due protocolli: HiperLAN, per il traffico ad alta velocità e DECT che opera a minore velocità. HiperLAN, che in futuro si chiamerà

BRAN, è una serie di specifiche per reti operanti a 5 GHz che possono arrivare a 24 Mbit/s (sufficienti per la televisione ad alta definizione). Per gli Stati Uniti non si sa se sarà possibile BRAN, perché la frequenza non è libera.

DECT opera invece tra 1880 e 1990 MHz e ha un bit rate di 1.152 Mbit/s, progettato principalmente per la voce umana; British Telecommunications sta cercando di adattarlo per la trasmissione dati.

Bluetooth (chiamato così in memoria del re scandinavo vissuto nel decimo secolo, che unì numerosi reami danesi) è lo standard che vuole unificare tutta la radiofrequenza a corto raggio (casa o ufficio) e far colloquiare tra loro dispositivi anche non connessi sulla stessa rete. In pratica si occupa di fare da ponte (bridge) tra reti a radiofrequenza diverse.

2.5.4 Standard di trasmissione misti

Standard EIB [1,6]: è uno standard europeo dei primi anni 90 posseduto dalla European Installation Bus Association cui sono associate o affiliate più di 100 aziende europee (tra cui ABB, Siemens, Vimar, Gewiss, Philips). I prodotti sviluppati sono certificati da laboratori. Sono previsti vari mezzi trasmissivi: il doppino, le onde convogliate, la radiofrequenza e Ethernet; possono collegarsi fino a 61455 dispositivi.

Standard BAtiBUS [6]: è uno standard europeo che comprende più di 80 aziende (tra cui Merlin Gerin, Airlec, EDF, Landis&Gir). È uno standard di comunicazione aperto, reso disponibile nel 1989. Sono previsti vari mezzi trasmissivi: il doppino, la radiofrequenza e i raggi infrarossi. La velocità di trasmissione è di 4800 baud.

Standard EHS [1]: è lo standard European Home System, sviluppato a partire dal 1987 nell'ambito dei programmi europei Eureka ed Esprit. È uno standard aperto per il quale è prevista una certificazione dei prodotti. Sono previsti vari mezzi trasmissivi: il doppino, le onde convogliate, il cavo coassiale, la radiofrequenza, i raggi infrarossi. Un sistema può gestire fino a 10^{12} indirizzi.

Standard LonWorks [12]: è uno standard nato nel 1990 (a partire da X-10 e da EIA 709) e posseduto dalla americana Echelon che produce tramite Motorola e Toshiba un chip neuronale. Chiunque può sviluppare prodotti che utilizzano lo standard LonWorks, i quali sono poi certificati. Questo standard è diffuso a livello mondiale e utilizzato da più di 4000 aziende (tra cui anche la Merloni). I moduli utilizzati in tutto il mondo sono 9 milioni, di cui il 20% utilizzato nelle case. La velocità di trasmissione può arrivare a 10 Kbit/s e si possono gestire fino a 32 indirizzi.

LonWorks implementa il CSMA per il livello fisico ed è indipendente dal mezzo trasmissivo utilizzato: TP5, le onde convogliate, il cavo coassiale, la radiofrequenza, i raggi infrarossi e le fibre ottiche.

Attraverso LonWorks è possibile controllare il consumo energetico di ogni singolo apparecchio, permettendo anche risparmi dell'ordine del 10 – 30% e permette di gestire scenari di sovraccarico elettrico e blackout.

HAVi [14] è un'architettura sviluppata da un insieme di aziende per facilitare interoperabilità fra apparecchi realizzati da industrie diverse, in modo da costruire più facilmente applicazioni distribuite sulla rete domestica.

La sua architettura è formata da un insieme di API, servizi e il protocollo di rete IEEE 1394 (high performance serial bus). HAVi realizza un ambiente paritetico, dove ogni device può scoprire, interrogare e

controllare ogni altro apparecchio, inoltre gruppi di device possono cooperare tra loro.

HAVi riconosce quattro categorie di apparecchi elettronici: FAV (Full AV), IAV (Intermediate AV), BAV (Base AV), LAV (Legacy AV). FAV e IAV fungono da controllori, mentre BAV e LAV possono essere solamente controllati.

I servizi di sistema definiti sono:

- Discovery: individua quando i device sono aggiunti o rimossi dalla rete domestica.
- Messaging: stabilisce uno schema di indirizzi e il trasferimento di messaggi.
- Lookup: fornisce il servizio di lookup alle applicazione e ai device.
- Events: implementa un sistema di eventi distribuiti, dove gli elementi software si possono sottoscrivere ad un certo tipo di eventi.
- Streaming: fornisce il servizio di data stream.
- Reservation: permette alle applicazioni di riservare dei device, inoltre si ha la possibilità di realizzare azioni programmate, come per esempio sintonizzarsi e registrare un programma televisivo ad una specifica ora.
- User interaction: realizza due UI.

2.6 Architettura di rete domestica

Quando una rete domestica è collegata ad Internet, vari tipi di dati entrano in casa attraverso un singolo canale; il sistema deve quindi identificare il tipo di dato e indirizzarlo alla destinazione appropriata. C'è bisogno quindi di un gateway tra Internet e la rete domestica (si veda la figura 2.1).

Ad esempio il gateway deve estrarre i comandi di controllo LonWorks dai pacchetti TCP/IP e mandarli alla rete.

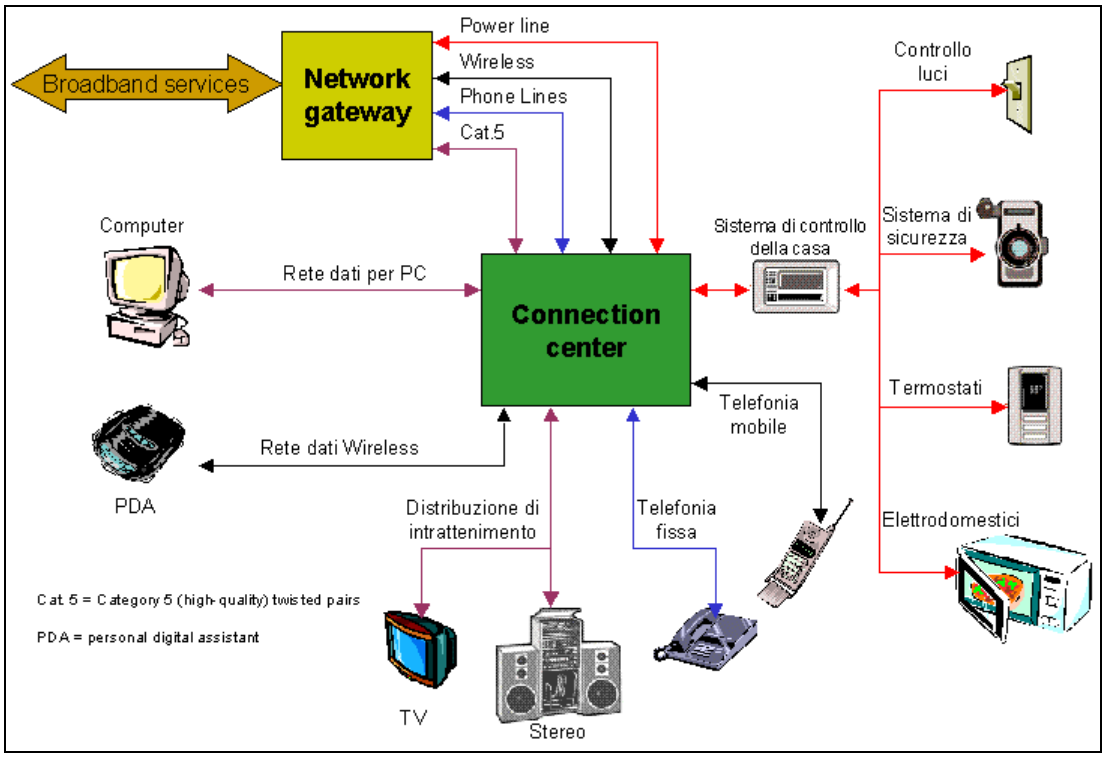

Figura 2.1: Schema di una rete domestica

Inoltre se l'utente della casa (o del piccolo ufficio) usa in modo massiccio Internet, bisogna inserire all'entrata della rete un server, che operi da firewall contro l'attacco di hackers, oltre che favorire il trasferimento di files e avere e-mail direttamente in casa.

3

TEORIA DELLE AGENZIE

"L'intelligenza sembra essere un fenomeno
che trascende l'implementazione tecnologica"
Genesereth

3.1 Premessa

I sistemi per la domotica che abbiamo illustrato nel capitolo precedente, mancano di un'interazione globale, cioè di un sistema che sia in grado di prendere autonomamente decisioni e di controllare la casa in modo più efficace, ovvero un sistema di intelligenza artificiale.

Il sistema da noi realizzato è in grado di ovviare a tale mancanza.

Illustriamo in questo capitolo la teoria delle agenzie partendo dall'intelligenza artificiale classica e il lavoro svolto sulle agenzie dinamiche dal prof. Marco Somalvico e dal dott. Francesco Amigoni presso il Laboratorio di Intelligenza Artificiale e Robotica al Dipartimento di Elettronica e Informazione del Politecnico di Milano.

Si parte dal concetto di agenzia e della teoria alla sua base, per arrivare a descrivere le fasi progettuali (con le tecniche che le rendono possibili), e il concetto di agenzia dinamica.

3.2 Agenzia

L'agenzia [34] è un sistema artificiale ("fatto ad arte dall'uomo") di tipo particolare: la macchina della cooperazione. Essa riflette l'ottica con cui può avvenire lo svolgimento di un compito complesso: si scompone il problema complesso in sottoproblemi semplici (struttura cellulare). La scomposizione può avvenire rispetto alla dimensione temporale (ottica lagrangiana), ovvero risoluzione sequenziale di ogni sottoproblema; oppure rispetto alla dimensione spaziale (ottica euleriana), ovvero risoluzione di sottoproblemi in parallelo. Mentre l'uomo trova più facile una distribuzione temporale (algoritmo risolutivo), in un'agenzia si possono attuare entrambe.

I singoli componenti di questa struttura sono chiamati agenti e alcuni di essi possono cooperare tra di loro: l'agenzia è quindi la macchina che reifica la cooperazione, mentre l'agente è un elemento della società, della cooperazione.

Il termine agenzia è stato usato per la prima volta da Marvin Minsky [32] nel tentativo di spiegare l'intelligenza come combinazione di cose più semplici: gli agenti sono le entità della nostra mente che sanno svolgere compiti semplici, ma presi individualmente non sono in grado di svolgere un compito complesso; per fare ciò basta (secondo Minsky) organizzare adeguatamente gli agenti elementari, ovvero strutturare un'interazione tra loro. Si rende disponibile l'informazione di cosa un agente fa, ma non di come lo fa (interfaccia).

Un'agenzia è quindi un'organizzazione di agenti che fornisce una prestazione complessiva non ottenibile da un singolo agente.

In questo caso gli agenti non sono semplici come quelli di Minsky, ma sono sede di attività inferenziale (produzione di nuova conoscenza, a partire dalla conoscenza attuale) e sono quindi agenti intelligenti, in grado di risolvere problemi non elementari.

Minsky dice che non esiste un modello unico per la mente umana, piuttosto esiste un insieme di modelli che cooperano e competono tra loro: l'agenzia della mente.

L'attività inferenziale consta di ragionamenti logici deduttivi, i quali partendo da alcune premesse (formule logiche), attraverso una catena di applicazioni di regole di inferenza, arrivano ad alcune conclusioni. La più famosa regola di inferenza è denominata Modus Ponens che a partire dalle premesse <α> e <α implica β> deduce la conclusione <β>.

L'utilizzo di un'agenzia è adeguato quando il compito da svolgere è complesso e non esiste un paradigma particolare per il suo svolgimento, e quando questo compito può essere suddiviso in tanti sottocompiti indipendenti, ognuno dei quali può essere risolto da un agente.

È importante che il problema sia scalabile, ovvero che ogni sottoproblema sia della stessa natura del problema di partenza, però di dimensione minore; la scalabilità permette di avere agenti omogenei tra loro.

Gli agenti, elementi costitutivi di una rete, possono essere umani o artificiali, intelligenti o meno [33].

Ciascun agente è una rozza emulazione di un elemento che compone la società, la quale a sua volta è emulata dall'intera agenzia.

Gli agenti non interagiscono solo tra di loro, ma agiscono anche con il loro ambiente e possono essere statici o mobili. Occorre dunque creare

meccanismi di interfaccia e cooperazione che consentano un utilizzo remoto e distribuito di più risorse, implementando forme di comunicazione tra applicazioni.

Gli agenti devono potersi trasferire da un calcolatore all'altro, eseguendo compiti in remoto, appoggiandosi a sistemi operativi che consentano la mobilità del codice.

3.3 Teoria alla base dell'agenzia

3.3.1 Bipolo uomo – macchina

Il ruolo dell'intelligenza artificiale, nei confronti della domotica, si può inquadrare se si considera la visione innovativa dell'uomo come bipolo uomo – macchina [39].

Con bipolo uomo – macchina, si intende un individuo pensante, interagente e comunicante, che esplica le proprie attività intellettive in due siti distinti: il polo uomo – corpo e il polo uomo – macchina. Il corpo integra il cervello con il sistema sensorio e motorio; esiste una macchina elaboratore che emula una parte dell'attività intellettiva del cervello e una macchina robot che emula una parte delle attività sensoriali e motorie dell'uomo.

Per effetto del progresso dell'intelligenza artificiale, esiste una tendenza di concentrazione nel polo uomo – corpo dell'attività di intelligenza creativa, e nel polo uomo – macchina dell'intelligenza fabbricativa [40].

Questa concezione dell'uomo come unica entità, ma fisicamente separata in due luoghi, impone una nuova visione dei processi di comunicazione. Si ha comunicazione intrabipolare (interna al bipolo) e

comunicazione interbipolare (interazione tra uomini). Quest'ultima può essere "comunicatio in praesentia", ovvero tra polo uomo – corpo di un individuo e polo uomo – corpo di un altro, oppure "comunicatio in absentia", tra poli uomo – macchina e poli uomo – corpo (tutte le altre combinazioni) [41].

Un sistema domotico deve quindi soddisfare una duplicità di esigenze relative alle comunicazioni:

- Facilitare l'interazione intrabipolare, dove la macchina è l'agenzia domotica, con la quale l'uomo deve interagire in modo efficace;
- Facilitare l'interazione interbipolare, tra gli uomini che abitano la propria casa domotizzata.

Negli individui anziani o disabili, il polo uomo – macchina può compensare le limitazioni del polo uomo – corpo, eseguendo le attività che questo non è in grado di svolgere; in questo modo l'individuo, inteso come bipolo, non è più limitato.

3.3.2 Modello

Un modello è una rappresentazione, costruita dall'uomo, della conoscenza relativa a un particolare fenomeno del mondo reale, in quanto è finito, oggettivo e sperimentabile; esso è composto da formalismo e forma ed è una scelta finita di informazioni da una sorgente infinita che è il fenomeno.

La macchina rappresenta un modello con componenti e architettura (modo di comporre le singole funzioni dei componenti). Una macchina quindi reifica un modello ed è un rozzo emulatore della realtà: la

macchina produce gli stessi effetti (visti dall'esterno), ma non con gli stessi processi (la macchina emula e non simula) del fenomeno reale.

Quando il modello concerne fenomeni del mondo reale, la macchina è detta macchina dei fenomeni o macchina del mondo.

Quando il modello concerne l'intelligenza e l'interazione col mondo reale, la macchina è detta macchina dell'informazione o macchina dei modelli o metamacchina (in questo caso l'operando è ancora un modello). Esistono tre tipi di informazione (tre sottoclassi di modelli): l'algoritmo, il dato e il problema; questi corrispondono ad altrettante macchine: elaboratore esecutorio, elaboratore gestionale ed elaboratore inferenziale.

Quando il modello concerne la cooperazione tra alcuni enti, allora la macchina è detta macchina della cooperazione o agenzia. L'agenzia quindi emula il comportamento di più uomini di una società.

3.3.3 Tecniche di Intelligenza Artificiale

La DAI (Distributed Artificial Intelligence) è l'estensione [34] delle tecniche tipiche dell'intelligenza artificiale a più risolutori automatici di problemi, motivata dalle seguenti ragioni: la scomposizione di un problema complesso in sottoproblemi più semplici può essere tale che i sottoproblemi siano insolubili se affrontati in modo indipendente l'uno dagli altri; in un sistema alcune parti possono essere più adatte di altre per risolvere certi compiti (o possono adattarsi a farlo).

Si parla di PAI (Parallel Artificial Intelligence), se il problema consta di istruzioni direttamente eseguibili da un componente del sistema distribuito; se ogni componente invece deve eseguire un compito (sottoproblema), non coincidente con un'istruzione elementare, allora si parla di DES (Distributed Expert System). Il DES a sua volta è diviso in

base alla specializzazione degli agenti: se tutti gli agenti possono risolvere un sottoproblema, allora si parla di DPS (Distributed Problem Solving), mentre se ci sono agenti specializzati in un compito che altri non possono risolvere, si parla di DKS (Distributed Knowledge Sources). Si tratta di due soluzioni duali riguardo ad affidabilità ed efficienza: DPS è più affidabile, perché in caso di guasto di un agente, un altro può sostituirlo, ma l'efficienza diminuisce, a causa della quantità di informazioni che gli agenti si devono scambiare; DKS ha la specializzazione degli agenti, così si conosce immediatamente chi deve fare cosa, ma in caso di guasto, non è robusto.

DPS [39] poi si suddivide in altri tre casi: DPS vero e proprio, HDP (Hybrid Distributed Planning), PMA (Plannig for Multiple Agents).

DPS puro si ha quando tutti gli agenti partecipano alla costruzione del piano e successivamente cooperano per eseguirlo: in questo caso, il fattore critico è l'equa ripartizione e distribuzione dei compiti.

In HDP c'è un unico agente preposto alla costruzione del piano, questo può non essere imposto, ma contrattato con gli agenti.

In PMA esiste un unico agente che costruisce il piano, lo verifica e lo distribuisce agli altri agenti.

L'organizzazione possibile tra agenti può essere verticale od orizzontale: verticale se, tra due agenti, uno è identificabile come master e l'altro come slave; il master assegna i compiti allo slave e usa i suoi risultati; orizzontale se ogni agente può essere sia master che slave. Inoltre l'organizzazione può essere fissata in fase iniziale (dynamic initialization), oppure può evolvere durante la soluzione del problema (dynamically opportunistic). Un ulteriore fattore per l'organizzazione è il grado di cooperazione tra agenti: gli agenti possono cooperare tra loro, oppure essere antagonisti o avere varie combinazioni di queste due tipologie.

La natura della comunicazione tra gli agenti si può analizzare con tre indici (o con un unico indice tridimensionale): paradigma con cui avviene la comunicazione, contenuto semantico, protocolli adottati.

La comunicazione è il mezzo che permette di realizzare la cooperazione, lo scambio di conoscenza tra agenti e la distribuzione del piano. Il paradigma è il modo con cui vengono scambiate informazioni. Il contenuto semantico indica la relazione tra gli oggetti utilizzati nella comunicazione e il loro significato. Il protocollo definisce il formato dei dati scambiati, quando e in base a quali regole deve avvenire lo scambio.

Modelli fisici per sistemi DPS:

- Blackboard: è un DKS nel quale si hanno varie KS (Knowledge Sources) indipendenti e asincrone, che comunicano mediante un database condiviso (blackboard) nel quale le KS possono leggere o scrivere in determinati livelli.
- Contract-Net: esiste un'unità centrale che costituisce il piano e poi lo distribuisce ad agenti organizzati orizzontalmente. La distribuzione deve essere uniforme e si usa per questo un sistema di richieste – offerte tra agenti "manager" e agenti "contractor".
- Centralized Multiagent Planning: esiste un piano centrale che, per essere eseguito, richiede che ogni agente esegua i suoi piani individuali. Ogni piano individuale può essere in opposizione agli altri, perché un singolo agente ha una vista parziale, locale e non conosce completamente gli altri agenti. Questi conflitti sono risolti tramite un agente privilegiato che riformula il piano globale per eliminarli. L'agente privilegiato può essere scelto in modo statico (inizializzazione) oppure scelto durante l'esecuzione del piano.
- Deductive Belief Model: questo modello divide gli agenti in due classi: gli osservatori e gli attori. I primi costruiscono dei piani in base alla conoscenza che hanno dei secondi.

- Teoria dei giochi: gli agenti cercano di raggiungere il proprio obiettivo effettuando le azioni più convenienti per essi e per il sistema globale, valutate in base a parametri noti a tutti gli agenti.
- Multientità: gli agenti mirano a raggiungere un obiettivo individuale. L'obiettivo globale è dato dall'insieme degli obiettivi individuali. Ogni agente è una macchina a stati finiti, la comunicazione non è prevista e gli obiettivi individuali sono enunciati di calcolo proposizionale.
- Body – Head – Mouth: esistono tre componenti, uno funzionale (corpo) orientato alla risoluzione di problemi, uno cooperativo (testa) che possiede una conoscenza degli altri agenti e uno comunicativo (bocca) che si occupa di mandare a destinazione messaggi provenienti dalla testa e ridirigervi messaggi provenienti dagli altri agenti. Gli agenti sono egoisti e richiedono la collaborazione solo quando non sono in grado di risolvere un task.
- Choir: considera gli agenti come robot, quindi hanno capacità percettive e attuative. Esiste un particolare agente "direttore" che sincronizza gli agenti e pianifica in modo dinamico. Ogni agente opera sul proprio obiettivo e "percepisce" come il sistema si sta muovendo verso il raggiungimento dell'obiettivo globale.
- Molecolare: gli agenti sono visti come molecole che comunicano fra loro con messaggi, ma anche in modo qualitativo tramite broadcast.

Riassumiamo i parametri per la classificazione dei sistemi di DPS:

- Tipo agente: attivo, passivo. Un agente è attivo, quando può partecipare alla stesura del piano; passivo viceversa.
- Obiettivo: globale, individuale. Un obiettivo è globale se interessa l'intera agenzia; individuale se interessa solo un agente.

- Piano: a priori, run time. Un piano può essere deciso a priori, prima dell'esecuzione; oppure può essere deciso durante l'esecuzione stessa.
- Azioni di un agente: osservare il mondo, pianificare, interpretare un piano, spostarsi, spostare oggetti.
- Comunicazione: paradigma, semantica, protocolli. Modo, contenuto e regole per la comunicazione.
- Memoria: conoscenza del mondo e dei piani altrui da parte di un agente.
- Stato: attributi di un agente e descrizione del mondo.

3.4 Progetto di un'agenzia

La prima fase del progetto di un'agenzia [35] è la decomposizione del problema in sottoproblemi, ognuno dei quali risolubile da un singolo agente ideale, ovvero un agente perfettamente adeguato al sottocompito che deve risolvere.

Alla fine della fase di decomposizione, il progettista ha a disposizione un'agenzia ideale, agenti ideali e la struttura delle connessioni di interazione tra essi.

L'esigenza iniziale è sempre generica, ovvero fa riferimento ad una classe di problemi, mai ad un problema specifico, il quale verrà poi assegnato all'agenzia dell'utente.

La seconda fase è la composizione, ovvero il passaggio dall'idealità alla realtà, cioè la copertura di tutti i sottoproblemi con la disponibilità di agenti reali, i quali sono in grado di soddisfare i requisiti espressi dagli agenti ideali.

La fase di composizione può essere vista come un reclutamento di agenti che devono essere in grado di coprire, con le proprie funzioni, le funzioni espresse dagli agenti ideali.

Se non è possibile realizzare l'agenzia ideale con risorse reali, allora bisogna individuare una nuova agenzia ideale; quindi decomposizione e composizione sono ripetute fino a quando è possibile reclutare con successo.

La terza fase consiste nell'integrazione degli agenti specializzati, eterogenei, in un unico schema omogeneo di cooperazione.

La soluzione migliore (in alternativa a quella di costruire agenti ad hoc) è quella di considerare gli agenti preesistenti come semiagenti operatori specializzati e di integrarli successivamente con un semiagente cooperatore.

La parte OP (operativa) è il semiagente fisso intelligente preesistente. La parte CO (cooperativa) è il semiagente a cui viene demandata la funzione di interfaccia, di comunicazione e cooperazione con il resto degli agenti costituenti l'agenzia. Questa parte può essere installata anche da un agente esterno. Definendo un meccanismo di cooperazione tramite l'installazione e l'attivazione della parte CO, si rende dinamico il meccanismo di cooperazione costruendo una infrastruttura di comunicazione.

Quindi si può inserire la parte CO manualmente (da parte del progettista), oppure sfruttando la tecnologia a codice mobile, che permette al codice stesso di spostarsi da un sito ad un altro, sospendendo la propria esecuzione (codice + stato) e riprendendola sul sito di arrivo. Per sito si intende un'entità in grado di provvedere all'esecuzione e allo spostamento delle unità di elaborazione: nel caso in cui sposto solo il codice, si ha *mobilità debole*, nel caso in cui sposto sia codice che stato, si ha *mobilità forte* [37].

La tecnologia software da utilizzare deve essere un efficiente standard, che permetta di costruire componenti che, pur non essendo stati sviluppati in un progetto comune, siano in grado di collaborare e di poter interagire dinamicamente.

Ad esempio la tecnologia Java consente di costruire componenti software interoperabili e portabili su qualunque piattaforma di una infrastruttura distribuita, la quale può essere immaginata come un vero e proprio middleware, in grado di assicurare la distribuzione e l'interoperabilità dei componenti software necessari per svolgere varie funzioni (agenti cooperativi).

Nella definizione di un'architettura di rete per una agenzia, bisogna tener conto di:

- come rendere pubbliche informazione e conoscenza agli altri agenti;
- sviluppare un modello concettuale per gli agenti e per la loro comunicazione;
- come far comunicare agenti e ambiente di esecuzione;
- individuare piattaforme di trasferimento degli agenti;
- individuare possibilità e requisiti delle risorse per gli agenti;
- identità dell'agente, autenticazione, appartenenza;
- localizzazione dei servizi di un agente;
- sorveglianza sugli agenti;
- garbage collection degli agenti non più attivi;
- ambiente aperto anche ad agenti esterni.

Inoltre gli agenti mobili possono dinamicamente replicarsi in copie che possono cooperare o interagire con altri agenti; gli ambienti possono estendere le funzionalità dei tradizionali sistemi di programmazione o essere essi stessi completi di programmazione.

3.5 Tecnologie a codice mobile

- Agent Tcl [37]: sviluppato presso la University of Darthmouth, fornisce un interprete Tcl esteso per supportare la mobilità forte del codice. Un agente è implementato da un processo Unix che in aggiunta può eseguire le seguenti operazioni:
    - jump: spostare l'esecuzione (interprete Tcl + codice + stato d'esecuzione) in un'altra locazione;
    - fork: creare un nuovo agente (che è una copia dell'agente stesso);
    - submit: inviare del codice in remoto.
- Agent0 [36]: sviluppato da Shoham, è stato il primo linguaggio orientato agli agenti. Fornisce un linguaggio di comunicazione fra gli agenti, basato sul parlato umano. Agent0 è un progetto accademico mai utilizzato per realizzare applicazioni reali.
- AgentBuilder [36]: permette di sviluppare e amministrare gli agenti. E' composto da due parti: la prima, chiamata Toolkit, dedicata allo sviluppo di applicazioni con agenti; la seconda, chiamata RunTime System, fornisce l'ambiente nel quale gli agenti vengono eseguiti. Gli agenti sono scritti in Java e comunicano attraverso KQML (Knowledge Query and Manipulation Language).
- Ara [37]: sviluppato presso la University of Kaiserslautern, è un sistema multiagente che supporta la mobilità forte. Gli agenti sono realizzati con un linguaggio specifico al quale si affianca un linguaggio di programmazione standard (C, C++, Tcl).
- Concordia [36]: sviluppato da Mitsubishi Electric, permette di realizzare e amministrare un'applicazione ad agenti mobili. Concordia estende Java con meccanismi per la mobilità e supporti per la sicurezza, è basato su un insieme di server (struttura runtime) che si occupano di:

  - mobilità: un agente si può spostare da un server ad un altro;
  - comunicazione e coordinazione: la comunicazione fra agenti avviene attraverso gli eventi, mentre la sincronizzazione con punti di sincronizzazione;
  - sicurezza: assicura il recupero da situazioni nelle quali un agente fallisce.
- D'Agents [36]: permette di sviluppare agenti mobili scritti in diversi linguaggi, tra cui Tcl e Java. Per l'esecuzione delle applicazioni ad agenti mobili, D'Agents si serve di un agente server per ogni host.
- IBM Aglets [36,37]: sviluppato da IBM Tokyo Research Laboratory, fornisce meccanismi che permettono di costruire oggetti Java mobili, in grado cioè di migrare da un nodo all'altro di Internet. I concetti chiave di questa tecnologia sono: gli *aglet* e i *contesti*. Un aglet è un oggetto Java mobile, in grado di migrare da un nodo a un altro di una rete. Ciascun aglet possiede un proprio thread nel quale viene eseguito ed è in grado di rispondere a messaggi provenienti da altri aglet o dal contesto in cui l'aglet viene eseguito. Il processo di migrazione di un aglet comporta il trasferimento del codice e dello stato dell'aglet stesso. Un contesto è un oggetto fisso (cioè non mobile), che fornisce l'ambiente in cui gli aglet vengono eseguiti. I contesti, inoltre, gestiscono le operazioni legate alla creazione di nuovi aglet e all'avvio di aglet giunti da altri contesti. Quindi è necessario che su ogni nodo della rete, nei quali gli aglet possono migrare, sia presente, oltre alla Java Virtual Machine, l'agente fisso contesto.
- Java [36,37]: sviluppato da Sun Microsystems, è un linguaggio object-oriented pensato per essere portabile su diverse piattaforme, facile da imparare e sicuro. Successivamente, a causa della grande crescita di Internet, sono stati introdotti meccanismi che consentono di realizzare applicazioni distribuite e mobili. Il

compilatore Java traduce il sorgente Java in un linguaggio intermedio e indipendente dalla piattaforma, chiamato Java Byte Code. Il byte code è interpretato dalla Java Virtual Machine (JVM), che è specifica per ogni particolare sistema operativo. Un altro importante aspetto di Java è il class loader, un meccanismo che permette di trovare e collegare dinamicamente le classi richieste. Il class loader è invocato dalla JVM quando il codice in esecuzione contiene un nome di una classe non conosciuta.

- Java Applet [36,37]: sono delle piccole applicazioni Java, che vengono inserite nelle pagine HTML ed eseguite all'interno di un browser (che all'interno contiene una JVM). Le funzioni che una applet può eseguire sono limitate, in modo da garantire la sicurezza, e vengono costruite a partire dalla classe Java chiamata applet.
- Jumping Beans [36]: è un software framework per sviluppare sistemi mobili multiagenti. In ogni nodo della rete viene eseguito un server che gestisce l'arrivo e la partenza degli agenti mobili; inoltre è presente un unico server che fornisce un supporto centrale per la migrazione degli agenti. JumpingBeans è un'estensione di Java, cioè un insieme di classi per costruire gli agenti. Ogni agente ha un itinerario, che è un albero dove ogni nodo rappresenta un host della rete nel quale l'agente andrà. Se il nodo corrente ha più di un figlio, l'agente si clonerà in modo da poter raggiungere tutti gli host associati al nodo corrente.
- M0 [37]: sviluppato presso l'Università di Ginevra, è un linguaggio interpretato, basato sul concetto di messengers. Messangers sono una sequenza di istruzioni che sono trasmesse ed eseguite dal ricevente.
- Obliq [37]: è un linguaggio interpretato sviluppato dalla DEC, che permette l'esecuzione remota di procedure. Un thread, l'unità di esecuzione Obliq, può richiedere l'esecuzione di una procedura su

un interprete in remoto. Il codice della procedura viene spedito all'interprete remoto ed eseguito da una nuova unità di esecuzione, l'unità di esecuzione di partenza si sospende fino a che l'esecuzione remota della procedura termina. Quando l'esecuzione avviene in remoto, i riferimenti ad oggetti locali sono automaticamente tradotti in riferimenti di rete.

- PLAGENT [36]: è un sistema per lo sviluppo e l'esecuzione di applicazioni ad agenti mobili che pianificano le loro attività. Un piano è una sequenza di azioni che un agente deve compiere. Ogni nodo della rete (dove l'applicazione PLAGENT viene eseguita) deve essere equipaggiato con un PLAGENT server, per accettare e gestire gli agenti in arrivo.
- Safe-Tcl [37]: è stato inizialmente sviluppato dagli autori dello standard Internet MIME. É un'estensione di Tcl concepita per supportare le active e-mail: messaggi che possono contenere del codice che viene eseguito quando il destinatario riceve o legge il messaggio. Quindi in Safe-Tcl non ci sono meccanismi per la mobilità e la comunicazione, ma si usa un supporto esterno, come le e-mail. Sono poi state studiate soluzioni che proteggono il destinatario: viene usata una coppia di interpreti, uno di cui ci fidiamo e l'altro no. L'interprete non fidato ha capacità ridotte, in modo da non fare danni. E' stato realizzato anche un plug-in per il browser Netscape, che permette di inserire nelle pagine HTML, in maniera simile alle Java Applet, degli script Safe-Tcl.
- Sumatra [37]: sviluppato presso la University of Maryland, estende Java in modo da supportare la mobilità forte del codice. Gli agenti vengono eseguiti da un execution engine e dinamicamente viene creato un interprete che estende la macchina astratta fornita dalla JVM, con metodi che permettono la migrazione e la clonazione di agenti.

- Tacoma [37]: estende il linguaggio Tcl con primitive che permettono la mobilità debole del codice. Gli agenti sono implementati da processi Unix che eseguono un interprete Tcl. Bisogna installare un supporto run-time per il check-in e check-out degli agenti.
- Telescript [37]: sviluppato dalla General Magic, è un linguaggio orientato agli oggetti, concepito per la realizzazione di applicazioni distribuite. Due fattori hanno guidato il progetto di questo linguaggio: la sicurezza e la mobilità. Telescript definisce due concetti chiave: i *posti* (places) e gli *agenti*. I posti sono locazioni virtuali nelle quali gli agenti possono essere eseguiti. Gli agenti sono processi software mobili, cioè capaci di interrompere la propria esecuzione all'interno di un dato posto, di migrare in un nuovo posto e di riprendere ivi l'esecuzione. Gli agenti possono comunicare fra loro in modo trasparente, indipendentemente dal fatto che occupino posti localizzati in nodi differenti di una rete. Il linguaggio Telescript fornisce quindi un ambiente per lo sviluppo di agenti mobili.

3.6 Agenzia dinamica

Agenzia dinamica [35] è la denominazione di quella particolare agenzia che si ottiene distribuendo l'uniformità su un insieme di semiagenti operatori eterogenei per mezzo di un sistema a codice mobile. Nell'agenzia dinamica il semiagente operatore è la somma dell'agente fisso intelligente (o FIA, Fixed Intelligent Agent) con l'interfaccia tra agente fisso e agente mobile (o MFI MIA/FIA Interface), mentre il

semiagente cooperatore viene costruito da un agente mobile intelligente (o MIA, Mobile Intelligent Agent).

I semiagenti cooperatori sono derivati da repliche dello stesso MIA, che quindi viaggia, raggiunge, si replica ed evolve su ogni FIA, il quale è in grado di svolgere uno specifico compito.

Il requisito di base è che gli agenti siano collegati da una rete di comunicazione (base per la cooperazione).

Esistono varie fasi nella costruzione di un'agenzia dinamica.

- La prima fase è l'invio del MIA.
- La seconda fase consiste nel viaggio, replica e installazione del MIA su ogni FIA.
- La terza fase è l'estrazione da parte del MIA delle informazioni e funzioni operative del FIA. Si costruisce quindi un AFIA (Abstracted FIA) che costituisce il modello del FIA sottostante.
- La quarta fase è l'organizzazione, cioè la comunicazione delle informazioni estratte nella terza fase. Questa attività è distribuita ed è eseguita da un componente, anch'esso distribuito, detto organizzatore, che coinvolge tutti i MIA. Le regole e i meccanismi per questa fase sono inseriti nei MIA dal progettista; si può avere un unico agente dove le informazioni vengono accentrate, il quale funziona da archivio che gli altri agenti possono interrogare in caso di necessità.
- La quinta fase è la cooperazione, cioè la fase in cui l'agenzia interagisce con l'utente e riceve il compito specifico da eseguire. Il componente cooperatore può essere centralizzato o distribuito ed è costituito alla fine della fase precedente.

In base al problema viene costruito un componente, il pianificatore strategico, che avrà il compito di assegnare gli obiettivi parziali agli

agenti. Il pianificatore crea anche altri due componenti distribuiti: il supervisore e l'integratore.

Dopo questa fase comincia la risoluzione del problema che a sua volta è composta da alcune fasi.

- La prima fase è la pianificazione strategica, che è effettuata dal pianificatore strategico. In questa fase si scompone il problema in sottoproblemi assegnati ai singoli agenti.
- La seconda fase è la pianificazione tattica, che è effettuata da tutti gli agenti al fine di costruire piani che permettano loro di risolvere gli obiettivi di ciascuno.
- La terza fase prevede l'esecuzione dei piani tattici (svolgimento delle operazioni previste nel piano tattico) e la supervisione dell'esecuzione da parte del supervisore, il quale deve controllare che non si verifichino conflitti tra gli agenti. È possibile che per risolvere un conflitto, il supervisore richieda una nuova pianificazione tattica o strategica.
- La quarta fase è l'integrazione dei risultati parziali, così da ottenere il risultato globale. Questa fase è eseguita dall'agente integratore che restituisce la soluzione al problema originario posto all'agenzia.

Vantaggi di un'agenzia dinamica:

- Semplicità di integrazione di agenti eterogenei
- Flessibilità:
    - Scelta la classe di problemi, le fasi di decomposizione e composizione possono essere fatte automaticamente.
    - Uno stesso gruppo di FIA, cambiando il MIA, può affrontare problemi diversi.
    - Si può variare dinamicamente la topologia di un'agenzia, inserendo nuovi FIA (ad esempio perché un problema non può essere affrontato dai FIA attuali reclutati).

4

TECNOLOGIE UTILIZZATE

Java: "write once run anywhere"
Jini: "write once run from anywhere"

4.1 Premessa

Il nostro obiettivo é quello di realizzare un'agenzia domotica, che integri tutte le funzionalità viste nel capitolo 2, permettendo quindi di avere un controllo e un monitoraggio in remoto della casa e una sua gestione "intelligente".

Il nostro presupposto di partenza é una casa già costruita, con una LAN domestica (PAN - Personal Area Network) e collegata alla rete Internet.

Ai fini di questa tesi, il comportamento delle varie apparecchiature è stato simulato via software. Le apparecchiature in questione saranno realizzate in modo da avere una certa capacità di calcolo, tale da poter ospitare un'agente.

Per realizzare tutto questo, abbiamo pensato che una buona scelta fosse Jini, una infrastruttura tecnologica basata sul linguaggio Java, che permette di creare un insieme di servizi (comunità) che possono cooperare tra loro, per svolgere un determinato compito.

Nel paragrafo seguente illustriamo il linguaggio di programmazione Java, il quale permette portabilità del codice su macchine diverse, non richiede capacità di calcolo elevate, ed é pensato per realizzare applicazioni distribuite.

Successivamente approfondiamo Jini, mostrando le sue componenti fondamentali.

Nel paragrafo 4.4 mostriamo come le Servlet sono un valido strumento per una programmazione lato web-server, per la creazione di pagine dinamiche.

Infine analizziamo il Jess, uno strumento per l'attività inferenziale basato su Java.

4.2 Java

Il linguaggio Java [38,44] è stato sviluppato da Sun Microsystems nel 1991 come parte del progetto Green, un gruppo di ricerca che si occupa di sviluppo di software per il controllo di dispositivi elettronici a larga diffusione.

Per sperimentare le proprie ricerche, i ricercatori del progetto Green svilupparono un prototipo chiamato Star7, una sorta di dispositivo di controllo remoto in grado di comunicare con altri dello stesso tipo. L'idea originale era di sviluppare il sistema operativo Star7 in C++, il più famoso linguaggio di programmazione orientato agli oggetti, ideato da Bjiarne Stroustrup.

Tuttavia uno dei membri del progetto Green, James Gosling, si stancò del modo in cui il C++ eseguiva una determinata operazione, si barricò nel proprio ufficio e scrisse un nuovo linguaggio in grado di gestire meglio lo Star7; questo linguaggio venne chiamato Oak (quercia) in onore dell'albero di fronte all'ufficio di Gosling.

Progettato tenendo presente le applicazioni anziché lo stato dell'arte dei PC, Java doveva essere di piccole dimensioni, efficiente e facilmente portabile verso un'ampia gamma di dispositivi hardware e, inoltre, doveva essere affidabile.

Gli elementi che hanno reso adatto Java per lo Star7 si sono rilevati adatti anche per il World Wide Web:

- è di piccole dimensioni, perciò velocizza il prelevamento di programmi da una pagina;
- è sicuro, perciò impedisce di scrivere programmi che diano libero sfogo alla distruzione nei sistemi degli utenti;
- è portatile, infatti può essere eseguito su Windows, Macintosh, Linux e altre piattaforme senza modifiche.

Java può essere anche utilizzato come linguaggio di programmazione di tipo generale per lo sviluppo di software, in grado di essere eseguito su piattaforme differenti, inoltre con l'introduzione del Remote Method Invocation (RMI) è molto adatto per realizzare applicazioni distribuite.

Per dimostrare il potenziale di Java e per evitare che il relativo progetto di ricerca venisse archiviato, nel 1994 venne creato un browser Web in grado di eseguire gli applet Java. Il browser dimostrò due cose di Java: che cosa poteva offrire al World Wide Web e che tipo di programmi si potevano creare tramite Java. I due programmatori Patrick Naughton e Janathan Payne utilizzarono Java per creare il browser, in origine chiamato WebRunner, ma poi ribattezzato HotJava.

Anche se Java e il browser HotJava ricevettero parecchia attenzione all'interno della comunità del Web, il linguaggio decollò realmente solo dopo che Netscape diventò la prima società ad averne la licenza

nell'agosto 1995. Marc Andreesen, un dirigente di Netscape, fu uno dei primi, al di fuori di Sun, ad accorgersi delle potenzialità di Java e offrì a questo linguaggio il suo entusiastico sostegno alla conferenza JavaOne del maggio 1996. "Java rappresenta un'enorme opportunità per tutti noi", disse nel suo intervento.

4.2.1 Caratteristiche

L'obiettivo principale di Java è quello di avere un linguaggio indipendente dalla piattaforma (platform indipendence), in modo che funzioni su macchine e sistemi operativi differenti, e quindi anche su sistemi embedded.

Java è un linguaggio orientato agli oggetti (object oriented) simile al C++, ma più semplice.

Le semplificazioni rispetto al C++ sono:

- non esiste un preprocessore (non esistono quindi le #include e le #define);
- non esiste un header (file .h);
- tutti i tipi di dato sono oggetti, sono quindi stati eliminati tutti i tipi di dato che non sono oggetti;
- è stato eliminato tutto ciò che era all'esterno delle classi (es. le funzioni);
- si ha un compromesso tra ereditarietà singola e multipla (perché l'ereditarietà multipla è difficile da gestire);
- è stata eliminata la possibilità di avere overloading degli operatori (ovvero assegnare lo stesso operatore a tipo di dato diversi);
- si è ridefinito il concetto di puntatore: cioè si è eliminato il polimorfismo; quindi a tutti gli oggetti si accede tramite reference, il che implica che tutte le variabili siano puntatori a oggetti;

- non esiste il GOTO;
- non esistono le conversioni automatiche di tipo di dato.

4.2.2 Compilazione ed esecuzione di un programma Java

Gli ambienti di compilazione ed esecuzione sono differenti rispetto ad un normale linguaggio di programmazione.

Normalmente, dopo aver scritto un algoritmo in un linguaggio di programmazione (il quale algoritmo può già contenere operazioni dipendenti dalla piattaforma di esecuzione), si procede alla fase di compilazione, nella quale si controlla la sintassi e si generano i file oggetto (obj), i quali poi vengono collegati (fase di link) tra loro e a librerie di sistema per creare il file eseguibile, che poi può essere eseguito solo da macchine dello stesso tipo. Nella figura 4.1 è riportato lo schema della compilazione di un linguaggio di programmazione tradizionale. Se poi si volesse utilizzare lo stesso programma su sistemi diversi è necessario ricompilare il codice sorgente su tali sistemi e, molto spesso, bisogna anche procedere precedentemente ad una modifica manuale del codice.

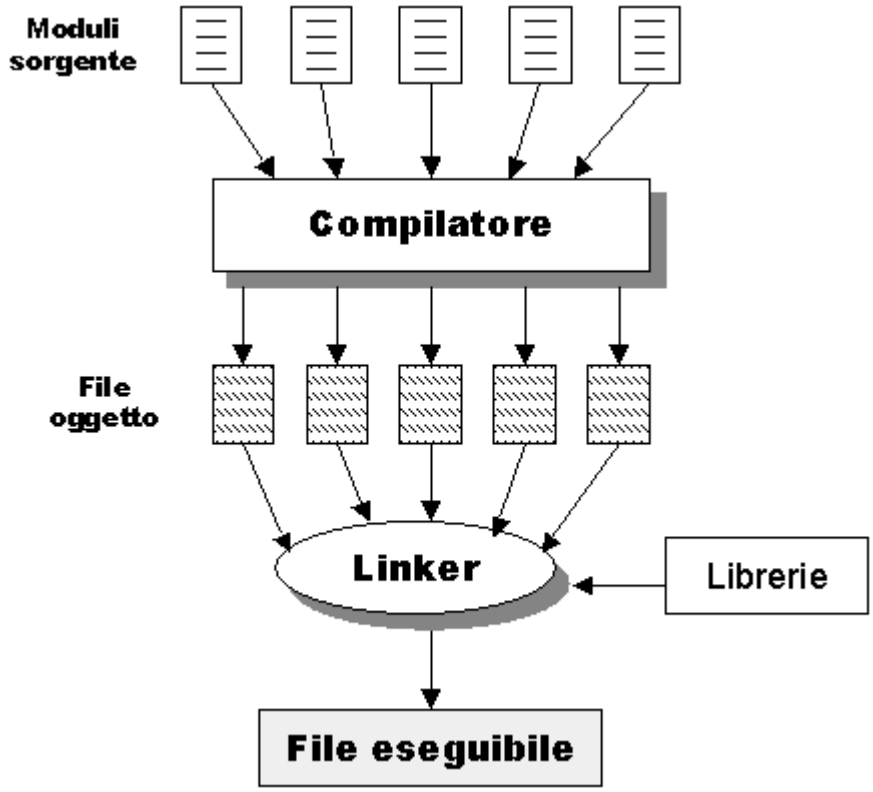

Figura 4.1: Programmi compilati tradizionali

In Java invece queste fasi sono diverse. La fase di compilazione, non porta ad avere istruzioni codificate in linguaggio macchina, ma ad ottenere il bytecode, il linguaggio di una macchina virtuale, la Java Virtual Machine. In pratica ogni sistema operativo deve simulare una Virtual Machine, la quale poi eseguirà i programmi scritti in Java.

Il bytecode è simile al codice macchina prodotto da altri linguaggi, ma non è specifico per alcun processore; in questo modo si aggiunge un livello fra il sorgente e il codice macchina, come mostrato nella figura 4.2.

Dopo la fase di compilazione interviene il Class Loader, la cui funzione è quella di caricare le classi in memoria, non tutte assieme, ma solo quando servono (in modo da risparmiare memoria); ogni classe può essere caricata dal file system locale oppure dalla rete. Dopo il Class Loader si trova il Verifier, il quale ha il compito di controllare istruzioni

potenzialmente pericolose e che una classe abbia tutti i metodi che servono (non è detto che il bytecode della classe sia quello che ci si aspetta). Si può avere anche un caricamento incrementale, ovvero aggiungere delle classi mentre un programma è in esecuzione. Al di sotto del Verifier si trova la Java Virtual Machine (non corrisponde ad un vero processore fisico), che ha il compito di eseguire le istruzioni, cioè trasformarle in azioni capibili dalla macchina sul quale il programma è in esecuzione; è quindi specifica per ogni piattaforma, e può essere inglobata in un browser. Lo schema della compilazione e dell'esecuzione di un linguaggio Java è mostrato nella figura 4.2.

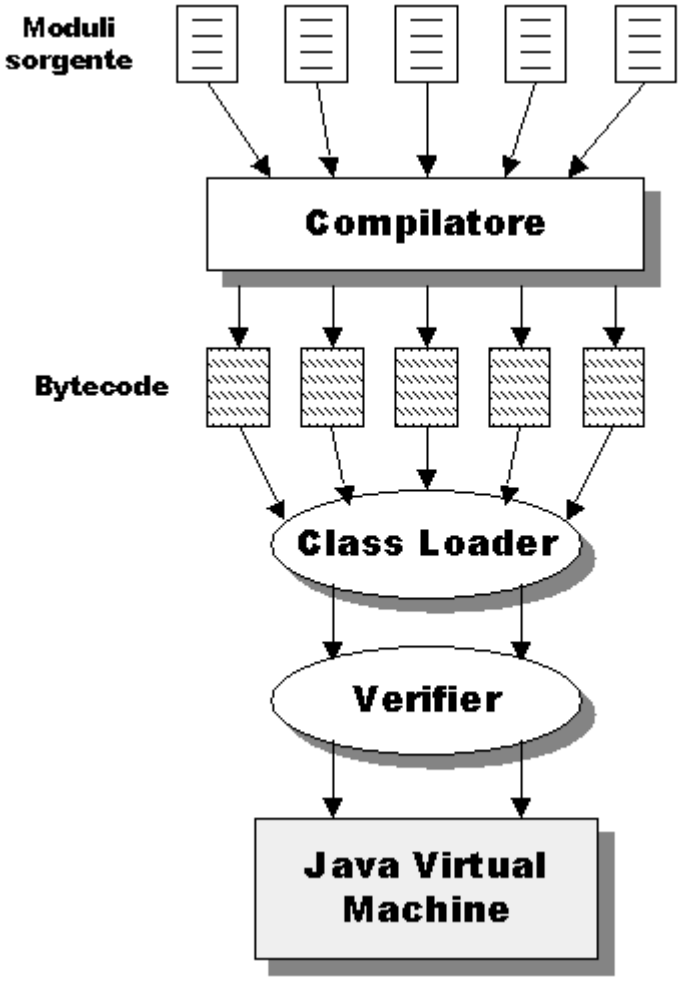

Figura 4.2: Compilazione ed esecuzione di un programma Java

4.2.3 Struttura del file sorgente

Fisicamente una o più classi possono essere contenute all'interno dello stesso file sorgente, che viene detto unità di compilazione (unità minima).

Logicamente i costrutti base sono il package e la class. Il package è un insieme di classi.

In Java poi non esiste l'#INCLUDE del C++, il quale viene sostituito dall'IMPORT.

La struttura di un file potrà essere la seguente:

```
package X.Y.Z;
import C.D.A ;
import F.G.* ;
class A
```

Il nome completo della classe A sarà X.Y.Z.A

4.2.4 Ereditarietà

In Java si può avere solo ereditarietà singola tra le classi, mentre si può avere ereditarietà multipla tra le interfacce; quindi una classe può implementare un certo numero di interfacce.

Il codice in questo caso potrà essere:

```
interface I2
extends I3, I4;


class X
extends Y
implements I2, I3, I4;
```

4.2.5 Deallocazione

In Java non è necessario deallocare gli oggetti creati (non è necessario il DISPOSE del C++), ma si ha la cosiddetta GARBAGE COLLECTION, ovvero il sistema stesso si occupa di togliere dalla memoria gli oggetti non più in uso.

4.2.6 Eccezioni

Un'eccezione è un errore di esecuzione, che non è possibile identificare in fase di compilazione, di solito porterebbe ad un'interruzione improvvisa (le eccezioni ereditano tutte da una superclasse comune); gli exception handling sono dei costrutti che servono ad analizzare la situazione e a trovare una soluzione.

Ad esempio ci può essere il metodo **p** che invoca il metodo **q**, in **q** ci potrebbe essere un *division by 0*, **q** lo può sapere a priori, **p** no, però **p** potrebbe essere in grado di correggere l'errore.

Nel metodo chiamato si inserisce quindi una riga di istruzione *throw*, questa "solleva" l'eccezione, ovvero segnala che in quel punto ci potrebbe essere un problema e, quindi, esegue un'operazione di salto non locale (salto tipo subroutine) al metodo chiamante. Il metodo chiamate a sua volta, usa l'istruzione *try* che serve ad indicare una serie possibile di soluzioni per altrettante eccezioni; le soluzioni sono indicate dall'istruzione-blocco *catch*.

I blocchi catch, all'interno di un try, vengono eseguiti in sequenza finché non si trova il blocco che serve, quindi i blocchi catch vanno scritti in ordine inverso di ereditarietà.

In Java, nell'intestazione del metodo, devo specificare quali eccezioni può sollevare (così le conosce già anche il compilatore).

Il codice sarà:

Metodo chiamato:

```
void in () throws ecc1, ecc2;
…
div (x,y);
if (y==0)
throw new ecc1();
```

Metodo chiamante:

```
try {
..
catch (ecc1) {
…
}
}
```

4.2.7 Multithread

In un linguaggio tradizionale di programmazione, esiste un unico "flusso" di istruzioni (un'istruzione per volta), esso non è sempre lineare, ma comunque è unico.

In Java si parte invece con un unico flusso di controllo, che può essere poi espanso. In Java un flusso è un oggetto di una classe il quale eredita dalla superclasse *Thread.* Ovviamente il flusso principale è il *main*.

Il codice in questo caso sarà:

```
main () {
…
x = new miothread ()
…
x.Start ();
```

a questo punto parte il codice nel metodo run della classe:

```
class miothread extend Thread
void run () {
…
}
```

Per evitare che due thread accedano alla stessa risorsa, si ha un meccanismo di monitor (oggetto): se si esegue un metodo all'interno del monitor, nessun altro thread può eseguirlo.

Il codice sarà:

```
class M {
synchronized void n () {
…
}
}
```

Se ad un certo punto si ha bisogno di generare un evento, si può sospendere la syncronized con l'istruzione WAIT, e poi si riattiva la syncronized con l'istruzione NOTIFY.

4.2.8 Applicazioni distribuite

In Java si possono eseguire parti di codice diverse, che si trovano fisicamente su macchine diverse, anche per costruzione (PC, minicomputer, workstation).

Esistono due modi di vedere la cosa: a basso e ad alto livello.

A basso livello il concetto primario è quello di SOCKET, ovvero un canale di comunicazione tra due macchine (connessione bilaterale).

Esistono due classi in Java: Socket e Server Socket. La classe Socket serve per la richiesta di connessione, e bisogna specificare il nome del

dominio della macchina verso la quale richiedere la connessione e la porta (livello 4 delle reti o TCP) di connessione.

```
Socket s = new socket (“ing.unico.it”, 1701)
```

La classe Server Socket è usata dall’altro lato e si specifica solo la porta su cui la classe si mette in ascolto.

```
Server Socket w = new ServerSocket (1701);
Socket s = w.accept ();
```

Si possono anche accettare connessioni multiple con il multithread.

Ad alto livello, un oggetto richiama i metodi di un altro oggetto, anche se sono dislocati su macchine diverse, tramite RMI (Remote Method Invocation).

In realtà il compilatore RMI (RMIC) genera due nuove classi: lo stub sul lato client e lo skeleton sul lato server. Lo stub riceve le richieste dal chiamante, le comunica attraverso un socket allo skeleton, che a sua volta invoca il metodo richiesto dell’oggetto remoto. Il processo inverso viene utilizzato per comunicare i risultati.

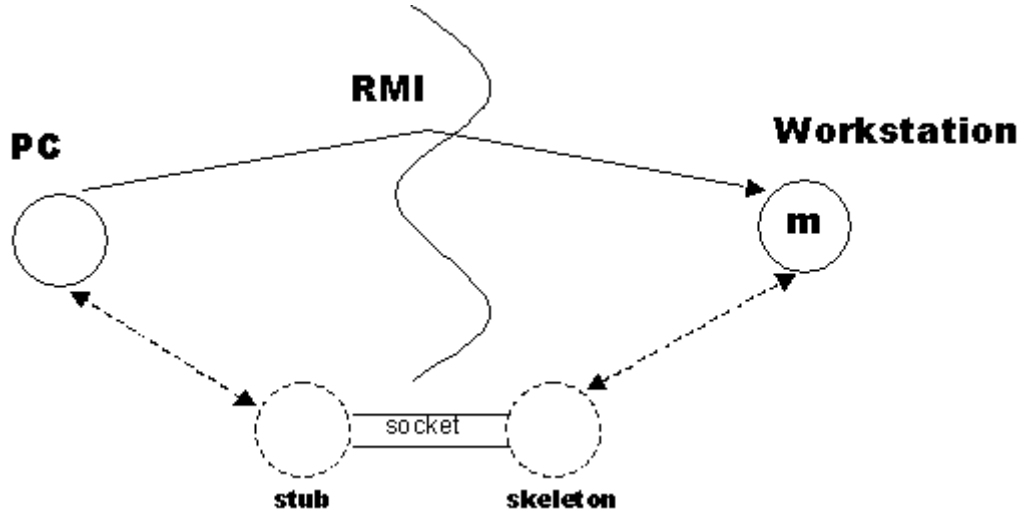

Figura 4.3: Applicazione distribuita

Nella figura 4.3 si vede come l'uso di RMI richiede, da parte del sistema, la creazione dello stub, dello skeleton e di un socket per la comunicazione.

Il codice per la realizzazione di un oggetto remoto è:

```
interface I extends Remote

class S extends UnicastRemoteObject
implements I
```

4.3 Jini

Jini è una tecnologia che si basa su Java e su TCP-IP e che, utilizzando protocolli di rete, permette di mettere in comunicazione oggetti differenti fra loro in un unico network, detto collaborativo, in cui ogni elemento non deve essere installato in un sistema centrale, ma dove ognuno mette a disposizione dei servizi, per tutti coloro che ne abbiano necessità.

In pratica, ogni volta che un nuovo device viene connesso al circuito, prima annuncia chi è e cosa è in grado di fare, successivamente spiega come deve essere utilizzato, attraverso la sua interfaccia.

In una comunità Jini i servizi possono essere dinamicamente aggiunti o tolti, le applicazioni possono poi ricercare tali servizi ed usarli, senza averne a priori una specifica conoscenza.

Ad esempio una stampante, nel momento in cui si connette ad una comunità Jini, prima si presenterà e poi esporrà la sua interfaccia, contenente per esempio il servizio print. Se successivamente un frigorifero Jini dovesse aver bisogno di stampare una pagina con dei dati relativi al proprio funzionamento, sarà sufficiente che vada a ricercare chi, nella comunità, mette a disposizione il servizio print e poi richiederne l'esecuzione.

4.3.1 Servizi

Il concetto più importante, all'interno dell'architettura di Jini, è quello di servizio [17]. Un servizio è un'entità che può essere usata da una persona, da un programma, o da un altro servizio. Un servizio può essere computazione, memoria, un canale di comunicazione, un dispositivo hardware, o un altro utente. Due esempi di servizi sono: stampare un documento e tradurre un documento da un formato ad un altro.

Un sistema di Jini non dovrebbe pensarsi come un insieme di clients e servers, o utenti e programmi, o programmi e archivi, piuttosto, un sistema Jini consiste in un insieme di servizi che sono raggruppati per il compimento di una particolare operazione. I servizi possono usare altri servizi e un client di un servizio può essere anch'esso in un servizio con altri clients. La natura dinamica di un sistema Jini permette ai servizi di essere aggiunti o ritirati in qualunque momento. Jini fornisce i meccanismi per la costruzione, la ricerca, la comunicazione e l'uso dei servizi del sistema distribuito. Esempi di servizi sono: dispositivi quali le stampanti, i monitor, o i dischi; software quali applicazioni o programmi di utilità; le informazioni quali le basi di dati o gli archivi; e gli utenti del sistema.

I servizi, in un sistema Jini, comunicano tra loro usando un protocollo, che è un insieme di interfacce scritte nel linguaggio di programmazione Java.

4.3.1.1 Lookup Service

Il Lookup Service [18] ha il compito di ricercare e trovare i servizi presenti nella federazione. Esso non è altro che una tabella contenente le interfacce, che descrivono le funzionalità dei servizi presenti nella comunità Jini; in aggiunta vengono memorizzati degli attributi, che servono per specificare in maniera più completa il servizio. Tipici attributi

sono il nome, la descrizione e la locazione; essi sono descrizioni testuali maggiormente comprensibili da un utilizzatore umano.

Un servizio è aggiunto al Lookup Service utilizzando due protocolli chiamati discovery e join: in primo luogo il servizio individua il Lookup Service (usando il protocollo discovery) e poi si unisce ad esso (usando il protocollo join).

4.3.1.2 Java Remote Method Invocation (RMI)

La comunicazione fra i servizi avviene utilizzando Remote Method Invocation (RMI) [17,24] fornita da Java.

RMI permette non soltanto di trasferire dati da un oggetto ad un altro attraverso la rete, ma anche oggetti completi, compreso il codice. Gran parte della semplicità del sistema Jini è consentita da questa capacità di spostare il codice, incapsulato nell'oggetto, attraverso la rete.

4.3.1.3 Leasing

L'accesso a molti dei servizi nell'ambiente Jini è basato sui leasing [23]. Un leasing è una concessione di accesso garantito per un certo periodo di tempo. Ogni leasing è negoziato fra il richiedente e il fornitore del servizio: un servizio è chiesto per un certo periodo; l'accesso è assegnato per un certo tempo, possibilmente tenendo in considerazione il periodo richiesto.

Se un leasing non è rinnovato prima che sia esaurito, o perché la risorsa non è più necessaria, o perché il client o la rete vengono a mancare, o ancora perché non è consentito al leasing di essere rinnovato, allora la risorsa può essere liberata.

I leasing possono essere di due tipi: esclusivi o non esclusivi; quelli esclusivi assicurano che nessun altro possa usare la risorsa (ottenere un altro leasing) durante il periodo del leasing; quelli, invece, non esclusivi

permettono un uso condiviso della risorsa (sono concessi più leasing sulla stessa risorsa).

4.3.1.4 Transazioni

Una serie di operazioni, all'interno di un singolo servizio o riferite a più servizi, possono essere inglobate in una transazione [21].

Una transazione è un'unità indivisibile di esecuzione; quindi o vengono eseguite tutte le operazioni di una transazione, oppure la transazione non deve avere alcun effetto sul sistema: approccio "tutto o niente". Inoltre l'esecuzione di una transazione deve essere indipendente dalla contemporanea esecuzione di altre transazioni.

In particolare Jini per garantire una transazione utilizza il protocollo di commit a due fasi (two-phase commit) delle basi di dati.

Tale protocollo ricorda, nelle sue linee essenziali, un matrimonio. La decisione di due persone viene raccolta e ratificata da una terza persona, che celebra il "matrimonio". Affinché il "matrimonio" abbia luogo, è necessario che entrambi i partecipanti esprimano la loro volontà di sposarsi; il celebrante, durante la prima fase, raccoglie il desiderio di sposarsi, espresso separatamente dai due partecipanti, per poi, in una seconda fase, dar loro notizia che il matrimonio è avvenuto.

In realtà, il celebrante (TM, Transaction Manager) invia un messaggio a tutti i partecipanti (RM, Resource Manager), per informarli che il protocollo è iniziato. Ogni RM manda la propria risposta al TM entro un certo periodo, risposta che può essere positiva o negativa. Il TM colleziona tutte le risposte, se tutte sono positive, si prepara a comunicare l'avvenuta transazione; se qualcuna è negativa oppure se il time-out scatta senza aver ricevuto tutte le risposte, il TM si prepara ad inviare la comunicazione di transazione abortita. Si conclude così la prima fase.

La seconda fase parte con il TM che invia la decisione a tutti gli RM e imposta un altro time-out. Tutti gli RM che ricevono la decisione, ne prendono atto e devono confermare al TM l'avvenuta ricezione. Il TM colleziona tutte le risposte, se il time-out scatta prima che esse siano arrivate, si ripete da capo la seconda fase.

4.3.1.5 Eventi

L'architettura Jini supporta gli eventi distribuiti [22]. Un oggetto può permettere che altri oggetti si registrino a suoi particolari eventi e, quindi, ricevano una notifica nel caso tale evento si verifichi.

4.3.2 Componenti

I componenti di un sistema Jini possono essere segmentati in tre categorie [17]: l'infrastruttura, il modello di programmazione e i servizi. L'infrastruttura è l'insieme dei componenti che permette lo sviluppo del sistema federato Jini, mentre i servizi sono le entità all'interno della federazione. Il modello di programmazione è l'insieme delle interfacce che permette la costruzione di servizi affidabili, compresi quelli che fanno parte dell'infrastruttura e quelli che si uniscono alla federazione. Queste tre categorie, benché distinte e separabili, sono intrecciate a tal punto che la loro distinzione può sembrare poco chiara. É possibile sviluppare sistemi che abbiano alcune delle funzionalità del sistema Jini con delle varianti in tutte e tre le categorie. La forza di un sistema Jini risiede nel fatto che è un sistema sviluppato con una particolare infrastruttura, con i modelli di programmazione descritti e basato sulla nozione di servizio.

4.3.3 L'infrastruttura

L'infrastruttura tecnologica di Jini include quanto segue:

- un sistema di sicurezza distribuito, integrato con RMI, che estende il modello di sicurezza della piattaforma Java al mondo dei sistemi distribuiti;
- discovery and join protocols, protocolli di servizio che consentono al servizio di scoprire, diventare parte, e avvisare gli altri membri della federazione;
- lookup service, che serve come un deposito di servizi: è una tabella dove le righe sono oggetti scritti in Java.

In figura 4.4 è riportata l'architettura complessiva di Jini.

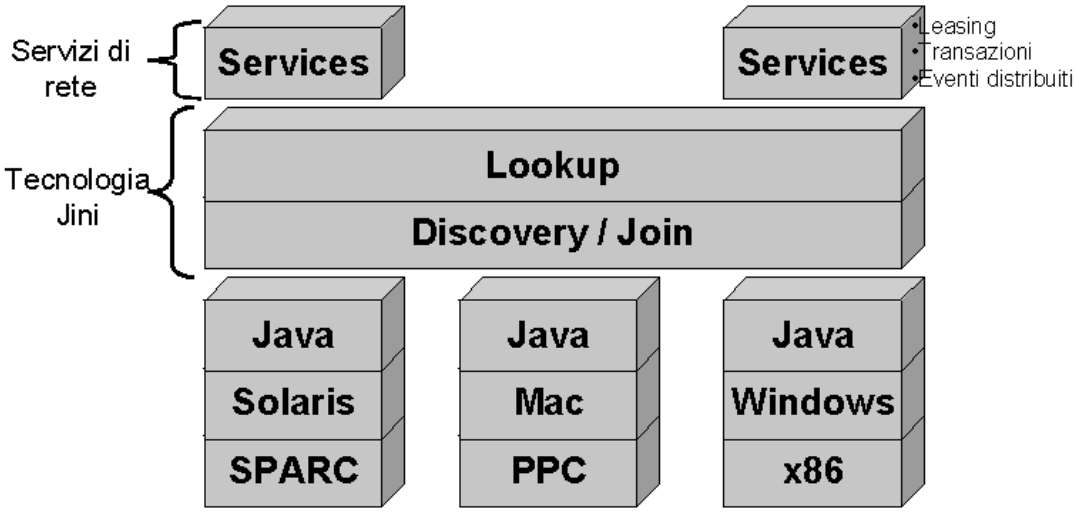

Figura 4.4: Architettura della tecnologia Jini

4.3.4 Il modello di programmazione

Le righe del lookup service sono gestite con il meccanismo del leasing, permettendo al lookup service di riflettere esattamente l'insieme dei servizi attualmente disponibili nella comunità Jini. Quando i servizi si uniscono o abbandonano il lookup service, vengono generati degli eventi. Gli oggetti che hanno manifestato l'interesse nel ricevere tali eventi ottengono le notifiche quando nuovi servizi diventano disponibili o vecchi servizi cessano di essere attivi. Il modello di programmazione si basa sulla capacità di spostare il codice, che è supportata dall'infrastruttura.

Le interfacce che compongono il modello di programmazione di Jini sono le seguenti:

- leasing interface: definisce un modo per allocare e liberare le risorse usando un modello rinnovabile e basato sulla durata;
- event e notification interface: sono un'estensione del modello di evento usato nei componenti JavaBeans™ per sistemi distribuiti, permettono la comunicazione basata sugli eventi tra servizi Jini;
- transaction interface: permette ad un gruppo di servizi di cooperare secondo l'approccio delle transazioni, in modo tale che il cambiamento fatto dal gruppo avvenga in modo atomico (tutto o niente – commit a due fasi).

Non è espressamente richiesto che l'implementazione di un servizio Jini utilizzi il modello di programmazione, ma tali servizi devono utilizzare tale modello per la loro interazione con l'infrastruttura della tecnologia Jini. Per esempio, ogni servizio comunica con il lookup service usando il modello di programmazione; se un servizio offre le risorse utilizzando il meccanismo del leasing, la registrazione del servizio, con il lookup service, deve essere effettuata con il leasing e rinnovata periodicamente.

4.3.5 Protocolli di Discovery, Join e Lookup

La parte basilare del sistema Jini è composta dai tre protocolli chiamati discovery [19,20,31], join [20,31] e lookup [18,27,31].

I protocolli di discovery e join vengono usati quando un nuovo servizio si vuole unire alla comunità Jini; il primo viene utilizzato dal servizio per cercare un lookup service presso il quale registrarsi; il secondo, invece, viene utilizzato quando il servizio, dopo aver individuato il lookup service, desidera unirsi. Infine Lookup si utilizza quando un client o un utente deve individuare ed utilizzare un servizio descritto dalla sua interfaccia (scritta nel linguaggio di programmazione Java) e da altri attributi.

Lo schema in figura 4.5 descrive il processo di discovery.

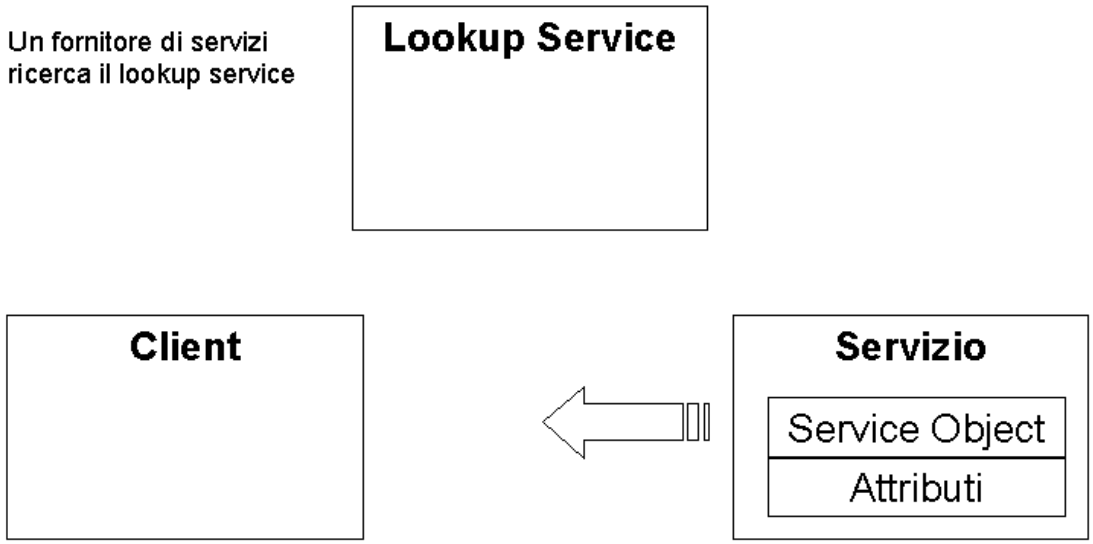

Figura 4.5: Discovery

Il processo di discovery usa tre tipi di sottoprotocolli:

- Muliticast Request Protocol: utilizzato da un nuovo servizio Jini per trovare tutti i Lookup Service presenti su una LAN, usando il multicast UDP.

- Multicast Announcement Protocol: usato dai Lookup service per annunciare la loro presenza su una LAN utilizzando il multicast UDP.
- Unicast Request Protocollo: usato da un servizio Jini per contattare un Lookup service utilizzando l'unicast TCP (Transport Control Protocol); questo protocollo viene solitamente utilizzato quando un servizio si vuole registrare presso un Lookup service che risiede fuori dalla LAN e di cui si conosce l'indirizzo.

Discovery/Join è il processo di aggiunta di un servizio ad un sistema Jini. In primo luogo, il fornitore di servizio individua un Lookup service tramite il protocollo di discovery; a questo punto, attraverso il protocollo di join (si veda lo schema in figura 4.6), il servizio si registra presso il Lookup service che provvederà ad inserire una nuova riga nella tabella contenente il Service Object del servizio e i suoi attributi. Il Service Object contiene l'interfaccia del servizio scritta nel linguaggio di programmazione Java, con indicati i metodi che possono essere invocati. In aggiunta vengono anche allegati una serie di attributi, che sono usati sia per descrivere meglio il servizio, che per la fase di ricerca.

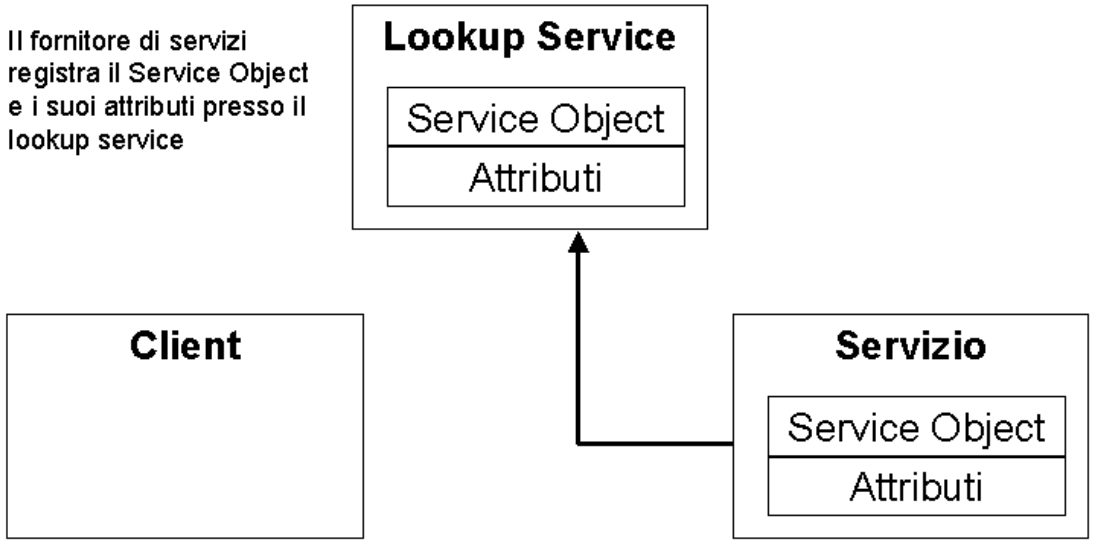

Figura 4.6: Join

In Jini sono stati introdotti (nel package `net.jini.lookup.entry`) i seguenti attributi:

- Name: contiene il nome;
- Location: specifica la collocazione fisica (edificio, piano, stanza);
- ServiceInfo: informazioni generiche sul servizio (nome, produttore, venditore, versione, modello, numero seriale);
- Comment: descrizione del servizio.

In aggiunta è possibile inserire nuovi attributi creati su misura dal programmatore e per fare ciò è sufficiente che un oggetto implementi l'interfaccia Entry.

Il protocollo di Lookup (figura 4.7) serve per trovare ed utilizzare un servizio presente nella comunità Jini. Un client richiede al Lookup service di individuare un servizio con certe caratteristiche (il tipo di interfaccia e gli attributi associati); se questo esiste il Lookup service spedisce al client copia del Service Object, che viene utilizzata dal client stesso per invocare direttamente i metodi del servizio (figura 4.8).

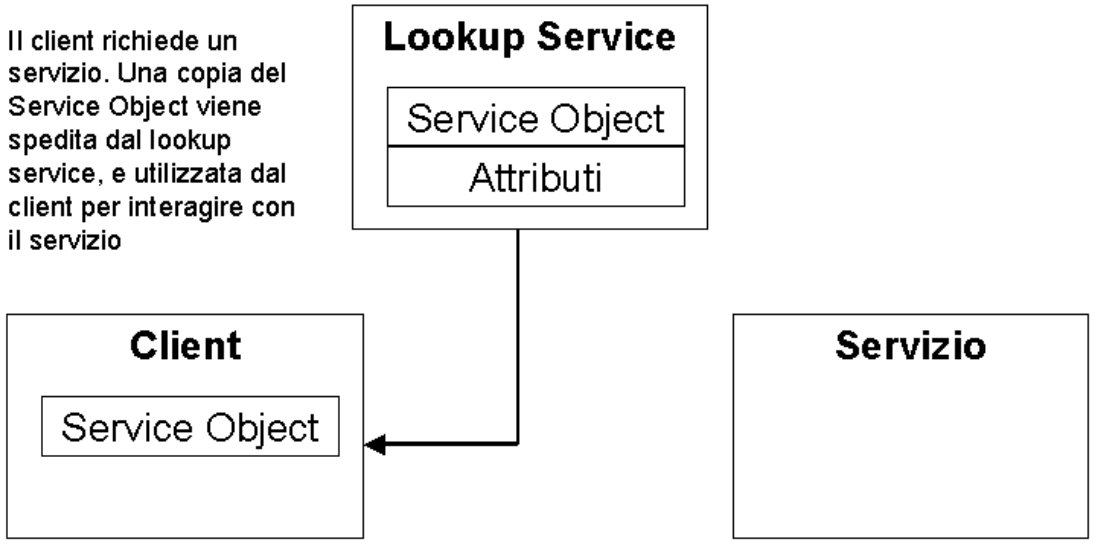

Figura 4.7: Lookup

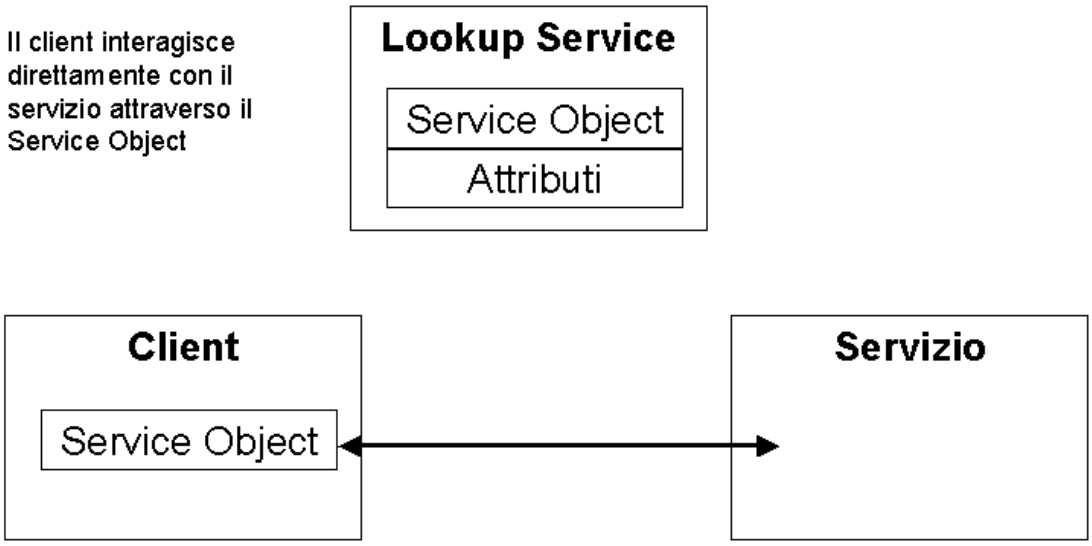

Figura 4.8: Utilizzo del servizio

I metodi presenti nel Service Object possono implementare un protocollo privato fra il client ed il fornitore del servizio; differenti implementazioni della stessa interfaccia possono usare protocolli di interazione completamente differenti.

Il client sa soltanto che sta utilizzando l'implementazione di un certa interfaccia scritta in Java. In questo modo il codice che realizza l'interfaccia può fare quello che serve per fornire il servizio senza che il client ne abbia la minima conoscenza.

Un'interfaccia utente può essere anch'essa memorizzata in un Lookup service, in questo modo un utente del sistema è in grado di usare il servizio direttamente.

4.4 Servlet

Un modo sicuro e facile per permettere a molti utenti di accedere a dati e risorse è l'accesso tramite client da Internet o Intranet. Questo tipo di acceso è basato sul protocollo HTTP e sul linguaggio HTML.

Le servlet non sono altro che una soluzione per rispondere a richieste HTTP.

4.4.1 Caratteristiche

Le servlet sono moduli (piccoli programmi Java indipendenti dalla piattaforma) che estendono i server orientati alla domanda/risposta, come ad esempio i web-server Java. Ad esempio una servlet può raccogliere dati tramite una form e aggiornare un database con questi dati.

Le servlet sono per i server, ciò che le applet sono per i browser [46]. A differenza delle applet, le servlet non hanno interfaccia grafica verso l'utente.

Le servlet sono un ottimo sostituto per gli script CGI, esse sono una via per generare documenti dinamici in modo facile e veloce; inoltre esse demandano il problema della programmazione dal lato server ad API specifiche di ogni piattaforma. Esse sono sviluppate con l'estensione standard di Java: Java Servlet API. Questo elimina anche tutta una serie di problemi che un programma crea, ad esempio di sicurezza.

Come detto, l'utilizzo primario è quello di gestire le richieste HTTP da parte di un client (attraverso web browser o altro programma per fare connessioni attraverso Internet), ma altri possibili utilizzi sono: permettere la collaborazione tra persone, infatti una servlet può gestire richieste concorrenti multiple e sincronizzarle (es. sistemi di conferenza

on-line); recapitare e reindirizzare richieste, infatti le richieste possono essere reindirizzate ad altre servlet o server.

4.4.2 Architettura

Il package `javax.servlet` fornisce le interfacce e le classi per scrivere servlet. L'astrazione centrale della Servlet API è l'interfaccia `servlet`. Tutte le servlet implementano questa interfaccia, direttamente, oppure più comunemente estendendo la classe che la implementa: `httpservlet`, come si vede nella figura 4.9.

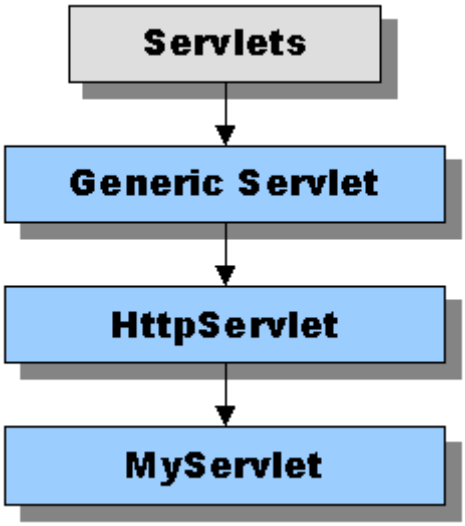

Figura 4.9: Ereditarietà servlet

L'interfaccia servlet dichiara, ma non implementa, i metodi per gestire le servlet e le loro comunicazioni con i client; chi scrive la servlet deve fornire alcuni o tutti questi metodi nello sviluppo.

4.4.2.1 Interfacce

Quando una servlet accetta una chiamata da un client, riceve due oggetti:

- `ServletRequest`, il quale incapsula la comunicazione dal client al server
- `ServletResponse`, il quale incapsula la comunicazione dal server verso il client

Questi due oggetti sono interfacce definite nel package `javax.servlet`

L'interfaccia ServletRequest permette accesso da parte della servlet a:

- informazioni come il nome dei parametri passati dal client;
- il protocollo (schema) in uso dal client;
- il nome dell'host remoto che ha fatto la richiesta;
- il nome del server che ha ricevuto la richiesta.
- Lo stream di input `ServletInputStream`: viene utilizzato per prendere dati dai client che usano protocolli HTTP POST e PUT.

L'interfaccia ServletResponse dà alla servlet i metodi per replicare al client. Essi:

- permettono alla servlet di settare il content lenght e il MIME type della risposta;
- forniscono lo stream di output `ServletOutputStream` e un oggetto `Writer` attraverso il quale la servlet può rispondere al client.

Le classi e le interfacce descritte sopra servono per una servlet "di base"; le servlet http hanno altri oggetti addizionali che provvedono alle capacità di controllo di sessione; lo scrittore di servlet può utilizzare queste API per mantenere lo stato tra una servlet e il client che si mantenga attraverso più connessioni (poiché il protocollo http è senza stato). Inoltre esistono API per i cookie, utilizzati per salvare dati del

client e per recuperarli successivamente. Da ultimo è possibile fare multithreading con le servlet.

4.4.2.2 Ciclo di vita

Il ciclo di vita di una servlet è semplice (si veda la figura 4.10):

1. la servlet viene inizializzata dal server quando viene invocata (il server chiama il metodo `init`)
2. la servlet gestisce zero o più richieste da parte di client
3. il server rimuove la servlet

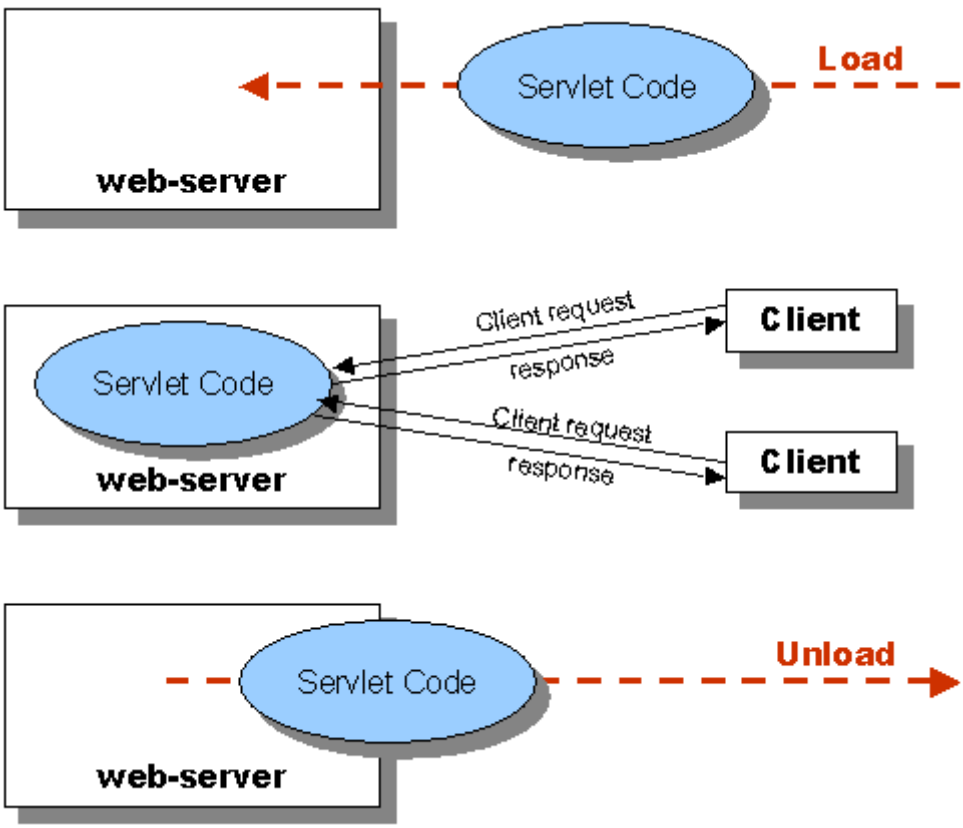

Figura 4.10: Ciclo di vita di una servlet

Di solito il passo 3 è fatto allo shutdown del server, in quanto accade raramente che una servlet venga distrutta mentre il server è ancora attivo (di solito questo avviene per ordine dell'amministratore del server).

Questo implica che una servlet attiva, mantiene memoria del proprio stato, ad esempio tramite le variabili globali, le quali non vengono reinizializzate fino a che la servlet è distrutta e poi ricaricata in memoria (o il server spento e riacceso). Questo significa che ogni campo di una classe servlet diventa un oggetto persistente, il che implica che è facile mantenere lo stato tra più richieste della stessa servlet, a differenza di un programma CGI, che deve scrivere i valori su disco, per mantenerli (con tutti i problemi che questo genera) [44].

4.4.2.3 Vantaggi

I vantaggi di utilizzare le servlet al posto di un normale meccanismo di estensione di server, sono i seguenti:

- sono più veloci dei programmi CGI, perché utilizzano un differente modello di processo;
- usano API standard che sono supportate da molti web-server;
- hanno tutti i vantaggi del linguaggio Java, compresa la facilità di programmazione e sviluppo e l'indipendenza della piattaforma;
- possono accedere ad un largo numero di API disponibili per la piattaforma Java.

4.5 Jess

Jess (Java Expert System Shell) [43] è una shell per sistemi esperti e un linguaggio di scripting scritto interamente in Java; permette quindi di richiamare funzioni Java direttamente dal Jess o di incapsulare il Jess all'interno di un programma scritto in Java.

Si tratta di un sistema molto simile al CLIPS (C Language Integrated Production System) lo strumento per la realizzazione di sistemi esperti sviluppato negli anni 80, presso il laboratorio Lyndon B. Johnson Space Center della NASA negli Stati Uniti.

In Jess, come in CLIPS, è possibile definire regole di produzione (dette anche più semplicemente *regole*), costituite essenzialmente da un insieme di *condizioni* e da una sequenza di *azioni*. Se le condizioni di una regola sono *soddisfatte*, essa può essere *eseguita*, e in questo caso verranno eseguite in sequenza le azioni in essa specificate.

I componenti principali sono:

- i fatti (facts);
- le regole (rules);
- il motore inferenziale (rules engine) o interprete.

L'insieme dei fatti e delle regole forma la base di conoscenza (Knowledge Base) del sistema esperto.

4.5.1 I fatti

I fatti sono uno dei modi per rappresentare informazione (conoscenza), ogni fatto rappresenta un pezzo di informazione che risiede nella lista dei fatti (fact-list).

Un fatto è un elenco di stringhe racchiuso tra parentesi e viene inserito nella fact-list con il comando assert:

```
(assert (fatto da asserire))
```

Ecco alcuni esempi di inserimento di fatti:

```
(assert (padre giovanni marco))
(assert (è uomo mario))
(assert (un uomo ha due gambe))
```

Quando un fatto viene inserito nella fact-list il sistema gli assegna un numero in modo da poterlo identificare univocamente (fact address):

```
<Fact-2>
```

Per eliminare un fatto si usa il comando retract indicando il suo identificatore numerico:

```
(retract 2)
```

I fatti possono essere di due tipi:

- ordinati (ordered): sono quelli che abbiamo visto precedentemente, consistono in un simbolo seguito da una sequenza di zero o più campi separati da spazi e racchiusi tra parentesi tonde.

In questi tipi di fatti l'ordine è fondamentale, infatti il fatto (Mario Rossi) è diverso dal fatto (Rossi Mario)

- non ordinati (unordered): non conta l'ordine, sono simili ai record dei linguaggi C e Pascal. Prima di poterli usare bisogna definire la propria struttura con il comando deftemplate:

```
(deftemplate persona          ;nome del template
  (slot nome                  ;nome del campo
    (type SYMBOL))            ;tipo del campo
  (slot cognome               ;nome del campo
    (type SYMBOL))            ;tipo del campo
  (slot annoNascita           ;nome del campo
    (type INTEGER))           ;tipo del campo
  (slot professione           ;nome del campo
    (type SYMBOL)             ;tipo del campo
    (default studente))       ;valore di default
```

Per inserire un fatto non ordinato si usa il comando assert nel seguente modo:

```
(assert persona (annoNascita 50) (nome Mario) (cognome
Rossi))
```

Una cosa importante da notare è che i fatti sono case-sensitive e quindi il fatto (padre giovanni marco) è diverso da (padre Giovanni Marco).

4.5.2 Le regole

Affinché un sistema si possa definire esperto, è necessario fornirlo di regole che possano produrre conoscenza a partire dai fatti. Esse specificano un insieme di azioni che devono essere eseguite in una data situazione.

Una regola è definita con il comando:

```
(defrule NomeDellaRegola  “commento”
  (pattern_1)
  (pattern_2)
  . . .
  (pattern_N)
=>
  (action_1)
  . . .
  (action_M)
)
```

La parte precedente allo speciale simbolo => (uguale maggiore) si chiama antecedente (LHS: Left Hand Side), mentre quella che viene dopo si chiama conseguente (RHS: Right Hand Side). L'antecedente è un insieme di condizioni (pattern) che devono essere tutte soddisfatte (AND) perché la regola sia applicabile, mentre il conseguente è composto da un insieme di azioni che vengono svolte. È compito del motore inferenziale verificare le condizioni ed eseguire le opportune azioni.

Un esempio di regola è quella che trova i nonni a partire da una lista dei fatti nel quale sono indicati i padri delle varie persone. Decidiamo di indicare che “Mario è il padre di Luca” con il fatto ordinato (padre mario luca).

Chi è nostro nonno? Nostro nonno è il padre di nostro padre, quindi se x è il padre di y e y è il padre di z allora x è il nonno di z, in Jess definiamo questa regola:

```
(defrule R_nonno "regola che trova i nonni"
  (padre ?x ?y)
  (padre ?y ?z)
=>
  (assert (nonno ?x ?z))
)
```

Quindi se nella fact-list sono presenti i seguenti fatti:

```
(padre mario luca)
(padre luca alberto)
```

la regola R_nonno scatterà con l'assegnazione: ?x=mario ?y=luca ?z=alberto

l'esecuzione della regola, che avviene quando viene dato il comando (run), provoca l'inserzione nella fact-list del fatto:

```
(nonno mario alberto)
```

4.5.3 L'interprete

Il funzionamento dell'interprete segue il ciclo *riconoscimento-azione* tipico dei sistemi a regole di produzione. A grandi linee il ciclo è strutturato come segue:

1. *riconoscimento*: si confrontano le condizioni di tutte le regole di produzione con tutti i fatti contenuti nella base dei fatti; per ogni regola, le cui condizioni sono soddisfatte, si inserisce la corrispondente attivazione nell'agenda (una regola può dar luogo anche a più attivazioni);
2. *azione*: se l'agenda è vuota, l'esecuzione termina; altrimenti si preleva l'attivazione in testa all'agenda e la si esegue

(eseguendo in sequenza le azioni specificate dalla regola corrispondente);

3. si ritorna al passo 1.

L'inserimento delle attivazioni nell'agenda avviene secondo una strategia ben definita (depth, breadth, lex, mea, simplicity, complexity e random); inoltre il programmatore può influire direttamente sull'inserimento definendo la priorità (salience) di una regola. Il ciclo riconoscimento-azione ha inizio quando si sottopone all'interprete Jess il comando (run).

4.5.4 Utilizzare Jess all'interno di un programma Java

La classe fondamentale è `jess.Rete` che implementa il motore inferenziale. Ogni oggetto `jess.Rete` ha la propria base di conoscenza, l'agenda, le regole, ecc.; per incapsulare Jess all'interno di Java basta semplicemente creare un oggetto `jess.Rete` e manipolarlo.

```
Rete r = new Rete();
r.executeCommand("(assert (padre mario luca))");
r.run();
```

Per la comunicazione dal motore inferenziale verso la classe Java, si utilizza la funzione Jess `call`. Per prima cosa è necessario collegarsi alla classe Java, tramite la funzione `bind`; successivamente con la `call` è possibile richiamare i metodi di tale classe e passare gli eventuali parametri che sono richiesti.

```
(bind ?classe (new nome_classe_Java))

(defrule nonno
     (nonno ?x ?z)
     =>
     (call ?classe stampa_nonno ?x ?z)
)
```

4.6 Jini e le differenti versioni di Java

In questa sezione diamo una breve indicazione di come un servizio Jini può essere realizzato su dispositivi (*device*) limitati, dal punto di vista della potenza di calcolo e della memoria.

4.6.1 Le versioni di Java

Un sistema Jini per funzionare deve avere a disposizione una Java Virtual Machine (JVM) in grado di eseguire le classi Jini. Esistono diverse soluzioni a seconda delle potenzialità del dispositivo, infatti, la piattaforma Java 2 non è distribuita dalla Sun Microsystems in un'unica versione, ma ci sono diverse edizioni (*editions*), configurazioni (*configurations*) e profili (*profiles*).

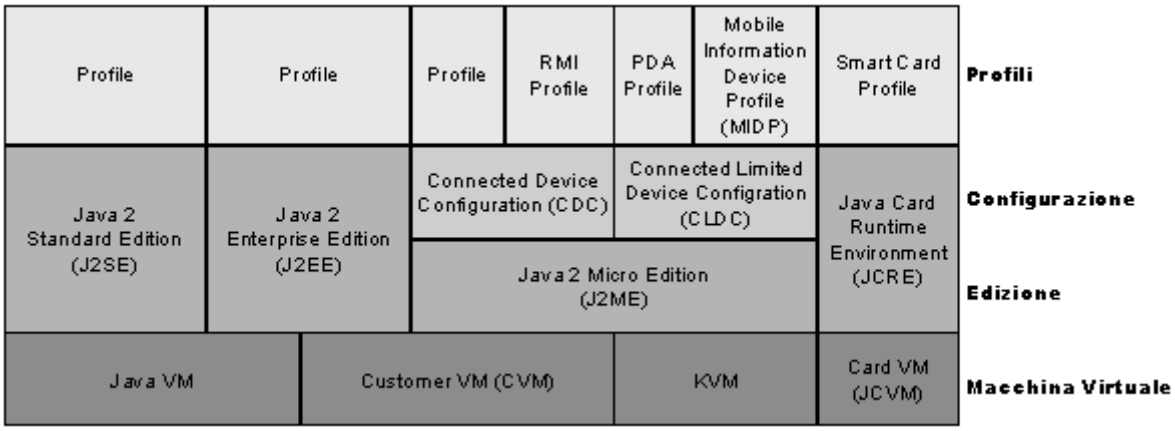

Figura 4.11: Edizioni, configurazioni e profili di Java

Nella parte inferiore della figura 4.11 sono riportate le JVM, che vengono eseguite sopra al sistema operativo del device. La KVM è una

virtual machine progettata appositamente per i piccoli device con limitata memoria.

Le tre edizioni più importanti di Java 2 sono Java 2 Enterprise Edition (J2EE), Java 2 Standard Edition (J2SE) e Java 2 Micro Edition (J2ME). Un'edizione definisce le classi API che il programmatore può utilizzare, J2SE include un sottoinsieme delle API di J2EE, e J2ME ne ha ancora meno; in realtà non è del tutto vero, perché un'edizione più limitata può avere delle classi specifiche che non sono supportate da un'edizione più estesa.

L'edizione più interessante per i device è la J2ME, essendo stata sviluppata appositamente per tali scopi.

Il livello successivo è chiamato configurazione, attualmente ne esistono due tipi per J2ME: Connected Device Configuration (CDC) e Connected Limited Device Configuration (CLDC). Una configurazione specifica una API, che il device che supporta tale configurazione deve implementare.

Sun ha anche specificato differenti profili al di sopra del livello di configurazione, un profilo definisce classi supplementari che possono essere usate dal device. Per esempio, se sulla configurazione CDC si vuole utilizzare RMI, bisogna aggiungere il profilo RMI, mentre sulla configurazione CLDC non si può eseguire Jini perché non si può utilizzare RMI e serialization.

4.6.1.1 Java 2 Enterprise Edition (J2EE)

Questa edizione è destinata ad applicazioni "pesanti" ed è perfetta per utilizzare servizi Jini; i servizi possono rispondere a diversi utilizzatori e possono utilizzare caratteristiche avanzate del linguaggio. Non è invece adatta per dispositivi limitati, dal momento che richiede molta memoria.

4.6.1.2 Java 2 Standard Edition (J2SE)

Questa è la versione standard di Java, destinata per i normali PC. Questa edizione supporta completamente Jini, ma ancora necessita di troppa memoria per essere utilizzata su device limitati.

4.6.1.3 Java 2 Micro Edition (J2ME)

Questa versione è stata progettata per funzionare su dispositivi con limitate risorse ed esiste nelle due configurazioni viste precedentemente: CDC e CLDC. CLDC è eseguita su KVM, una virtual machine appositamente realizzata per occupare poca memoria, mentre CDC, più potente di CLDC, utilizza la Customer Virtual Machine (CVM).

Jini per funzionare ha bisogno di Java 2, infatti, tra l'altro, il Lookup necessita del `java.rmi.MarshalledObject`, che è una caratteristica introdotta per la prima volta in Java 2 e che permette di impacchettare un oggetto serializzandolo. Inoltre Jini utilizza alcune classi del package `java.security` che non sono disponibili in JDK 1.1.x.

Il punto critico nello sviluppo di servizi Jini su device limitati è la JVM, che deve essere abbastanza potente per eseguire le classi Jini.

4.6.2 Soluzioni hardware

Un diverso approccio è di montare un chip in grado di eseguire direttamente in hardware il Java bytecode. Alcuni esempi sono:

- PicoJava: realizzato da Sun Microsystems;
- aJ-100: prodotta da aJile Systems è compatibile con Jini;
- Xpresso: progettato da Zuccotto Systems è in grado di integrare BlueTooth, Java e Jini.

Questo approccio alternativo apre nuove possibilità nella costruzione di apparecchi, ma richiede che il dispositivo sia dotato di questo chip.

5

ARCHITETTURA LOGICA DEL SISTEMA

"Io sono un cervello, Watson.
Il resto di me non è che una semplice appendice."
Sherlock Holmes

5.1 Premessa

Nel capitolo 2 abbiamo visto il punto di partenza per la nostra agenzia domotica, nel 3 la teoria alla base, nel 4 le tecnologie di cui ci siamo avvalsi, i "mattoni" fondamentali; in questo capitolo descriviamo l'architettura del nostro sistema. Nella sezione 2 analizziamo analizza da un punto di vista logico, partendo dal problema, per arrivare alla soluzione concettuale da noi adottata. Nella sezione 3 illustriamo il progetto di agenzia ideale, sul quale abbiamo costruito il nostro sistema, riguardante la gestione di allarmi e scenari, il telecontrollo, il

monitoraggio e la programmazione delle azioni, che viene analizzata in dettaglio nel capitolo 7.

5.2 Progetto logico

Prendendo come punto di partenza tutto quanto descritto nel capitolo 2, si deduce che la costruzione di un'agenzia domotica prevede una parte di gestione "intelligente" della casa nel suo complesso, la possibilità di monitorare lo stato e di telecontrollare ogni singolo apparecchio presente.

Per quanto riguarda la prima parte, una delle problematiche maggiori è che gli elettrodomestici, visti come agenti, devono anche lavorare in maniera tradizionale, interagendo con l'uomo, quindi cambiare i propri piani di conseguenza.

Per quanto riguarda la complessità, si passa da un problema di tipo "pulisci questa stanza" a una classe superiore di problemi del tipo "gestisci la casa", che è apparentemente molto più difficile. In pratica, una volta individuate dall'uomo-progettista della base di conoscenza le problematiche da risolvere, e stabilito come gestirle, la classe di difficoltà si abbassa.

Il monitoraggio della casa deve poter avvenire sia quando l'uomo è fisicamente presente, sia, soprattutto, quando è assente, tramite vari mezzi di comunicazione.

Per quanto riguarda il telecontrollo valgono gli stessi concetti visti per il monitoraggio, con in più la possibilità di interagire attivamente sullo stato di ogni singolo dispositivo presente nella casa.

5.2.1 Concetti

Si configura quindi un'agenzia domotica composta da 3 parti: gestione, monitoraggio e telecontrollo. Questi sottosistemi del più grande sistema che è l'agenzia domotica (si veda la figura 5.1), sono complementari (ad esempio, dove non può intervenire il sottosistema di gestione, esso può richiedere l'intervento umano, anche tramite telecontrollo) e non in competizione tra loro (gli obiettivi sono comuni).

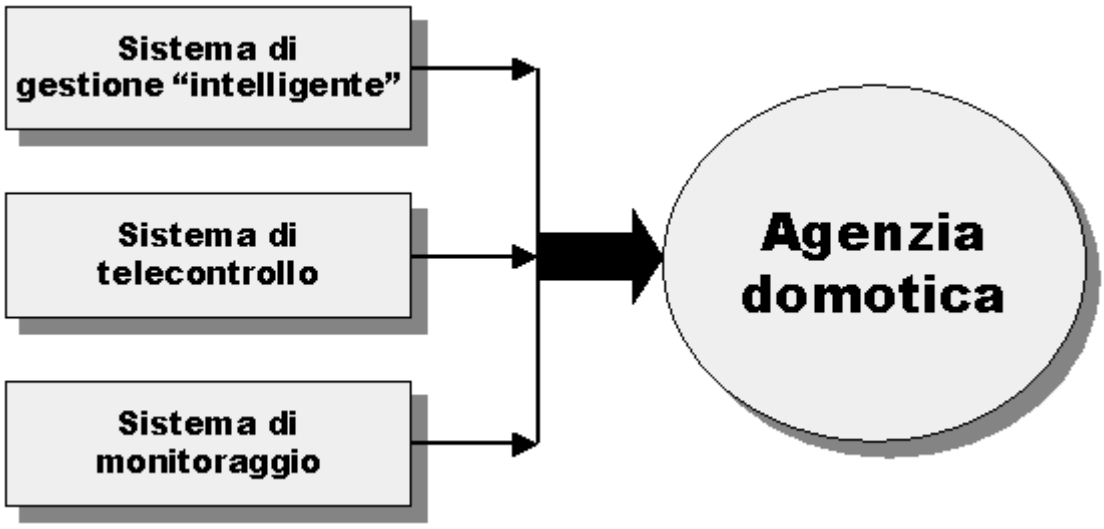

Figura 5.1: Sistemi che compongono l'agenzia domotica

Per la gestione della casa, non si tratta solo di avere un problema complesso e di scomporlo, di risolvere i sottoproblemi e di generare il risultato, ma l'agenzia in questione deve essere sempre attiva 24 ore al giorno, lasciar gestire la casa anche all'uomo e intervenire (avvisando eventualmente) solo quando necessario.

Quindi l'agenzia rappresenta proprio la parte di bipolo uomo-macchina, che interviene nella gestione della casa, quando il polo uomo-corpo non può.

Questa limitazione è data da un'incapacità fisica del polo uomo-corpo di eseguire i compiti: per inabilità o handicap, per assenza dal luogo fisico della casa, oppure perché impegnato in altre occupazioni.

La parte di monitoraggio deve potersi avvalere di diversi mezzi di comunicazione in modo che l'utente, in qualunque situazione si trovi, possa avere un'informazione in tempo reale sullo stato della casa.

I possibili media da noi ritenuti più opportuni, tenendo conto dell'evoluzione tecnologica che negli ultimi anni sta cambiando il nostro modo di vivere, sono Internet (sia web che posta elettronica) e il telefono sia fisso che mobile.

Gli stessi canali di comunicazione sono utilizzati in modo bidirezionale per la parte di telecontrollo.

I tre sistemi, pur avendo nature diverse, sono tra loro simili e confluiscono tutti nella costruzione dell'agenzia domotica; non si può quindi pensare di realizzare tre sistemi tra loro completamente separati, ma bisogna adottare una visione complessiva, infatti devono tutti poter avere una conoscenza della casa, in particolare sapere quali dispositivi sono funzionanti e i servizi da loro offerti.

5.2.2 Soluzione teorica

Come detto nel capitolo 3, nella definizione di un architettura per un'agenzia, per prima cosa bisogna sviluppare un modello concettuale per gli agenti e per la loro comunicazione.

La migliore soluzione concettuale, a nostro avviso, è quella dove esiste un agente supervisore, depositario della base di conoscenza, che inferisce

e distribuisce i compiti agli altri agenti. Ogni agente, per risolvere la propria parte di piano, si avvarrà della collaborazione degli altri agenti di cui ha bisogno. Si ha quindi un'agenzia collaborativa, la cui unica possibile competizione sta nell'accesso a risorse comuni, che poi sono rappresentate da altri agenti. Si configura quindi una comunità (o federazione) dove tutti gli agenti cooperano e un'entità superiore che controlla, distribuisce i compiti, coordina i risultati. Gli agenti devono essere distribuiti nello spazio e possono appartenere fisicamente a macchine diverse, ma devono pur sempre cooperare tra loro.

Questo, a nostro parere è il miglior modello, perché è quello che più "naturalmente" si avvicina alla realizzazione di un polo uomo-macchina per la gestione della casa.

L'organizzazione tra i nostri agenti è di tipo verticale, per l'esistenza di un master supervisore, e dinamica, perché la tipologia e il numero di agenti della comunità non è prefissata, ma può variare nel tempo.

I nostri agenti operatori sono specializzati, in quanto ciascuno di essi offre dei servizi ben definiti, infatti in una prima fase, questi agenti devono comunicare quali tipi di compiti essi sono in grado di svolgere e devono iscriversi alla federazione. Successivamente, quando ci sarà bisogno di loro, avverrà la vera e propria fase di reclutamento, nella quale il supervisore e gli stessi agenti utilizzeranno i servizi offerti da tutti gli agenti, e quindi dalla comunità.

In ogni momento deve essere possibile per un agente staccarsi (ovviamente dopo aver eseguito l'operazione corrente) o riagganciarsi alla comunità: da qui l'organizzazione dinamica.

In figura 5.2 sono mostrate le operazioni di iscrizione e di abbandono di un agente dalla federazione.

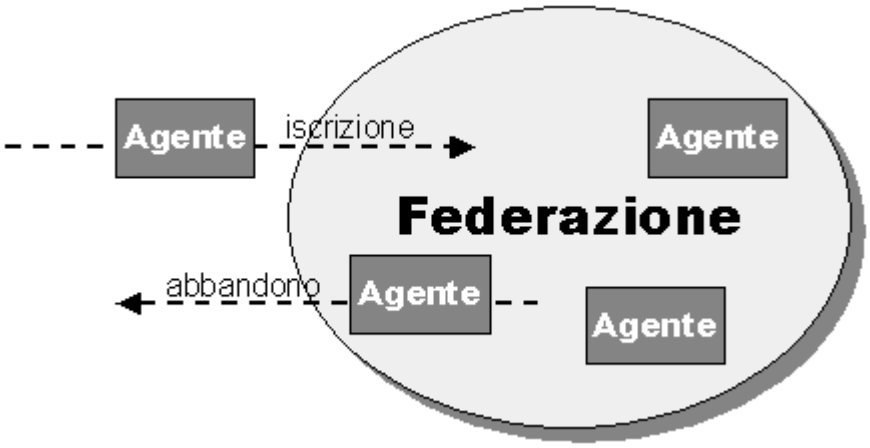

Figura 5.2: Iscrizione e abbandono di un agente alla federazione

La fase successiva è quella di definire la comunicazione tra gli agenti. Andando ad analizzare il problema ad un livello basso, si nota che gli agenti devono tutti accedere ad una rete di comunicazione comune, quindi anche se locati fisicamente su macchine diverse, ci deve essere una perfetta e completa trasparenza della rete, in modo che tutta la comunità sia accessibile da tutti gli agenti.

La comunicazione avviene in due modi: richiesta-risposta oppure tramite generazione di eventi (si veda la figura 5.3).

La comunicazione di tipo richiesta-risposta è usata comunemente tra l'agente supervisore e gli agenti operatori, e tra gli agenti operatori stessi. Classico esempio è quello dove l'agente supervisore assegna un compito ad un agente, il quale per poterlo eseguire ha bisogno anche dei servizi di un altro agente: tutte le comunicazioni in questo caso saranno di tipo richiesta-risposta.

La generazione di eventi è utilizzata invece dagli agenti quando vogliono comunicare qualcosa verso l'agente supervisore (esempio: un sensore ha rilevato una fuga di gas). Si profila quindi la figura di un nuovo agente, che sta sempre in ascolto di eventi e li comunica all'agente

supervisore, in modo che l'agente supervisore possa identificare il problema e costruire il piano per risolverlo.

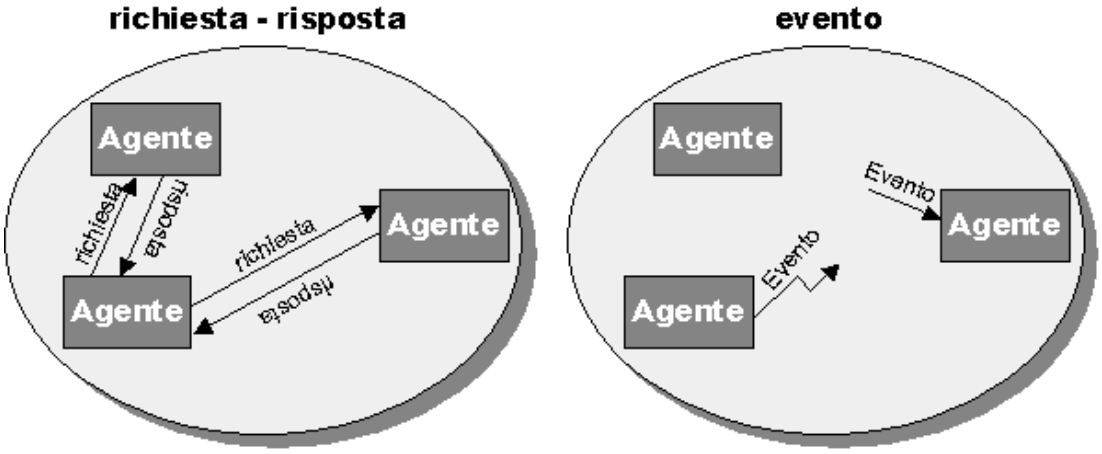

Figura 5.3: Comunicazione fra gli agenti

Organizzando la comunicazione ad eventi, non è necessario che un qualsiasi agente sappia che esiste un supervisore e che deve informare lui di qualsiasi situazione rilevante, basta dire all'agente di mandare sulla rete un evento, ci penserà poi l'agente sempre in ascolto a ridirigerlo al supervisore.

Inoltre, in questo modo, si rende pubblica a tutti la conoscenza di ogni singolo agente, cosicché qualunque altro agente interessato, può ascoltare e agire di conseguenza.

5.3 Agenzia Ideale

Riassumiamo quindi le caratteristiche della nostra agenzia domotica definita fino adesso, supponendo di avere a disposizione agenti ideali.

Agenti operatori [39]:

a) Sensori di gas, acqua, fumo, temperatura, luce e movimento, sistemati, possibilmente, in ogni stanza;
b) Attuatori ad ogni porta, finestra, porta-finestra, tapparella, tenda e porta d'ingresso;
c) Valvole di controllo per l'impianto idraulico e gas;
d) Interruttori per l'impianto elettrico;
e) Termostati programmabili;
f) Elaboratori individuali, comprensivi di allacciamento alla rete Internet;
g) Telefoni, segreterie telefoniche e fax;
h) Televisione (anche via cavo e via satellite), VCR e Hi-Fi;
i) Elettrodomestici: cucina, frigorifero, lavatrice, lavastoviglie, forno, ecc.;
j) Temporizzatori dei dispositivi domestici;

Poi sono presenti: un agente supervisore, che è anche depositario della base di conoscenza, l'agente "ascoltatore" di tutto ciò che avviene tra gli agenti operatori, gli agenti che si occupano della comunicazione con l'uomo via web, e-mail e SMS; l'agente che si occupa dell'interfaccia con l'utente (possibilmente grafica, di facile comprensione e utilizzo).

Tutti gli agenti possono generare eventi quali, il collegamento o scollegamento alla federazione, il malfunzionamento o la rottura. Poi gli agenti di tipo a), e), f) e i) possono generare eventi speciali, di rilevante importanza per la gestione della casa.

Tutti questi dispositivi, che singolarmente non rappresentano niente di innovativo, devono unirsi per far fronte alle esigenze vecchie e nuove del padrone di casa.

Il cambiamento dello stile di vita negli ultimi anni ha portato alla "creazione" di nuove richieste da parte dei consumatori. La donna che

lavora, il diffondersi dei singles, l'aumento della vita media, lo sviluppo della "seconda casa", l'accresciuto benessere della popolazione, la maggiore permanenza fuori casa, il crescere della criminalità (tanto per citarne alcuni) hanno portato all'esigenza di una migliore qualità della vita, un'abitazione più sicura, più confortevole e più facile da gestire e da controllare sia dentro la casa stessa, che fuori.

Dall'analisi fatta, si individua una serie di possibili funzioni da pianificare:

- Clima
    - Programmazione del clima
    - Controllo delle temperature
    - Controllo dei consumi (Rilevazioni statistiche e autoadattamento)
- Energia
    - Controllo linee
    - Controllo sovraccarichi di tensione, picchi di corrente
    - Controllo dei consumi (Rilevazioni statistiche e autoadattamento)
    - Controllo dell'alimentazione di emergenza
- Sicurezza
    - Controllo perimetrale, interno, di accesso
    - Sistema antincendio, antiallagamento, anti fuga di gas
- Gestione degli elettrodomestici
- Gestione comunicazioni.

Queste funzionalità possono essere svolte automaticamente dal sistema di gestione della casa, ma deve sempre essere possibile l'intervento manuale da parte dell'uomo, sia quando è fisicamente presente, tramite interruttori multifunzionali situati in ogni stanza, oppure

tramite un elaboratore centrale, sia quando è assente, tramite il telecontrollo a distanza, utilizzando vari media.

Come già detto, la nostra parte implementativa tratta la sicurezza, la gestione degli elettrodomestici, delle comunicazioni e la programmazione temporizzata.

Si riporta nella figura 5.4 lo schema dell'architettura logica del sistema così come è stato definito in questo capitolo (i blocchi tratteggiati sono definiti da altri e da noi utilizzati).

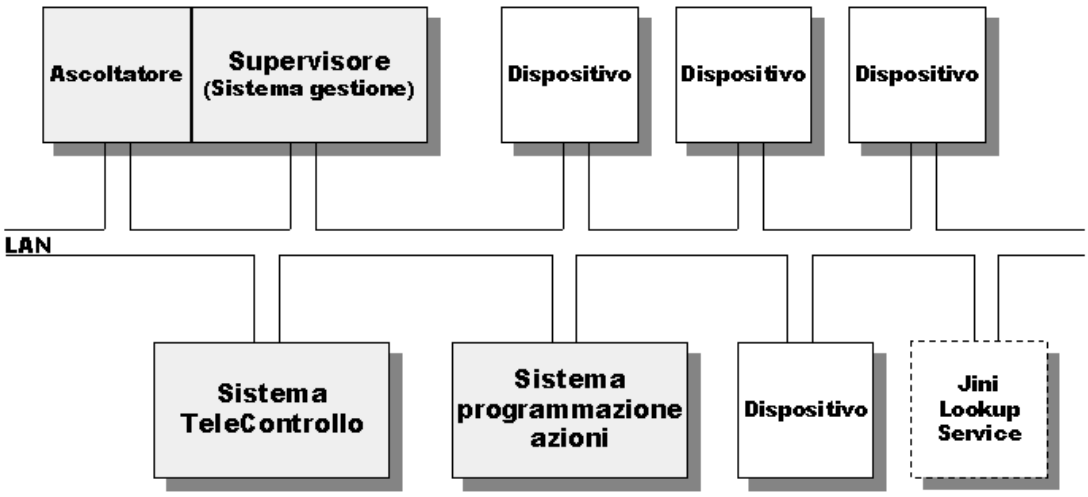

Figura 5.4: Architettura dell'agenzia

6

PROGETTO DELL'AGENZIA DOMOTICA

"Anyone who has never made a mistake
has never tried anything new."
Albert Einstein

6.1 Premessa

Nel passaggio naturale da progetto logico a fisico, la scelta migliore per ottenere la struttura descritta nel capitolo precedente, è stata, secondo il nostro punto di vista, l'utilizzo della tecnologia Jini (per una trattazione dettagliata rimandiamo al paragrafo 4.3) per la costruzione della federazione e dell'infrastruttura delle comunicazioni, e di Jess (si veda il paragrafo 4.5) per l'attività inferenziale svolta dall'agente supervisore.

6.2 I dispositivi domestici

I dispositivi presenti in una casa hanno natura e funzionamenti completamente diversi fra loro; abbiamo deciso di raggrupparli in famiglie, contraddistinte da caratteristiche e operazioni comuni. Tale suddivisione ci ha permesso di definire lo schema di interfacce fra loro collegate (ereditarietà), riportato in figura 6.1, nel quale sono indicate le operazioni (metodi) generali che i dispositivi della famiglia sono in grado di eseguire. Scendendo lungo l'albero di ereditarietà si aumenta il livello di specializzazione, indicando quali sono le nuove operazioni disponibili.

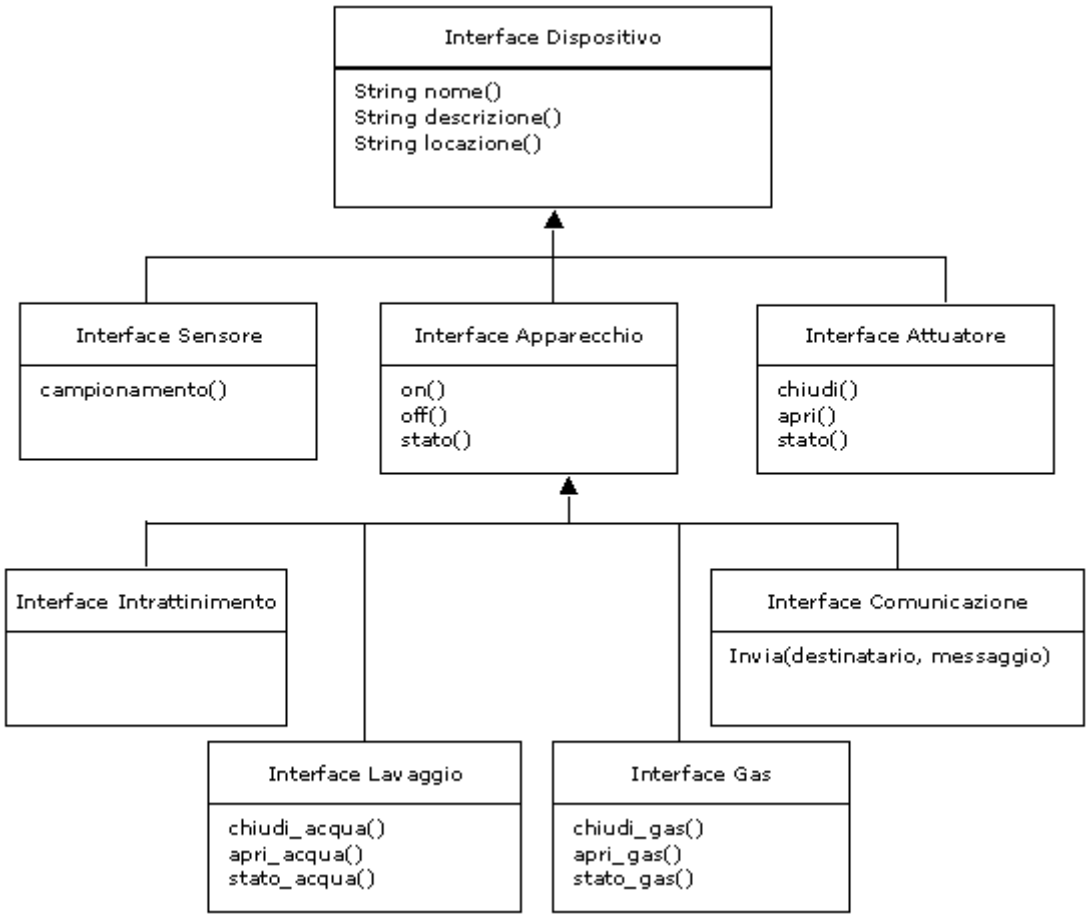

Figura 6.1: Suddivisione in famiglie dei device domestici

Abbiamo chiamato Dispositivo la famiglia di più alto livello, e che, quindi, comprende tutti i device domestici. Questa famiglia definisce le operazioni generali che tutti dovranno poi implementare, quali: il nome, la locazione, descrizione del dispositivo.

Le tre sottofamiglie più specializzate di Dispositivo sono: Sensore, Apparecchio e Attuatore.

Sensore è un dispositivo che traduce un fenomeno fisico in un segnale elettrico e quindi aggiunge l'operazione di campionamento di tale fenomeno.

Attuatore è un dispositivo che traduce un segnale elettrico in un fenomeno fisico, ha le operazioni di chiusura, apertura e stato dell'attuatore.

Apparecchio è un dispositivo elettrico che può essere acceso, spento e di cui si può conoscere lo stato. Esso può eseguire un compito specifico, e quindi abbiamo individuato le seguenti sottofamiglie: Intrattenimento, Lavaggio, Gas e Comunicazione.

Intrattenimento è un apparecchio per lo svago dell'utente.

Lavaggio è un apparecchio collegato all'impianto idraulico di cui si può controllare la valvola di collegamento alla rete idraulica (chiudi, apri, stato).

Gas è un apparecchio collegato alla rete di distribuzione del gas, del quale si può controllare la valvola di accesso al metano (chiudi, apri, stato).

Comunicazione è un apparecchio che permette di inviare un messaggio ad un destinatario.

Si noti come qualunque dispositivo elencato nel paragrafo 5.3 si può catalogare nello schema da noi adottato, inoltre dallo schema di partenza si arriva ad uno schema di ereditarietà tra classi Java come si può vedere nel paragrafo 7.2.

Ogni nuovo dispositivo che si collega alla nostra comunità, ricade in una delle famiglie definite e deve implementare le operazioni di quella famiglia, in modo che altri agenti che lo vogliano utilizzare sappiano quali operazioni esso è in grado di svolgere e come le devono utilizzare. Quindi, questa suddivisione permette di aggiungere all'agenzia domotica nuovi dispositivi, senza che sia necessario modificare gli altri agenti.

Ovviamente ogni dispositivo potrà essere in grado di svolgere altre operazioni specifiche. Si prenda come esempio una lavastoviglie: ricade sicuramente nella famiglia Lavaggio e quindi deve necessariamente definire le operazioni di nome, descrizione, locazione, accensione, spegnimento, stato elettrico, apertura, chiusura e stato del rubinetto dell'acqua, ma può anche realizzare altri tipi di funzionalità specifiche del dispositivo in questione, quali, ad esempio, scelta del tipo di lavaggio, della temperatura dell'acqua, stato del sale e del brillantante.

La suddivisione in famiglie qui riportate è frutto di una nostra scelta, quindi potrebbe essere necessaria una maggiore specializzazione, introducendo nuove tipologie di dispositivi e di conseguenza definendo per esse le operazioni possibili.

6.3 L'agenzia

Nella fase iniziale della costituzione dell'agenzia, è necessario, prima di tutto, definire la rete di comunicazione e i meccanismi di comunicazione su di essa, poi si forma la federazione Jini, alla quale si iscrivono uno ad uno tutti i dispositivi e gli agenti con funzioni speciali.

6.3.1 La comunicazione

La comunicazione avviene tramite i metodi definiti in 5.2.2: per quanto riguarda la richiesta-risposta, questo tipo di messaggi viaggia sulla rete definita da Jini, dal mittente verso tutti i destinatari interessati; per quanto riguarda la generazione di eventi, anche questi viaggiano sulla rete e tutti possono ascoltarli, ci pensa l'agente "ascoltatore" a ridirigere i messaggi destinati all'agente supervisore, il quale ha il compito, eventualmente, di prendere le opportune decisioni che possono rendere necessarie altre comunicazioni di tipo richiesta-risposta verso altri agenti.

L'agente supervisore ha anche il compito di reclutare i vari agenti che si rendono necessari per la risoluzione del problema complessivo, suddividendolo in sottoproblemi parziali che i singoli agenti sono in grado di risolvere. Il reclutamento è quindi una scelta, fatta durante l'esecuzione, del sottoinsieme dei dispositivi domestici attivi, tale per cui sia possibile portare a compimento il piano di risoluzione del problema. Nel nostro caso, i fattori che sono considerati nella fase di reclutamento sono:

- famiglia di appartenenza del dispositivo
    - comunicazione
    - intrattenimento
    - sensore
    - attuatore
    - lavaggio
    - gas
    - apparecchio
- locazione fisica
    - piano
    - stanza
- tipologia specifica del dispositivo

6.3.2 Gli allarmi

Come visto, un'importante sezione del sistema riguarda la gestione degli allarmi. Le situazioni potenzialmente pericolose possono essere riferite a: fughe di gas, perdite d'acqua, presenza di fumo e intrusione.

I primi tre tipi di allarme vengono gestiti a livelli: abbiamo deciso di dare per ogni allarme quattro livelli (rilevazioni temporali successive), a ognuno dei quali si associano azioni diverse da compiere. Ad un primo livello, si registra l'allarme e non si compie nessuna azione; alla successiva rilevazione temporale dell'allarme, si incominciano a prendere automaticamente le prime contromisure, al terzo livello si prendono contromisure più decise e si informa il padrone di casa, dell'allarme in corso, al quarto livello, si richiede l'aiuto di personale esterno, come i vigili del fuoco e la sicurezza. Abbiamo deciso di introdurre la struttura a livelli per evitare che situazioni temporanee siano considerati come veri e propri allarmi. Ad esempio, quando viene accesa la piastra di cottura, il sensore della cucina potrebbe rilevare la presenza di gas nell'aria e quindi generare un evento di tipo fuga di gas; nel nostro caso questo viene rilevato come un evento singolo e quindi interpretato come un allarme gas di livello 1 e, giustamente, nessuna operazione viene compiuta.

Il quarto tipo di allarme, l'intrusione, viene gestito in modo da bloccare il "ladro" nella stanza dove si trova, in modo che non sia libero di girare per la casa, avvisando nel contempo la sicurezza per richiedere un suo intervento.

Anche il controllo della temperatura può essere ricondotto agli allarmi; temperature troppo alte o troppo basse, possono essere indici di malfunzionamento, oltre che portare disagi a chi vive in casa; in questo caso si cerca di riportare la temperatura ad un livello prefissato di benessere.

Si noti che la gestione degli allarmi a livelli porta ad avere un reclutamento incrementale, nel senso che salendo di livello aumenta la cardinalità dell'insieme degli agenti reclutati. Questo perché, con la nostra gestione, più il livello è alto, più l'allarme è considerato rilevante e quindi un maggior numero di dispositivi sono coinvolti nel tentativo di risoluzione di tale situazione.

6.3.3 Scenari

In aggiunta agli allarmi, l'agente supervisore gestisce anche una serie di scenari di utilizzo della casa. Tipico scenario è l'uscita del padrone dalla casa; in questo caso si spengono tutte le luci, si abbassano le tapparelle, si chiudono le porte, si attiva il sistema di allarme e si abbassa la temperatura, per attuare una politica di risparmio energetico. Al rientro si ristabiliscono le condizioni climatiche presenti prima dell'uscita e si stacca il sistema di allarme antintruso.

Un altro scenario è il controllo delle luci: infatti se una persona entra in una stanza buia, per prima cosa vengono alzate le tapparelle, se l'illuminazione è ancora insufficiente, si procede all'accensione delle luci presenti nel locale.

6.4 Mezzi di comunicazione

Per dare all'utente la possibilità di monitorare e telecontrollare la propria abitazione a distanza e di ricevere la comunicazione degli allarmi, quando essi avvengono, è ovviamente necessario usufruire di un mezzo di comunicazione. Quelli da noi utilizzati sono: Internet e il telefono cellulare.

Sfruttando Internet, si sono create nuove funzionalità: le più importanti per il nostro scopo sono il web e la posta elettronica.

Un web browser, che richiede l'utilizzo di un elaboratore personale, consente un accesso ad interfaccia grafica, quindi di gradevole presentazione e di facile comprensione nell'utilizzo. Inoltre il browser è ormai uno strumento famigliare e diffuso per il lavoro e lo svago di molte persone, quindi l'utente è già portato "naturalmente" ad usarlo.

La posta elettronica, che è possibile inviare da una vasta gamma di apparecchi elettronici, come un elaboratore, un PDA o un telefono GSM, prevede, per il monitoraggio e il controllo, dei comandi impartiti tramite linea di testo, che quindi è di più difficile utilizzo e apprendimento, ma che, grazie alla varietà dei dispositivi utilizzabili, permette una maggiore flessibilità di gestione, in varie situazioni. La posta elettronica è inoltre utilizzata per la spedizione di avvisi di allarme, agli indirizzi conosciuti del dominus, quando egli è fuori casa.

Anche il telefono cellulare è utilizzato per l'invio di allarmi, i quali arrivano immediatamente a destinazione; quindi si raggiunge il padrone di casa in ogni punto del globo si trovi, tramite SMS, messaggi brevi di testo.

7

REALIZZAZIONE SPERIMENTALE

“Let's see what's out there. Engage!”
Captain Jean-Luc Picard

7.1 Premessa

In questo capitolo analizziamo la nostra realizzazione sperimentale: la gestione degli allarmi e di alcuni scenari di utilizzo, il monitoraggio e telecontrollo dei dispositivi domestici e la programmazione domestica temporizzata. Per l’installazione e l’uso del sistema si rimanda all’appendice B.

Analizzeremo dapprima il sistema nel suo complesso, come abbiamo realizzato la nostra agenzia, poi i singoli moduli con le loro funzioni e brevi listati di esempio. Per una trattazione più completa del codice sorgente vedere l’appendice C.

Nel paragrafo 2 mostriamo come sono stati organizzati i componenti riguardanti i dispositivi domestici; nel paragrafo 3 illustriamo il sottosistema di monitoraggio e telecontrollo nelle sue tre versioni: programma Java, web, e-mail; nel paragrafo 4 illustriamo il sottosistema di gestione realizzato; nel paragrafo 5 mostriamo il sottosistema di programmazione temporizzata degli agenti.

7.2 I dispositivi domestici

I dispositivi domestici sono stati suddivisi in due parti (si veda la figura 7.1), quella cooperativa e quella operativa, seguendo la teoria delle agenzie presentata nel capitolo 3.

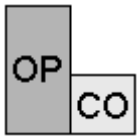

Figura 7.1: Parte cooperativa (CO) e operativa (OP) di un dispositivo

7.2.1 Parte CO (cooperativa)

Per realizzare la parte cooperativa abbiamo utilizzato la tecnologia fornita da Jini; per prima cosa abbiamo suddiviso i dispositivi domestici in varie famiglie contraddistinte da caratteristiche e operazioni comuni, come illustrato nel paragrafo 6.2. Tale suddivisione ci ha permesso di definire uno schema di interfacce fra loro collegate (ereditarietà), nelle quali sono indicate le operazioni (metodi) generali che i dispositivi della famiglia sono in grado di eseguire (si veda la figura 7.2).

7.2.1.1 La suddivisione in famiglie

La famiglia più generale è quella dei dispositivi nella quale abbiamo individuato le seguenti operazioni:

String nome() : restituisce il nome del dispositivo;

String descrizione() : restituisce la descrizione del dispositivo;

String locazione() : restituisce la stanza in cui il dispositivo è situato.

AddRemoteEventListener : metodo che serve se si vuole dare la possibilità al dispositivo di generare degli eventi remoti; i servizi della comunità Jini che sono interessati a riceverli devono invocare questo metodo.

Le possibili sottoclassi (quindi più specializzate) di un dispositivo sono i sensori, gli attuatori e gli apparecchi.

L'interfaccia Sensore (per esempio i termostati, i sensori di fumo, gas, …) ha, oltre ai metodi dell'interfaccia Dispositivo, l'operazione:

int campionamento() : restituisce il valore della misura ricavata dal sensore (temperatura, intensità luminosa, …).

La famiglia Attuatore (meccanismi per apertura automatica di porte, finestre, tapparelle, …) estende l'interfaccia Dispositivo con i seguenti metodi:

void chiudi() : attiva il meccanismo di chiusura;

void apri() : attiva il meccanismo di apertura;

int stato() : restituisce lo stato (APERTO o CHIUSO).

La famiglia Apparecchio (dispositivo elettrico generale) estende l'interfaccia Dispositivo con i seguenti metodi:

void on() : accende l'apparecchio;

void off() : spegne l'apparecchio;

int stato() : restituisce lo stato elettrico dell'apparecchio(ON o OFF).

Abbiamo poi specializzato l'apparecchio in quattro tipologie, quelle di intrattenimento, di lavaggio, di gas e di comunicazione.

L'apparecchio di tipo Lavaggio (quali lavatrice, lavastoviglie, …) ha in più le seguenti operazioni:

void chiudi_acqua() : chiude il rubinetto dell'acqua;

void apri_acqua() : apre il rubinetto dell'acqua;

int stato_acqua() : restituisce lo stato del rubinetto dell'acqua (ACQUA_APERTA o ACQUA_CHIUSA).

L'apparecchio di tipo Gas (quali forno, caldaia, …) ha le seguenti operazioni:

void chiudi_gas() : chiude il rubinetto del gas;

void apri_gas() : apre il rubinetto del gas;

int stato_gas() : restituisce lo stato del rubinetto del gas (GAS_APERTO o GAS_CHIUSO).

L'apparecchio di tipo Comunicazione (come il telefono, la posta elettronica, il fax, …) ha la seguente operazione:

int invia(String dest, String msg) : invia il messaggio msg a dest, ritorna la costante OK, se l'operazione di invio è andata a buon fine, altrimenti, se si è presentato un errore, restituisce la costante ERRORE.

Lo schema complessivo delle interfacce, con la definizione delle variabili e dei metodi, è mostrato nella figura sotto riportata.

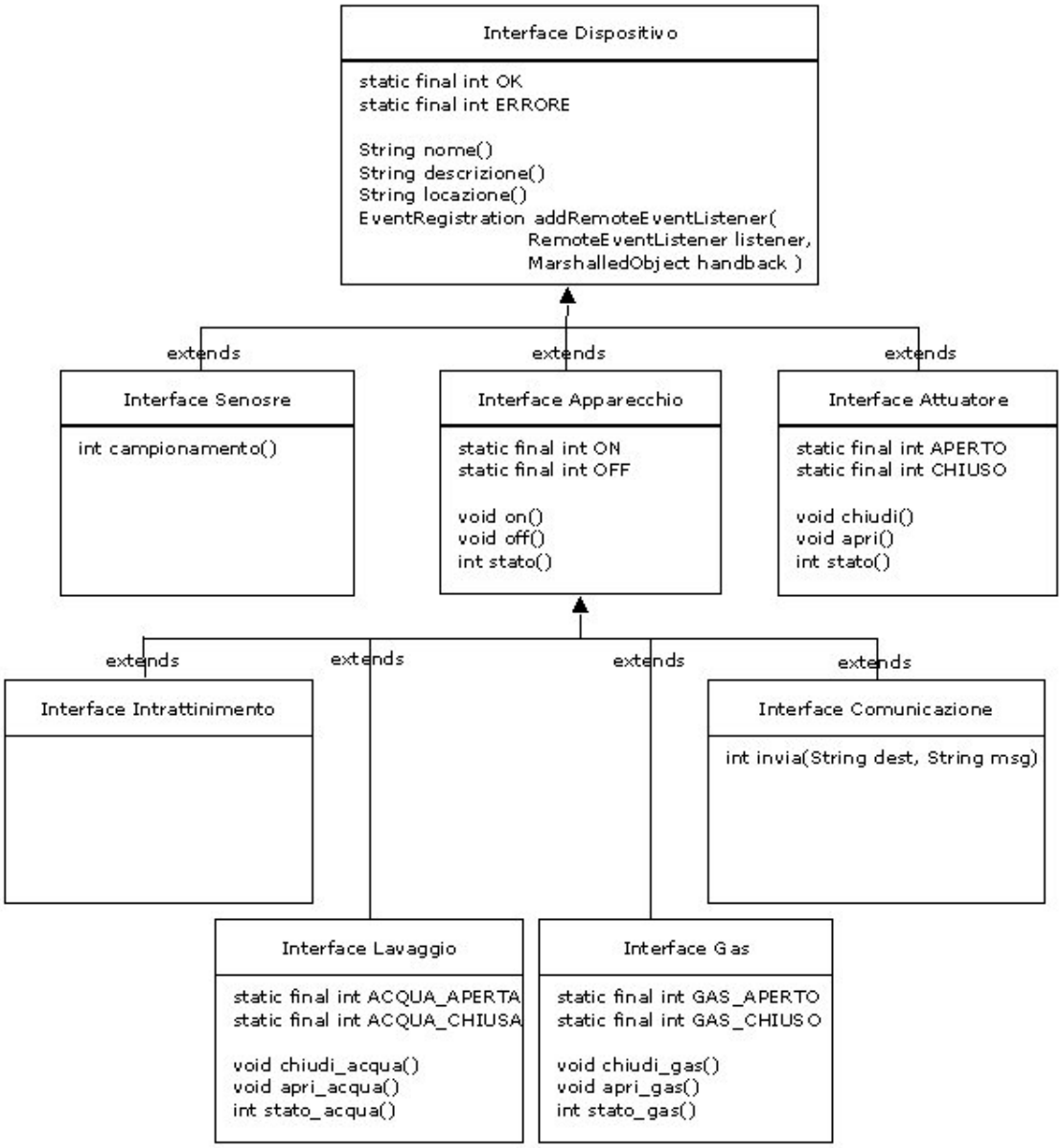

Figura 7.2: Schema delle interfacce

7.2.1.2 Interfaccia utente grafica (GUI)

Abbiamo poi definito, per ogni tipologia di dispositivo, una JApplet standard che realizza l'interfaccia utente grafica (GUI) per monitorarlo e

controllarlo, essa viene utilizzata dal sistema di TeleControllo se negli attributi del servizio non viene trovata un'applet realizzata ad hoc dal produttore del dispositivo. Alcuni esempi di GUI sono riportate nella 7.3.

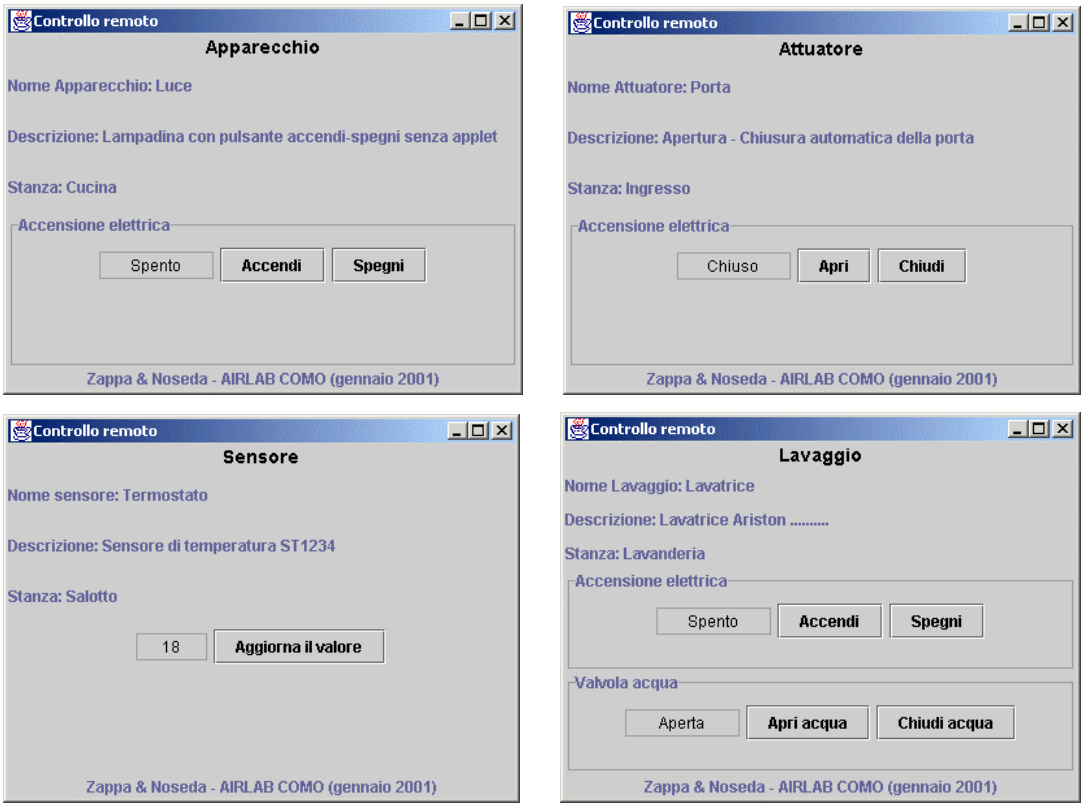

Figura 7.3: Applet standard per il controllo e il monitoraggio dei dispositivo di tipo Apparecchio, Attuatore, Sensore e Lavaggio

L'interfaccia utente e la funzionalità di un elettrodomestico sono, così, state completamente separate; questo approccio è differente da quello tradizionale, nel quale i due aspetti erano legati in maniera molte forte. L'unico punto in comune è l'interfaccia java del dispositivo, dove vengono indicati solamente i metodi e le variabili definite, senza darne un'implementazione. Questa netta separazione permette l'uso del dispositivo sia da parte di un altro servizio, che ne conosce l'interfaccia

(quindi sa quali metodi può invocare e presumibilmente ne conosce l'effetto), sia da parte di una persona che non ha a priori una conoscenza del funzionamento del dispositivo, ma, attraverso la GUI associata, è in grado di controllarlo in maniera semplice e intuitiva.

L'interfaccia utente può essere vista come un adattatore umano del servizio, nel senso che "traduce" l'interfaccia java implementata dal servizio in una rappresentazione capibile dall'utente (si veda la figura 7.4). Noi abbiamo costruito delle interfacce utenti grafiche, ma lo stesso approccio può essere utilizzato per realizzare vari tipi di interfacce, quali, ad esempio, interfacce di tipo testuale, mondi 3D o perfino sistemi che riconoscono comandi vocali; si tratta, quindi, di un aspetto molto importante, soprattutto se lo scopo principale dell'agenzia domotica è quello di dare un aiuto ad un portatore di handicap, infatti, si potrebbero costruire degli adattatori appositi per il disabile, in modo che sfruttino esclusivamente le capacità e le abilità dell'utente.

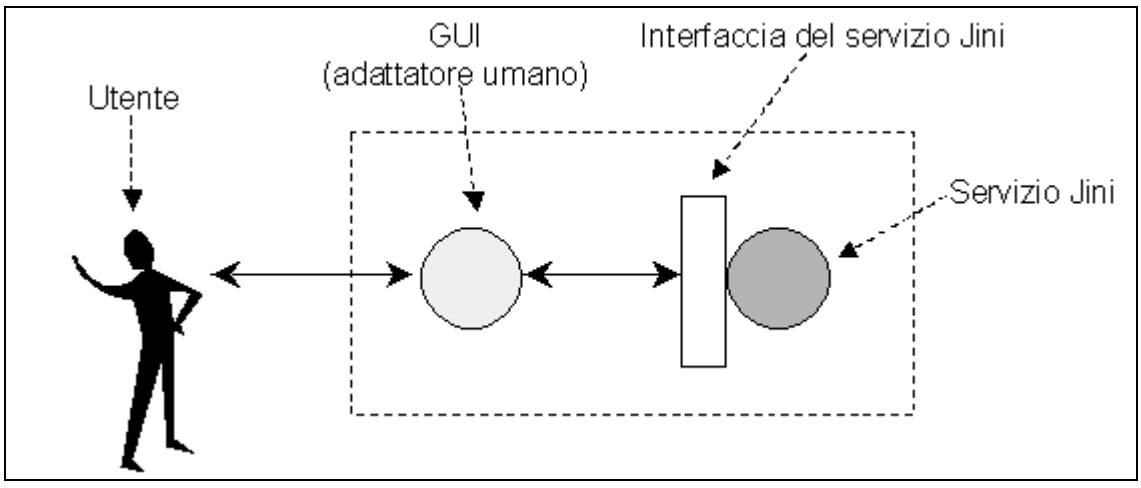

Figura 7.4: Interfaccia utente grafica (GUI)

Inoltre abbiamo definito una classe (EventoCasa.java) che contiene gli identificatori degli eventi che i vari dispositivi sono in grado di generare, e

il metodo descrizioneEvento(long ID), che ritorna una stringa con la descrizione dell'evento associato ad ID. Gli eventi vengono utilizzati dai servizi Jini per portare a conoscenza del supervisore, o di chiunque sia interessato, una situazione anomala, quali sconnessione, temperatura troppo alta o bassa, rilevazione gas, fumo, acqua, …

L'ultima classe generale da noi definita è Icona.java che estende l'interfaccia Jini `net.jini.core.entry.Entry` e quindi può essere utilizzata come uno degli attributi che il servizio usa per iscriversi presso il Lookup Service.

In particolare, con questa classe è possibile permettere a un dispositivo di associare un'icona (piccola immagine che simboleggia il device) in modo da permettere all'utilizzatore umano di avere una percezione visiva di tale apparecchio (come avviene in tutti i sistemi operativi definiti user friendly, dove i programmi e i file sono associati ad una particolare icona).

Tutte queste interfacce e classi sono state compilate e racchiuse nel package `com.zappanoseda`, utilizzato da tutti i servizi dell'agenzia domotica da noi realizzata.

7.2.2 Parte OP (operativa)

Nella parte operativa vengono descritte le funzionalità realizzate dall'elettrodomestico. La prima operazione da fare è quella di individuare qual è la famiglia di elettrodomestici corrispondente al device considerato, tra quelle precedentemente introdotte. Una volta associata al device la sua corrispondete famiglia, si sono anche individuate quali operazioni esso deve implementare.

Per esempio un termostato farà parte della famiglia Sensore e, quindi, dovrà realizzare l'operazione di campionamento, in modo da ritornare la temperatura rilevata, in aggiunta, visto che Sensore eredita da

Dispositivo, bisognerà anche implementare i metodi nome, descrizione e locazione.

In questo modo se un altro servizio deve usare il termostato, sapendo che fa parte della famiglia dei sensori, è a conoscenza dell'esistenza dell'operazione di campionamento e, quindi, è in grado di utilizzarla per ottenere la temperatura misurata e poi, invocando il metodo locazione, può individuare in quale stanza è stata rilevata.

Il dispositivo può anche dichiarare delle operazioni specifiche che non sono indicate nell'interfaccia della famiglia di appartenenza. In questo caso è necessario definire una nuova interfaccia Java che estende la famiglia di partenza, indicando le nuove funzionalità supportate. La parte OP deve quindi implementare la nuova interfaccia realizzata ad hoc per tale device, definendo, quindi, sia le operazioni generali che quelle specifiche. Si pensi all'esempio già citato nel capitolo precedente della lavastoviglie, bisogna costruire una nuova interfaccia che estenda quella di Lavaggio, nella quale vengono aggiunte le funzioni di scelta del tipo di lavaggio, di temperatura dell'acqua, di stato del brillantante e del sale.

Una volta descritta la funzionalità del dispositivo bisogna unirlo all'agenzia domotica (comunità Jini). Per fare questo vengono utilizzati i protocolli Jini di discovery, per cercare un lookup service, e di join, per eseguire la registrazione.

Quando un servizio si registra presso il lookup service (si veda il capitolo 4) può specificare una serie di attributi che vengono utilizzati sia per una migliore descrizione del servizio, sia nella fase di ricerca, quando vengono specificati i valori di tali attributi.

Per questi motivi abbiamo deciso che i dispositivi domestici registrino i seguenti attributi (listato 7.1):

- Name: contiene il nome;
- ServiceInfo: informazioni generiche sul dispositivo (nome, produttore, venditore, versione, modello, numero seriale);

- Location: specifica la collocazione fisica (edificio, piano, stanza);
- Comment: descrizione del dispositivo.

```
static String nome = "Luce";
static String descrizione = "Lampadina con pulsante accendi-
spegni";
static String stanza = "Salotto";

static Entry[] aeAttributes =  { // Descrizione del servizio
     // aeAttributes[0]
   new Name( nome ),                 // Nome (.name)
     // aeAttributes[1]
   new ServiceInfo(nome,             // Nome (.name)
           "Zappa & Noseda", // Produttore (.manufacturer)
           "AIRLAB COMO",    // Venditore     (.vendor)
           "1.0",            // Versione      (.version)
           "on-off",         // Sigla modello (.model)
           "LU001"),         // Serial number (.serialNumber)
     // aeAttributes[2]
   new Location("0",                 // Piano    (.floor)
                stanza,              // Stanza   (.room)
                "Casa1"),            // Edificio (.building)
     // aeAttributes[3]
   new Comment( descrizione ),    // Descrizione (.comment)
   }; // 4 attributi registrati, applet e icona vengono
aggiunte nel main
```

Listato 7.1: Attributi del dispositivo

Inoltre, è possibile allegare altri attributi opzionali, come una applet realizzata dal costruttore del dispositivo per permetterne il controllo da parte dell'utente e un'icona, cioè una piccola immagine che simboleggia il device (listato 7.2).

```
myServer = new Luce();

   // Aggiunta dell'attributo che contiene l'applet Java per
il controllo in remoto

Entry[] attributoApplet = {
  new LuceApplet (myServer)
};

aeAttributes =
LookupAttributes.add(aeAttributes,attributoApplet);

// Aggiunta dell'attributo che contiene l'icona
Icona ic = new Icona ("luce.gif");
```

```
if ( ic.icon.getIconHeight() < 0 ) // icona non trovata
  System.out.println("Luce: ICONA NON TROVATA!");
else {
  Entry[] attributoIcona = {
  ic
  };
  aeAttributes =
LookupAttributes.add(aeAttributes,attributoIcona);
}

joinmanager = new JoinManager(
      myServer,
      aeAttributes,
      myServer,
      new LeaseRenewalManager () );
```

Listato 7.2: Attributi opzionali e registrazione del dispositivo

Per ultimo, se si vuole dare la possibilità al servizio di generare degli eventi, che sono poi usati dal modulo di gestione degli allarmi per indicare delle situazioni anomale, è necessario introdurre l'operazione addRemoteEventListener (listato 7.3), che viene invocata dai servizi che vogliono ricevere gli eventi prodotti da questo dispositivo.

```
// Aggiunta di un servizio alla lista dei servizi a cui
consegnare gli eventi generati

public EventRegistration addRemoteEventListener(
          RemoteEventListener listener,
          MarshalledObject handback) throws RemoteException {

 try {
   listeners.put(listener,handback);
   System.out.println ("RemoteEventListener aggiunto.");
   return new EventRegistration(
               EventoCasa.SCONNESSIONE,
               this,
               null,
               countEvent);
 }
 catch (Exception e) {
   e.printStackTrace();
   return null;
 }
}
```

Listato 7.3: Sottoscrizione di un servizio per ricezione eventi generati

Si veda l'appendice C per il listato completo della implementazione di un dispositivo.

7.3 Sistema di telecontrollo e monitoraggio

Questo sistema permette all'utente di gestire e controllare tutti i dispositivi presenti nella casa. Abbiamo realizzato tre tipi di sistemi di telecontrollo e monitoraggio: il primo è un programma Java apposito (versione programma); il secondo permette di collegarsi ovunque siamo attraverso la rete Internet con un browser (versione web); con il terzo è possibile controllare e monitorare tutti gli elettrodomestici presenti nella casa attraverso la posta elettronica (versione e-mail). Nella figura 7.5 è riportata la visione complessiva dell'agenzia domotica da noi realizzata. Nei paragrafi seguenti sono illustrati nel dettaglio queste le tre differenti versioni del sistema di monitoraggio e telecontrollo.

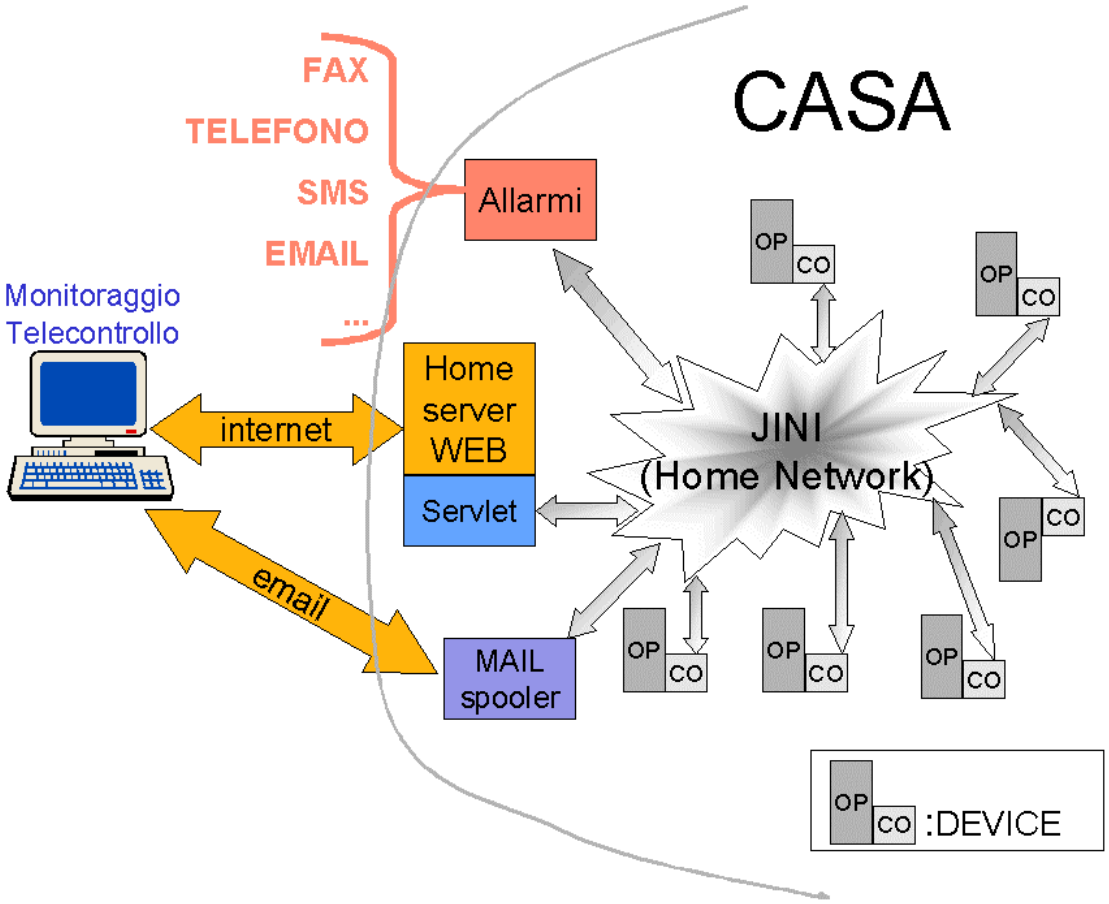

Figura 7.5: Visione complessiva dell'agenzia domotica

7.3.1 Versione programma

Si tratta di un programma java (TeleControllo.java) che per funzionare deve avere a disposizione una JVM e Jini, quindi è stato, da noi, pensato per essere utilizzato o sul PC di casa o su quello in ufficio.

Quando il programma viene lanciato, per prima cosa individua il lookup service dell'agenzia domotica all'indirizzo specificato sulla linea di comandi, gli indirizzi sono del tipo `jini://indirizzoIP:porta`, ad esempio `jini://131.175.143.99:8080`, se nessun argomento è stato

passato il lookup service viene cercato sulla macchina locale (localhost) come mostrato nel listato 7.4.

```
/* Ricerca del Jini Lookup Service (reggie), sul localhost o,
   se specificato come argomento, ad su un certo IP
   ================================================  */

  try {
    if ( args.length > 0 ) {
      System.out.println ("ARGOMENTO : " + args[0]);
      lookup = new LookupLocator ( args[0] );
    }
    else {
      lookup = new LookupLocator ( "jini://localhost" );
    }
  }
  catch (Exception e) {
    System.out.println ("Lookup non attivo, exception: "+ e);
  }
```

Listato 7.4: Ricerca del lookup service

Una volta individuato, il lookup service viene interrogato in modo da ottenere la lista di tutti i dispositivi presenti attualmente nella casa, tale ricerca viene effettuata cercando i servizi che implementano l'interfaccia Dispositivo, che, come abbiamo visto, è il punto di partenza di tutti i device domestici (listato 7.5).

```
/* Ricerca dei dispositivo iscritti presso il Jini Lookup
   Service (reggie)
   ================================================  */

   Class[] c = { Dispositivo.class };
   template = new ServiceTemplate (null, c, null);
   matches = registrar.lookup(template, 50);
```

Listato 7.5: Ricerca dei dispositivi domestici

A questo punto si ha l'oggetto matches di tipo `ServiceMatches` che contiene tutti i dispositivi ricercati; in particolare sono presenti:

- il campo `totalMatches`, che indica quanti servizi sono presenti;
- il campo `items`, che è un array di `ServiceItem` che contiene tutti i servizi.

In ogni ServiceItem sono incapsulati gli attributi registrati (attributeSets) e l'oggetto che rappresenta il servizio vero e proprio (service) usato per accedere alle funzionalità del dispositivo domestico.

Il programma di TeleControllo scandisce la lista dei dispositivi trovata e per ognuno di essi ricerca negli attributi: se è stata registrata un'icona, in caso affermativo viene visualizzata, ed, inoltre, se è presente un'applet costruita dal costruttore per controllare l'elettrodomestico. Se tale applet è presente, quando l'utente esprime la volontà di collegarsi al dispositivo, tale applet viene caricata in una nuova finestra, se invece l'applet non è stata registrata, viene utilizzata quella generica a secondo della famiglia implementata dal dispositivo (listato 7.6).

```
for (i = 0; i < lista.totalMatches; i++) {
    if (lista.items[i].service instanceof Dispositivo) {
      disp= (Dispositivo) lista.items[i].service;
      testo=disp.nome() + " " + disp.locazione() ;
      bottone = new JButton( testo );
      bottone.setToolTipText( disp.descrizione() );//Helpino

      al = new Azione();
      al.n = testo;

      numDisp++;

/*    Cilco sugli attributi dei servizi trovati.
      ================================================  */
      for (j = 0; j < lista.items[i].attributeSets.length;
j++) {
         object = lista.items[i].attributeSets[j];
/*   Se la classe di un attibuto è istanza di una Applet
     associa al bottone l'apertura di tale Applet.
     Se invece si tratta di una icona la visualizza.
     ================================================  */

     if (object instanceof Applet) {
       al.applet = (Applet) object;
     }
     else if (object instanceof Icona) {
       Icona icona = (Icona) object;
       bottone.setIcon( icona.getIcon() );
       al.iconcina = icona.getImage() ;
     }
   }

/* Se negli attributi del servizio non è stata trovata
   nessuna Applet viene carica quella di default
   nell'interfaccia implementata dal servizio
   ================================================  */
```

```
   if ( al.applet == null ) {    // nessuna applet trovata
     if (lista.items[i].service instanceof Sensore)
       al.applet = new SensoreApplet( (Sensore) disp );
     else if (lista.items[i].service instanceof Apparecchio){
       if (lista.items[i].service instanceof Gas)
         al.applet = new GasApplet( (Gas) disp );
       else if (lista.items[i].service instanceof Lavaggio)
         al.applet = new LavaggioApplet( (Lavaggio) disp );
       else if (lista.items[i].service instanceof
Intrattenimento)
         al.applet = new IntrattenimentoApplet(
(Intrattenimento) disp);
       else if (lista.items[i].service instanceof
Comunicazione)
         al.applet=new ComunicazioneApplet((Comunicazione)
disp );
       else
         al.applet=new ApparecchioApplet((Apparecchio)disp);
     }
     else if (lista.items[i].service instanceof Attuatore)
       al.applet = new AttuatoreApplet ( (Attuatore) disp );
     else
       al.applet = new DispositivoApplet((Dispositivo)disp );
   }
  } //for
}
```

Listato 7.6: Lista dei dispositivi domestici

7.3.2 Versione web

Questa versione è pensata per l'utente che vuole monitorare e telecontrollare la propria casa dal proprio PC senza voler installare una JVM e Jini, ma anche per chi si muove con frequenza e vuole utilizzare un'interfaccia grafica gradevole, senza utilizzare l'e-mail. In qualunque momento e in qualunque punto del globo, il padrone di casa si può connettere alla propria casa tramite un comune browser e avere sotto controllo e poter comandare l'intera serie di dispositivi.

Per alleggerire al massimo il lato client e per avere un'integrazione migliore con il sistema Jini, la soluzione adottata prevede l'utilizzo delle servlet, che sono installate in un elaboratore della casa, il quale monta anche un server-web che le supporta, gestisce le richieste che arrivano

via Internet e manda le risposte. Nel nostro caso il web-server utilizzato è il Tomcat 4.0, il quale è utilizzato anche come server per le richieste-risposte sulla nostra rete domestica.

Quando ci si collega alla casa via web, per prima cosa, la servlet si collega alla comunità Jini per ricercare i servizi disponibili in questo momento (listato 7.7).

```
private void connessioneJini () {

/*  Connessione alla comunità jini per ricercare i servizi
disponibili
=========================================================  */
  int iPort;
  String sHost;
  Entry[] aeAttributes;
  LookupLocator lookup;
  ServiceID id;

  try {
/* Impostazione del security manager per permettere a RMI
   class loader di raggiungere il codebase per le classi che
   non sono disponibili localmente
   ================================================  */
    System.setSecurityManager (new RMISecurityManager ());


/*  Ricerca del Jini Lookup Service (reggie), sul localhost
    ================================================  */
    lookup = new LookupLocator ("jini://localhost");
    sHost = lookup.getHost ();
    iPort = lookup.getPort ();


/*  Ricerca del Jini Lookup Service (reggie), sul localhost
    o, se specificato come argomento, ad su un certo IP
   ================================================  */
    registrar = lookup.getRegistrar ();
    id = registrar.getServiceID ();
  }
  catch (Exception e) {
    out.println ("<font size='1'>exception: " + e +
"</font><br>");
    out.println ("<font color='#FF0000'><b>Impossibile
trovare il lookup service<br></b></font><br>");
  }
}
```

Listato 7.7: Connessione alla comunità Jini

L'utente ottiene una pagina HTML generata dinamicamente dalla servlet, nella quale vengono elencati tutti i dispositivi attualmente presenti nella federazione Jini ordinati in una tabella dove per ognuno, si trova: nome, eventuale icona, descrizione, locazione nella casa, tipo di dispositivo.

A questo punto si prospettano varie possibilità per l'utente. Si possono effettuare ricerche avanzate di dispositivi: per nome, produttore, venditore, versione, sigla modello, numero seriale, piano di locazione, posizionati in una determinata stanza e per famiglia (dispositivo, sensore, attuatore, apparecchio, intrattenimento, lavaggio, gas, comunicazione).

Ci si può collegare ad un dispositivo, vedere il suo stato e compiere azioni su di esso, ad esempio accenderlo, spegnerlo, campionarlo (in caso di sensore), ecc. Per ogni classe di dispositivo (sensore, attuatore, apparecchio, intrattenimento, lavaggio, gas, comunicazione), ovviamente l'interfaccia grafica è diversa e permette di effettuare le operazioni (chiamate ai metodi) proprie della classe.

Il problema che si riscontra nel costruire una interfaccia web è che il protocollo http è privo del concetto di stato, infatti funziona, in maniera approssimativa, in una serie di richieste di pagine HTML da parte del browser al quali il web-server risponde. Questo inconveniente viene superato dalle servlet, come già detto nel paragrafo 4.4, che sono in grado di "ricordare" il proprio stato, grazie al fatto che una servlet non viene distrutta dopo aver inviato la risposta, ma è mantenuta in esecuzione dal web-server, in questo modo gli oggetti globali che sono stati inizializzati ed assegnati nella richiesta precedente continueranno ad esistere in quella successiva. In questo modo è possibile, per esempio, far vedere all'utente, gli accessi complessivi alla casa, da quando è stato attivato il web-server, quindi nel nostro caso, da quando è stata avviata l'agenzia domotica; basta definire una variabile globale di tipo int,

inizializzata ad 0, e successivamente incrementata di 1 ad ogni accesso, e inviarla nella pagine HTML creata per la risposta.

Con lo stesso meccanismo abbiamo potuto gestire la nostra agenzia domotica, i passi fondamentali sono i seguenti:

- al primo accesso la servlet si collega alla comunità Jini, ricerca i dispositivi domestici presenti, memorizzandoli in un oggetto globale, e risponde con una pagina HTML con la tabella nella quale sono elencati tutti i dispositivi;
- quando l'utente si collega ad un certo dispositivo il web-server richiede nuovamente la risposta alla servlet che riconosce a quale dispositivo l'utente si vuole collegare (legge ID attraverso il metodo GET) e risponde di conseguenza.

Il web-server, da noi utilizzato, concede alle servlet pochi permessi per questioni di sicurezza, in modo da tutelarsi da eventuali azioni potenzialmente pericolose. Il primo problema, da noi affrontato, è stato quello di impostare la configurazione di sicurezza del web-server, in modo da concedere alla nostra servlet la possibilità di utilizzare l'infrastruttura Jini. Per fare questo bisogna editare il file delle policy del web-server (se si utilizza Tomcat il file è `tomcat.policy` presente nella cartella conf).

7.3.3 Versione e-mail

Con questa versione (MonitoraggioEmail.java) del programma di telecontrollo e monitoraggio degli elettrodomestici della casa, si vuole dare la possibilità all'utente di dare dei comandi inviando una e-mail all'indirizzo di posta elettronica della casa e ricevendo, di conseguenza, la risposta nella propria casella.

Per utilizzare questa tipologia di controllo, è necessario avere a disposizione solamente un accesso ad Internet e la possibilità di inviare

messaggi di posta elettronica; quindi è pensata per poter telecontrollare la casa in qualsiasi posto, da qualsiasi PC e perfino con un PDA o con un telefono cellulare con il quale sia possibile spedire e-mail.

Le impostazioni della casella di posta elettronica sono fornite al programma attraverso la linea di comandi e hanno la seguente sintassi `pop3://login:password@server_pop server_smtp`. Per esempio, per la nostra applicazione abbiamo utilizzato, per la casa, l'indirizzo di posta elettronica `controllo.casa@katamail.com` e quindi la stringa da passare al programma è del tipo: "pop3://controllo.casa: ***@mail.katamail.com komodo.ing.unico.it".

Il programma, una volta avviato, prepara la connessione alla casella di posta elettronica e poi ciclicamente la legge per vedere se sono arrivate nuove e-mail e le analizza per individuare ed eseguire i comandi presenti (listato 7.8). Per la lettura e l'invio di messaggi di posta elettronica viene utilizzato JavaMail 1.2, si tratta di un insieme di API prodotto da Sun Microsystems che permettono di utilizzare le e-mail nei programmi Java.

```
private void letturaEmail() throws MessagingException,
IOException {

  Store store = session.getStore(urlName);
  store.connect();
  Folder inbox = store.getFolder("INBOX");
  inbox.open(Folder.READ_WRITE);
  int count = inbox.getMessageCount();
  System.out.println("Ci sono " + count + " messaggi");
  for(int i = 1;i <= count;i++) {
    Message m = inbox.getMessage(i);
    javax.mail.Address[] ad = m.getFrom();
    if ( ad.length >0 )
      System.out.println(count + ") DA: " + ad[0] );
      System.out.println("   OGGETTO: " + m.getSubject() );
      if ( letturaComandi(m) == true )
       // Cancellazione del messaggio
       m.setFlag(Flags.Flag.DELETED,true);
    }
  inbox.close(true);
  store.close();
}
```

Listato 7.8: Procedura di lettura delle e-mail ricevute

Quando viene ricevuta una nuova e-mail, viene chiamata una funzione che ha il compito di individuare le operazioni richieste. Abbiamo deciso di inserire i comandi nell'oggetto del messaggio di posta elettronica con una sintassi molto semplice:

- LISTA: risponde con un messaggio nel quale sono indicati tutti i dispositivi domestici attivi, con il loro nome, la descrizione, la locazione e le operazioni possibili su tale device, inoltre viene associato ad ogni device un identificatore numerico (ID) in modo che successivamente possa essere usato per controllarlo;
- CONTROLLO ID=<num> OP=<num>: viene invocata l'operazione indicata sul dispositivo a cui era stata associato l'ID specificato.

7.4 Sistema di gestione degli allarmi e degli scenari

Nella parte di gestione della casa (allarmi e scenari) si definisce l'agente supervisore e quello ascoltatore, la comunicazione mediante eventi e la base di conoscenza.

Abbiamo quindi una classe Java che si occupa di ricevere gli eventi dagli altri agenti (GestoreAllarmi.java), trasformarli in fatti che poi vengono inseriti nella base di conoscenza (Motore.clp). Il sistema esperto realizzato in Jess, ogni volta che viene fatto partire, inferisce le azioni da far compiere agli agenti a seconda del fatti presenti, i quali sono una rappresentazione dello stato della casa e vengono inseriti in seguito alla rilevazione degli eventi.

7.4.1 La gestione degli allarmi

La classe java, riceve gli eventi generati da un qualsiasi agente e asserisce i fatti da inserire nella base di conoscenza, poi fa partire il motore inferenziale.

Per prima cosa, bisogna definire un oggetto di tipo Rete, che è il tipo del motore inferenziale. Poi bisogna importare la base di conoscenza con i fatti iniziali e le regole (listato 7.9).

```
/* Importa e definisce il motore inferenziale */
r = new Rete();

r.executeCommand("(batch jess/scriptlib.clp)");
r.executeCommand("(batch Motore.clp)");
```

Listato 7.9: Caricamento del motore inferenziale

Ogni volta che viene rilevato un evento, il sistema controlla di che tipo è e agisce di conseguenza, asserendo i fatti necessari; nel listato 7.10 e riportato un esempio riferito all'allarme gas.

```
if ( IDevento == EventoCasa.RILEVAZIONE_GAS ) {
  Fact f = new Fact ("allarme", r);
  f.setSlotValue("tipo", new Value("gas", RU.STRING));
  f.setSlotValue("stanza", new Value (disp.locazione(),
RU.STRING));

  r.assert(f);
  r.run();
}
```

Listato 7.10: Rilevazione di una fuga di gas

Quando un inferenza si riferisce ad un'azione che un agente deve compiere, bisogna invocare il metodo relativo, ad esempio la chiusura della valvola generale del gas. Abbiamo, quindi, realizzato una serie di procedure (listato 7.11) che vengono richiamate direttamente dalle regole

definite nel sistema esperto e che hanno il compito di collegarsi ai vari dispositivi presenti in quel momento nell'agenzia, invocando i metodi necessari per attuare l'azione richiesta.

```
Class[] cl = { Gas.class };
if ( s.compareTo("0") == 0 )
  s = null;

Entry[] attributi = { new Location(null, s, null) };
ServiceTemplate template = new ServiceTemplate (null, cl,
attributi);
try{
  matches = registrar.lookup(template,50);
  for (int i = 0; i < matches.totalMatches; i++) {
    Gas appGas= (Gas) matches.items[i].service;
    appGas.chiudi_gas();
  }
}
catch (Exception e) {
  System.out.println ("Exception: " + e );
}
```

Listato 7.11: Chiusura valvole gas

7.4.2 Motore inferenziale

Per la definizione dei fatti, abbiamo utilizzato i tipi di fatto non ordinato, ovvero per i quali è stata definita una struttura (template), che indica i campi (slot) presenti (listato 7.12).

```
(deftemplate allarme
     (slot tipo)
     (slot stanza)
)

(deftemplate allarme_temp
     (slot a_b)
     (slot stanza)
     (slot temperat)
)

(deftemplate dominus
     (slot locazione)
)

(deftemplate stato_dispositivo
```

```
        (slot dispositivo)
        (slot stanza)
        (slot stato)
)

(deftemplate liv_allarme
        (slot tipo)
        (slot livello)
        (slot stanza)
)
```

Listato 7.12: Definizione della struttura dei fatti

Un fatto di tipo allarme è inserito nella base di conoscenza dall'agente supervisore in seguito al verificarsi di un evento generato da un sensore o da un elettrodomestico che ha individuato una situazione da gestire (ad esempio allarme fuga di gas). Nell'allarme è, quindi, indicato il tipo di situazione ed inoltre la stanza della casa nella quale è stata rilevata.

Allarmi particolari sono quelli di temperatura troppo alta o troppa bassa per i quali viene definito un tipo di fatto a parte, in quanto bisogna anche registrare il valore della temperatura campionata dal sensore.

Il fatto di tipo dominus è utilizzato per sapere se il padrone di casa è attualmente presente o meno.

Per la gestione degli scenari è necessario conoscere lo stato di particolari dispositivi, quindi è stata definita la struttura stato_dispositivo nel quale viene indicato il tipo, lo stato del dispositivo e la sua locazione.

La struttura liv_allarme serve per definire il livello di un particolare allarme, infatti , come visto nel paragrafo 6.3.2, abbiamo deciso di dare per ogni allarme quattro livelli (rilevazioni temporali successive), a ognuno dei quali si associano azioni diverse. Con questa tipologia di allarmi abbiamo gestito le fughe di gas, le perdite di acqua e le rilevazioni di fumo. Ad ogni livello, e per ogni tipologia esiste un'opportuna regola. La regola di livello 1 registra solamente l'allarme, quella di livello 2 prende le prime contromisure, quella di livello 3 si occupa di avvisare il padrone di casa e di attuare altre contromisure, con quella di livello 4 si richiede

l'aiuto di personale esterno specializzato nella tipologia dell'allarme rilevato.

```
; Gas
(defrule allarme_gas_1
     ?g <- (allarme (tipo "gas") (stanza ?stanza))
     (not (liv_allarme (tipo gas) (livello ?) (stanza
?stanza)))
     =>
     (retract ?g)
     (assert (liv_allarme (tipo gas) (livello 1) (stanza
?stanza)))
)

(defrule allarme_gas_2
     ?g <- (allarme (tipo "gas") (stanza ?stanza))
     ?g1 <- (liv_allarme (tipo gas) (livello 1) (stanza
?stanza))
     =>
     (retract ?g ?g1)
     (assert (liv_allarme (tipo gas) (livello 2) (stanza
?stanza)))
     (call ?cl chiudi "gas" ?stanza)
     (call ?cl chiudi "corrente" ?stanza)
)

(defrule allarme_gas_3
     ?g <- (allarme (tipo "gas") (stanza ?stanza))
     ?g2 <- (liv_allarme (tipo gas) (livello 2) (stanza
?stanza))
     =>
     (retract ?g ?g2)
     (assert (liv_allarme (tipo gas) (livello 3) (stanza
?stanza)))
     (call ?cl chiudi "gas" "0")
     (assert (avvisa_dominus "gas"))
)

(defrule allarme_gas_4
     ?g <- (allarme (tipo "gas") (stanza ?stanza))
     ?g3 <- (liv_allarme (tipo gas) (livello 3) (stanza
?stanza))
     =>
     (retract ?g ?g3)
     (assert (liv_allarme (tipo gas) (livello 4) (stanza
?stanza)))
     (assert (avvisa_pompieri "gas"))
     (assert (avvisa_sicurezza "gas"))
     (call ?cl apri_porte ?stanza)
)
```

Listato 7.12: Allarme gas a più livelli

Le altre tipologie di allarmi da noi presi in considerazione sono: la rilevazione di un intruso, gli allarmi di temperatura troppo alta o troppo bassa; per i quali si ha un unico livello di allarme, nel quale vengono prese le contromisure necessarie.

Il rilevamento di un intruso nella casa viene attivato dall'individuazione di un movimento in una stanza e dell'assenza del padrone di casa. Questi fatti portano ad eseguire delle azioni in modo di tentare di isolare la persona nella stanza e nell'avvisare la sicurezza e il padrone, quindi si provvede a richiedere la chiusura delle porte e il successivo spegnimento di tutti i dispositivi presenti nella stanza (listato 7.13).

```
; Intruso
(defrule allarme_intruso
     ?i <- (allarme (tipo "movimento") (stanza ?stanza))
     (not (dominus (locazione "casa")))
     =>
     (retract ?i)
     (call ?cl chiudi_porte ?stanza)
     (call ?cl chiudi "corrente" ?stanza)
     (assert (avvisa_sicurezza "intruso"))
     (assert (avvisa_dominus "intruso"))
)
```

Listato 7.13: Allarme intruso

Un ulteriore impiego del sistema esperto interviene nella gestione degli scenari. Noi abbiamo individuato come scenari possibili l'entrata, l'uscita di casa del padrone e l'ingresso in una stanza buia.

Per esempio, per quanto riguarda lo scenario ingresso in una stanza buia, abbiamo definito due regole, la prima viene attivata quando si rileva un movimento in una stanza, il padrone è in casa, l'intensità della luce è bassa e le tapparelle sono chiuse, in questo caso viene richiesta all'agente supervisore l'azione di apertura delle tapparelle della stanza considerata. La seconda regola scatta quando si rileva un movimento in una stanza, il padrone è in casa, l'intensità della luce è bassa, ma le tapparelle sono chiuse, in questo caso si richiede l'accensione della luce della stanza (listato 7.14).

```
(defrule poca_luce1
     ?m <- (allarme (tipo "movimento") (stanza ?stanza))
     ?d <- (dominus (locazione "casa"))
     ?l <- (allarme (tipo "luce_bassa") (stanza ?stanza))
     ?sd <- (stato_dispositivo (dispositivo "tapparella")
(stanza ?stanza) (stato "chiusa"))
     =>
     (retract ?m ?l)
     (call ?cl apri_tapparelle ?stanza)
)

(defrule poca_luce2
     ?m <- (allarme (tipo "movimento") (stanza ?stanza))
     ?d <- (dominus (locazione "casa"))
     ?l <- (allarme (tipo "luce_bassa") (stanza ?stanza))
     ?sd <- (stato_dispositivo (dispositivo "tapparella")
(stanza ?stanza) (stato "aperta"))
     =>
     (retract ?m ?l)
     (call ?cl accendi_luci ?stanza)
)
```

Listato 7.14: Scenario entrata in stanza buia

Come si può vedere, le azioni da attuare vengono invocate direttamente all'interno del motore Jess verso la classe Java GestoreAllarmi, con il comando `call`, nel quale si indica la classe Java cui fare riferimento, il nome del metodo da richiamare, seguiti dagli eventuali parametri richiesti dal metodo stesso.

7.5 Sistema di programmazione delle azioni

Il sistema di programmazione delle azioni (ToDo.java) permette di eseguire delle operazioni sugli elettrodomestici della casa in un prefissato istante temporale.

Questo programma gestisce una lista ordinata rispetto al tempo delle azioni da attivare (AzioneDifferita). AzioneDifferita (listato 7.15) è composta da:

- data: l'istante temporale nel quale eseguire l'operazione;
- dispositivo: il dispositivo domestico interessato;
- operazione: l'operazione da svolgere.

Questa classe è stata da noi implementata a partire dall'interfaccia Java `java.lang.Comparable`, in modo da avere un totale ordinamento fra le azioni, in particolare l'ordinamento è basato sul campo data.

```
class AzioneDifferita implements Comparable {

  Date data;
  Dispositivo dispositivo;
  int operazione;

  public AzioneDifferita (Date d,
                          Dispositivo disp,
                          int op ) {
    data = d;
    dispositivo = disp;
    operazione = op;
  }

  public int compareTo(Object o) {
    AzioneDifferita a = (AzioneDifferita) o;
    int comp =  data.compareTo(a.data);
    if ( comp == 0 ) { comp = -1; }
    return comp;
  }
}
```

Listato 7.15: La classe AzioneDifferita

Quando il programma viene eseguito, si collega all'agenzia domotica e ricerca l'elenco completo dei dispositivi domestici attivi, dando la possibilità all'utente di inserire una nuova azione su uno di essi; in particolare bisogna indicare su quale dispositivo si vuole eseguire la particolare operazione nella specificata data.

Il programma attiva ciclicamente la funzione controllaAzioni(), che scandisce la lista delle azioni e ricerca quelle che necessitano di essere eseguite andando a leggere l'ora di sistema (listato 7.16); è possibile

impostare il periodo di esecuzione del ciclo assegnando alla variabile globale periodo, l'intervallo in secondi desiderato.

```
private static void controllaAzioni () {

  Date dataAttuale = new Date();
  Iterator it = tabella.iterator();
  AzioneDifferita a;
  while ( it.hasNext() ) {
    a = (AzioneDifferita) it.next();
    if ( dataAttuale.compareTo(a.data) >= 0 ) {
      // Azione da attivare
      it.remove();
      try {
        if ( a.dispositivo instanceof Lavaggio ) {
          Lavaggio lav = (Lavaggio) a.dispositivo;
          switch ( a.operazione ) {
            case 1: // Accendere
              lav.on();
              break;
            case 2: // Spegnere
              lav.off();
              break;
            case 3: // Aprire acqua
              lav.apri_acqua();
              break;
            case 4: // Chiudere acqua
              lav.chiudi_acqua();
              break;
          }
        }
        else if ( a.dispositivo instanceof Gas ) {
          Gas gas = (Gas) a.dispositivo;
          switch ( a.operazione ) {
            case 1: // Accendere
              gas.on();
              break;
            case 2: // Spegnere
              gas.off();
              break;
            case 5: // Aprire gas
              gas.apri_gas();
              break;
            case 6: // Chiudere gas
              gas.chiudi_gas();
              break;
          }
        }
        else if ( a.dispositivo instanceof Apparecchio ) {
          Apparecchio app = (Apparecchio) a.dispositivo;
          switch ( a.operazione ) {
            case 1: // Accendere
              app.on();
              break;
            case 2: // Spegnere
              app.off();
```

```
                break;
              }
            }
            else if ( a.dispositivo instanceof Attuatore ) {
              Attuatore att = (Attuatore) a.dispositivo;
              switch ( a.operazione ) {
                case 7: // Aprire
                  att.apri();
                  break;
                case 8: // Chiudere
                  att.chiudi();
                  break;
              }
            }
          }
          catch (Exception e) {
            System.out.println ("ToDo: controllaAzioni()
exception: " + e);
          }
        }
        else { return; }
      }
    }
```

Listato 7.16: Ricerca delle azioni da eseguire controllaAzioni()

7.6 Valutazioni

Il sistema da noi realizzato, nel complesso è abbastanza esteso, come si può notare dalla tabella sottostante, che indica le LOC (Lines Of Code, linee di codice) dei vari moduli. Questi vengono raggruppati a seconda del loro scopo:

- package com.zappanoseda, contiene le interfacce delle famiglie dei dispositivi e le classi di utilità generale;
- la simulazione dei dispositivi;
- la parte di telecontrollo e monitoraggio;
- la parte di gestione “intelligente” con il motore inferenziale.

File	Descrizione	LOC
Package com.zappanoseda		**1294**
Apparecchio.java	Interfaccia famiglia Apparecchio	40
ApparecchioApplet.java	GUI famiglia Apparecchio	170
Attuatore.java	Interfaccia famiglia Apparecchio	40
AttuatoreApplet.java	GUI famiglia Apparecchio	106
Comunicazione.java	Interfaccia famiglia Comunicazione	20
ComunicazioneApplet.java	GUI famiglia Comunicazione	150
Dispositivo.java	Interfaccia famiglia Dispositivo	48
DispositivoApplet.java	GUI famiglia Dispositivo	60
EventoCasa.java	Classe che definisce gli eventi	112
Gas.java	Interfaccia famiglia Gas	42
GasApplet.java	GUI famiglia Gas	154
Icona.java	Classe per gestire le Icone dei dispositivi	40
Intrattenimento.java	Interfaccia famiglia Intrattenimento	16
Lavaggio.java	Interfaccia famiglia Lavaggio	40
LavaggioApplet.java	GUI famiglia Lavaggio	155
Sensore.java	Interfaccia famiglia Sensore	19
SensoreApplet.java	GUI famiglia Sensore	82
Simulazione dispositivi		**4819**
Forno.java	Simulazione di un forno	329
Lavastoviglie.java	Simulazione di una lavastoviglie	318
LavastoviglieApplet.java	GUI lavastoviglie	347
LavastoviglieInterface.java	Interfaccia lavastoviglie	48
Lavatrice.java	Simulazione di una lavatrice	276
Luce.java	Simulazione di una lampadina	258
LuceApplet.java	GUI luce	187
Luce2.java	Simulazione di una lampadina	251
LucePotenziometro.java	Simulazione di una lampadina	249
LucePotenziometroApplet.java	GUI luce con potenziometro	58
Porta.java	Simulazione di una porta	247
PostaElettronica.java	Sistema per invio posta elettronica	358
PostaElettronicaApplet.java	GUI posta elettronica	84
PostaElettronicaInterface.java	Interfaccia posta elettronica	13
SMS.java	Sistema per invio SMS	556
SMSApplet.java	GUI SMS	84
Telefono.java	Simulazione di un telefono	270
Termostato.java	Simulazione di un termostato	344
ToDo.java	Sistema programmazione azioni	642
Telecontrollo e monitoraggio		**2180**
MonitoraggioEmail.java	Sistema di telecontrollo (versione e-mail)	817
ServletCasa.java	Sistema di telecontrollo (versione web)	969
TeleControllo.java	Sistema di telecontrollo (versione programma)	394
Gestione		**826**
GestoreAllarmi.java	Sistema di gestione allarmi e scenari	553
Motore.clp	Motore inferenziale in Jess	273
TOTALE		**9219**

Tabella 7.1: Linee di codice dell'agenzia domotica

8

CONCLUSIONI E SVILUPPI FUTURI

"The truth is usually just an excuse
for a lack of imagination"
Elim Garak

8.1 Conclusioni

L'intelligenza artificiale applicata alla domotica è un campo di studio recente. Il nostro sforzo è stato quello di integrare i risultati teorici e pratici, raggiunti dalle due discipline separatamente, in modo da ottenere quello che ci eravamo prefissati e cioè un sistema in grado di gestire in modo "intelligente" la casa ed inoltre la possibilità di monitorarla e telecontrollarla.

Con la nostra tesi abbiamo fornito una metodologia che permette di sviluppare una reale applicazione di home automation e di costruire una

smart-home, definendo cosa si intende per gestione “intelligente” della casa e come essa possa essere realizzata.

Il sistema da noi realizzato utilizza tecnologie moderne (Java, Jini, Jess), che stanno sempre più prendendo piede e che potranno, in futuro, giocare un ruolo importante in una serie di nuove applicazioni, che nasceranno grazie alla possibilità di avere a disposizione un sempre più maggior numero di dispositivi in grado di collegarsi tra loro formando una grande rete globale.

L’obiettivo di questa tesi è di realizzare un’agenzia domotica con Jini, tale tecnologia ha richiesto una fase iniziale di studio teorico e apprendimento pratico non irrilevante da parte nostra. All’inizio, Jini ci è sembrato fin troppo complesso per la nostra realizzazione, ma poi, con il progressivo ampliamento del sistema, si è rivelato una scelta azzeccata, che ha permesso di ottenere più facilmente e in modo più elegante, risultati notevoli. L’utilizzo combinato di Java, Jini e Jess, si è tradotto, per l’utilizzatore dell’agenzia domotica, in un sistema flessibile e di facile utilizzo.

8.2 Sviluppi futuri

In primo luogo sarebbe necessario passare dai dispositivi domestici simulati a quelli reali e cioè inserire, all’interno della nostra implementazione, le istruzioni necessarie per compiere il compito richiesto dalla particolare istruzione, quindi, per esempio, campionare una temperatura da un termostato, accendere e spegnere una lampadina, ecc.

Un altro aspetto rilevante, per costruire un’agenzia domotica realmente utilizzabile, è quello di aggiungere, all’interno del sistema delle forti

politiche di sicurezza. Questo aspetto coinvolge soprattutto il sistema di telecontrollo e monitoraggio nelle sue tre versioni (programma, web, e-mail); dovrà essere in grado di riconoscere la persona che tenta di collegarsi alla casa e concedere l'utilizzo del sistema solo al padrone della casa e a chi ne ha il permesso. In pratica bisogna costruire un meccanismo di sicurezza in grado di proteggere la casa da una nuova tipologia di ladri, quelli telematici.

Si potrebbe pensare anche di utilizzare nuovi mezzi di comunicazione per controllare e monitorare la casa, quelli, ad oggi da noi ritenuti particolarmente interessanti, sono il telefono e gli SMS. Con il telefono si potrebbe realizzare un sistema che riconosca i comandi vocali impartiti e che ne effettui le operazioni richieste, mentre con gli SMS bisognerebbe sviluppare un sistema molto simile a quello da noi fatto con i messaggi di posta elettronica.

Un'altra parte del sistema che si può pensare di ampliare è quella di gestione degli allarmi e scenari, bisognerebbe quindi realizzare un motore inferenziale più ampio e complesso.

Un aspetto importante, soprattutto per l'aiuto ai disabili, è quello di costruire nuove interfacce utenti (UI) per i dispositivi domestici. L'agenzia domotica fornisce già un grosso aiuto, perché permette il controllo di tutta la casa dal computer senza la necessità di muoversi da una stanza all'altra, ma bisognerebbe costruire delle UI apposite per il disabile, in modo che possano sfruttare esclusivamente le capacità e le abilità dell'utente. Ad esempio, per persone che hanno difficoltà con l'uso della tastiera si potrebbero sviluppare UI che riconoscano il parlato. Altro tipo di interfaccia può essere la realtà virtuale, con la quale si ricostruisce in tre dimensioni l'ambiente domestico, nel quale sono inseriti i dispositivi e i loro pannelli di controllo; un utente con problemi motori si può muovere nell'ambiente virtuale, avendo l'impressione di muoversi realmente verso un certo oggetto.

Un aspetto di rilevanza teorica che può portare a studi futuri, riguarda il reclutamento run-time degli agenti, in particolare soffermandosi sugli aspetti legati alla cardinalità del sottoinsieme degli agenti reclutati per lo svolgimento di un particolare piano di lavoro, individuando le scelte migliori da attuare a seconda della situazione.

Nuove direzioni di ricerca possono andare verso la connessione con altre agenzie, di vario tipo. Collegarsi con altre agenzie domotiche, situate fisicamente vicine (come un quartiere), permette di estendere le funzionalità e di crearne di nuove (ad esempio, un sistema di gestione degli allarmi globale). Collegarsi con l'agenzia antropica del padrone di casa, permette di telecontrollare le funzioni vitali di una persona ammalata o disabile e di intervenire in caso di necessità. Collegarsi con l'agenzia mobile automobilistica, permette sia un controllo della casa verso il padrone in viaggio (ad esempio avvisandolo di situazioni di traffico o strade interrotte), sia di monitorare e gestire la casa direttamente dall'automobile.

BIBLIOGRAFIA

[1] SMAU. *Tecnologia e applicazioni per la casa: mercato e scenari evolutivi*. Maggio 2000.

[2] Wong Wylie. *Future of home networking rest on FCC*. CNET News.com. 22 maggio 2000.

[3] Merloni. *Ariston Digital: La nuova generazione di elettrodomestici*. 1999.

[4] MIT. *Home automation and intelligent appliances*. 1999.

[5] Mongiovì Paolo. *L'home networking e la soluzione Smart Home*. Convegno SMAU: "Home Automation: nuove prospettive dell'applicazione delle tecnologie digitali per la casa". Milano, 19 ottobre 2000.

[6] Patriarca Enrico. *Gli standard europei verso una definitiva convergenza*. Convegno SMAU: "Home Automation: nuove prospettive dell'applicazione delle tecnologie digitali per la casa". Milano, 19 ottobre 2000.

[7] Berruto Roberto. *Sistemi di comunicazione via web per l'home automation*. Convegno SMAU: "Home Automation: nuove prospettive dell'applicazione delle tecnologie digitali per la casa". Milano, 19 ottobre 2000.

[8] Falcioni Paolo. *La supervisione e la manutenzione remota di componenti di un sistema di home automation*. Convegno SMAU: "Home Automation: nuove prospettive dell'applicazione delle tecnologie digitali per la casa". Milano, 19 ottobre 2000.

[9] Santini Ernesto. *SCS: l'integrazione dei sistemi domotici*. Convegno SMAU: "Home Automation: nuove prospettive dell'applicazione delle tecnologie digitali per la casa". Milano, 19 ottobre 2000.

[10] SMAU. *SMART HOME: La casa intelligente*. 2000.

[11] Di Pasqua Emanuela. *Anche gli elettrodomestici hanno un Q.I.?*. SMAU. 19 ottobre 2000.

[12] Dutta-Roy Amitava. *Networks for homes*. IEEE Spectrum 36(12):26-33. Dicembre 1999.

[13] Roncarolo Barbara. *Quale sarà lo standard per la casa del futuro?*. SMAU. 2000.

[14] Lea Rodger, Gibbs Simon, Dara-Abrams Alec, Eytchison Edward. *Networking home entertainment devices with HAVi*. IEEE Computer 33(9): 35-43. Settembre 2000.

[15] Garber Lee. *Companies Power Up Networking*. IEEE Computer 33(8): 26. Agosto 2000.

[16] Sun Microsystems. *Jini Tecnology glossary*. Gennaio 1999.

[17] Sun Microsystems. *Jini Architecture Specification*. Gennaio 1999.

[18] Sun Microsystems. *Jini Lookup Service Specification*. Gennaio 1999.

[19] Sun Microsystems. *Jini Discovery Utilities Specification*. Gennaio 1999.

[20] Sun Microsystems. *Jini Discovery and Join Specification*. Gennaio 1999.

[21] Sun Microsystems. *Jini Transaction Specification*. Gennaio 1999.

[22] Sun Microsystems. *Jini Distributed Event Specification*. Gennaio 1999.

[23] Sun Microsystems. *Jini Distributed Leasing Specification*. Gennaio 1999.

[24] Sun Microsystems. *Java Remote Method Invocation Specification*. Ottobre 1998.

[25] Puliti Giovanni. *Jini, il genio venuto da Java*. Computer Programming 82: 20-24. Luglio/Agosto 1999.

[26] Venners Bill. *The Jini Vision. JavaWorld*. Agosto 1999.

[27] Venners Bill. *Locate services with the Jini lookup service*. JavaWorld. Febbraio 2000.

[28] Venners Bill. *Separating UI and Functionality*. Artima.com. 2000.

[29] Murphy Keiron. *Jini released from its magic lamp*. Febbraio 1999.

[30] Murphy Keiron. *A Jini pioneer: an interview with Freeman Jackson*. Marzo 1999.

[31] Marchal Benoît. *A first look at Jini lookup and discovery protocols*. Novembre 2000.

[32] Minsky Marvin. *La società della mente*. Adelphi. 1989.

[33] Somalvico Marco*. Aspetti filosofici dell'Intelligenza Artificiale*. Atti del 4° Convegno AIIA. Parma, 1994

[34] Amigoni Francesco, Zanisi Damiano. *Descrizione teorica e proprietà della nozione di agenzia in intelligenza artificiale*. Tesi di Laurea, Politecnico di Milano. A.A. 1994-95.

[35] Amigoni Francesco, Somalvico Marco. *Dynamic agencies: concepts and applications.* Proceedings of the "Sixth Symposium of Italian Association for Artificial Intelligence (AI*IA)". Padova, 1998.

[36] Amigoni Francesco. *Mobile Agent Applications*. Technical Report 98-102, Dipartimento di Elettronica e Informazione, Politecnico di Milano.

[37] Fuggetta Alfonso, Picco Gian Pietro, Vigna Giovanni. *Understanding Code Mobility*. IEEE Transaction on software engineering 24(5): 342-361. Maggio 1998.

[38] Lemay Laurea, Cadenhead Rogers*. Java 1.2 Guida completa*. 1998.

[39] Somalvico Marco. *Natura e ruolo della Domotica*. Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Politecnico di Milano. 2000.

[40] Somalvico Marco. *Intelligenza Artificiale ed Ubiquitous Computing. Metodi e prospettive.* Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Politecnico di Milano. 2000.

[41] Somalvico Marco. *Intelligenza Artificiale.* Progetto di Intelligenza Artificiale e Robotica, Dipartimento di Elettronica e Informazione, Politecnico di Milano. 2000.

[42] Rampa Vittorio. *Telecomunicazioni e Ubiquitous Computing.* Centro di Studio sulle Telecomunicazioni Spaziali del CNR. Dipartimento di Elettronica e Informazione, Politecnico di Milano. 2000.

[43] Friedman-Hill Ernest. *Jess, The Java Expert System Shell*. Distributed Computing Systems, Sandia National Laboratories. Aprile 2000.

[44] Eckel Bruce. *Thinking in Java. Second Edition*. Prentice-Hall. Giugno 2000.

[45] Paulson Linda Dailey. *Exploring the Wireless LANscape.* Computer. Ottobre 2000

[46] Davidson James Duncan, Ahmed Suzanne. *Java Servlet API Specification.* SUN Microsystems, Novembre 1998

[47] Donatella Sciuto. *Le tecnologie Hardware per l'ubiquitous computing.* Dipartimento di Elettronica e Informazione, Politecnico di Milano. 2000.

BIBLIOGRAFIA SU WEB

[web1] HomePlug Powerline Alliance: associazione per la ricerca di comunicazioni ad alta velocità su linea elettrica. http://www.homeplug.org

[web2] Consumer Electronics Association. http://www.ce.org

[web3] HomePNA: organizzazione di industrie per unificare gli standard di comunicazione su linea telefonica. http://www.homepna.org

[web4] ITU: organizzazione internazionale per le telcomunicazioni. http://www.itu.int

[web5] SMAU: ente organizzatore di fiere e convegni sull'information technology. http://www.smau.it

[web6] Merloni: azienda italiana che produce elettrodomestici. http://www.merloni.com

[web7] Bticino: azienda italiana che produce sistemi elettrici. http://www.bticino.it

[web8] Sistema Casa: azienda italiana attiva nella domotica. http://www.inmilano.it/sistemacasa/

[web9] Siemens: multinazionale dell'elettronica. http://www.siemens.com

[web10] Domustech: divisione dell'Olivetti che si occupa di automazione domestica. http://www.olivettilexikon.it

[web11] Electrolux: azienda produttrice di elettrodomestici. http://www.electrolux.it

[web12] HAVi Organization: organizzazione per lo studio di uno standard per i dispositivi di intrattenimento. http://www.havi.org

[web13] Sun Microsystems: azienda che ha sviluppato Java e Jini. http://www.sun.com

[web14] Jini Community: sito ufficiale della comunità Jini. http://www.jini.org

[web15] JavaWorld: rivista elettronica su Java. http://www.javaworld.com

[web16] Gamelan: portale di Information Technology http://www.gamelan.com

[web17] Artima: azienda attiva in progetti Jini. http://www.artima.com

[web18] Siliware: azienda che produce applicazioni in Jini. http://www.jinivision.com

[web19] Noel's Nuggets: esempi e utilizzo di Jini. http://www.enete.com

[web20] Jini Tutorial: guida all'uso di Jini. http://www.javacats.com/US/articles/Ben/Jini1.html

[web21] Jini Technology: lezione su Jini. http://members.tripod.com/gsraj/jinni/chapter

[web22] Dipartimento di Elettronica e Informazione del Politecnico di Milano. http://www.elet.polimi.it

[web23] Java: linguaggio di programmazione multipiattaforma. http://java.sun.com

[web24] Bruce Eckel: Autore del libro “Thinking in Java”, distribuito gratuitamente su Internet. http://www.bruceeckel.com

[web25] Jess: sistema esperto in Java. http://herzberg.ca.sandia.gov/jess

[web26] Servlet Tutorial: guida all’uso delle servlet. http://web2.java.sun.com/docs/books/tutorial/servlets

[web27] aJile Systems: chip che esegue bytecode in hardware. http://www.ajile.com

A

GLOSSARIO

ADSL Asymmetric Digital Subscriber Line. Tecnologia a larga banda per le connessioni residenziali verso Internet.

API Application Programming Interface. Le interfacce che un programmatore utilizza.

AWT Abstract Window Toolkit. Toolkit originale per costruire interfacce grafica in Java.

BRAN Broadband Radio Access Networks.

BT BlueTooth, standard wireless.

Bytecode Il "codice macchina" della JVM. Viene ricavato dalla compilazione del codice sorgente.

Browser Programma che permette la visualizzazione di pagine HTML, utilizzato su Internet per la navigazione.

CCK Complementary Code Keying.

CDC Connected Device Configuration, specifiche Sun per Java API su device limitati per potenza di calcolo e/o memoria, come PDA.

CEBus Standard di CIC per ricercare e usare device su powerline.

CGI Common Gateway Interface.

CIC CEBus Industry Council. Organismo di standardizzazione di CEBus.

Cjip Chip di Imsys, è in grado di eseguire Java bytecode in hardware.

CLDC Connected Limited Device Configuration, specifiche Sun per Java API su device limitati per potenza di calcolo e/o memoria, come i telefoni cellulari.

CLIPS C Language Integrated Production System; sistema di produzione integrato col linguaggio C.

CPU Central Processing Unit; processore.

CSMA Carrier-Sensing Multiple Access. Sistema di accesso multiplo.

CSMA/CD Carrier-Sensing Multiple Access with Collision Detection. Sistema di accesso multiplo con controllo delle collisioni.

CSMA/CDCR Carrier-Sensing Multiple Access with Collision Detection and Collision Recovery. Sistema di accesso multiplo con controllo e risoluzione delle collisioni.

CTP Centralized Token Passing.

CVM Customer Virtual Machine; macchina virtuale Java.

DAI Distributed Artificial Intelligence; intelligenza artificiale distribuita.

DECT Digital European Cordless Telecommunication.

DES Distributed Expert System; sistema esperto distribuito.

DHCP Dynamic Host Configuration Protocol, usato da un dispositivo connesso alla rete per ottenere un indirizzo IP dinamico.

DKS Distributed Knowledge Sources; basa di conoscenza distribuita.

DMT Discrete MultaTone.

DPS Distributed Problem Solving; risoluzione dei problemi distribuita.

DVD Digital Video Disk; dischi per video digitali.

DS Direct Sequence.

DSMA Datagram Sensing Multiple Access.

djinn Il vecchio nome della comunità Jini.

EIA Electronic Industries Alliance.

EJAE Embedded Java Application Environment; ambiente per applicazioni Java embedded.

eJava Le Java API che EJAE supporta.

ETSI European Telecomunications Standards Institute.

FCC U.S. Federal Communication Commission.

FDM Frequency Division Multiplexing.

FDQAM Modulazione di ampiezza in quadratura con differenza di frequenza.

FH Frequency Hopping.

FIA Fixed Intelligent Agent; agente intelligente fisso.

GPS Global Position System; sistema di posizionamento planetario.

GSM Global System for Mobile communications. Sistema di telefonia mobile.

GUI Graphical User Interface; interfaccia utente grafica.

HAVi Home Audio Video interoperability; sistema per interoperabilità audio-video.

HDP Hybrid Distributed Planning; pianificazione distribuita ibrida.

HDTV High Definition TeleVision.

HTML HyperText Markup Language. Linguaggio che si usa per realizzare pagine web.

HTTP HyperText Transfer Protocol. Protocollo di trasferimento di ipertesti.

IDL Interface Definition Language; linguaggio di definizione di interfacce.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IP Internet Protocol.

IPv6 IP versione 6.

IrDA Infra Red Data Access; accesso dati infrarosso.

ITU International Telecomunication Union.

J2EE Java 2 Enterprise Edition, la versione completa del linguaggio Java.

J2ME Java 2 Micro Edition, la versione più limitata del linguaggio Java.

J2SE Java 2 Standard Edition, la versione standard del linguaggio Java.

J9 JVM limitata di IBM per J2ME.

Jar Java Archeives. Formato per impacchettare le classi e le risorse usate da un programma Java. Utilizza lo stesso formato dello Zip.

Java Linguaggio di programmazione indipendente dalla piattaforma sviluppato da Sun.

JCRE JavaCard Runtime Environment.

JCVM JavaCard Virtual Machine.

JDK Java Development Kit.

JESS Java Expert System Shell. Sistema a regole in Java.

Jini Tecnologia Sun per realizzare sistemi distribuiti in Java.

JIT Just In Time compiling, una JVM che compila il Java bytecode in linguaggio macchina specifico della piattaforma su cui l'applicazione Java viene eseguita.

JNI Java Native Interface; interfaccia nativa Java.

JRE Java Runtime Environment.

JSK Jini Starter Kit.

JVM Java Virtual Machine, il programma che interpreta ed esegue il Java bytecode.

kAWT Kilobyte AWT per KVM.

kJava Le Java API supportate da KVM.

KQML Knowledge Query and Manipulation Language; linguaggio di manipolazione e interrogazione della conoscenza.

KVM Kilobyte Virtual Machine.

LAN Local Area Network; rete locale.

LST Layered Space-Time processing.

MAC Medium Access Control; controllo di accesso ad un mezzo.

MFI MIA/FIA Interface. Interfaccia tra agente mobile e agente fisso.

MIA Mobile Intelligent Agent; agente intelligente mobile.

MIME Multipurpose Internet Mail Extensions.

MPEG Moving Picture Experts Group. Sistema di compressione video.

OS Operating System; sistema operativo.

OFDM Orthogonal Frequency Division Multiplexing; multiplazione a divisione di frequenza ortogonale.

PAI Parallel Artificial Intelligence; intelligenza artificiale parallela.

PAN Personal Area Network; rete personale.

PDA Personal Digital Assistant, un piccolo computer, spesso contiene calendario, agenda, blocco note, e altre piccole applicazioni.

PLC Onde convogliate su linea elettrica.

PMA Planning for Multiple Agents; pianificazione per agenti multipli.

POP3 Post Office Protocol 3; protocollo usato per le email su Internet.

QAM Modulazione di ampiezza in quadratura.

RFC Request For Comments.

RMI Remote Method Invocation. La caratteristica di Java per accedere ad oggetti remoti.

RMIC RMI Compiler, il compilatore distribuito con JDK per gli oggetti remoti.

RMID RMI Daemon.

RPC Remote Procedure Call; chiamata di procedura remota.

Salutation Tecnologia per realizzare sistemi distribuiti, simile a Jini.

SMS Short Message System.

SMTP Simple Mail Transfer Protocol; protocollo usato su Internet per inviare i messaggi di posta elettronica.

SWAP Shared Wireless Access Protocol.

Swing Graphics toolkit di Sun. Toolkit per costruire interfacce grafica in Java distribuito a partire dalla versione 1.2 di JDK.

TACS Sistema analogico di telefonia mobile.

TCL Tool Command Language.

TCP Transport Control Protocol. Protocollo di trasporto usato su Internet.

TP5 Twisted Pair category 5; doppino telefonico di categoria 5.

TINI Embedded Java 1.1 circuit (Dallas semiconductor).

UDP User Datagram Protocol. Protocollo di trasporto usato da Internet.

UI User Interface; interfaccia utente.

UPnP Universal Plug and Play. Standard Microsoft per ricercare ed utilizzare device su una rete.

URL Uniform Resource Locator.

VCR Videoregistratore.

VDSL Very High-Speed Digital Subscriber Lines.

WRAP Web Ready Appliance Protocol. Protocollo della Merloni per la comunicazione tra elettrodomestici.

x86 Famiglia di processori di Intel.

Xpresso Chip di Zucotto che è in grado di eseguire codice Java. Supporta Jini e BlueTooth.

B

INSTALLAZIONE ED USO

B.1 Requisiti software

Il sistema da noi realizzato per funzionare necessita dei pacchetti software sotto elencati; si tratta di sistemi scaricabili gratuitamente dall'indirizzo Internet indicato e presenti, assieme alla documentazione e all'intero sistema di agenzia domotica, all'interno del CD-ROM allegato:

- JDK 1.2 o superiore (http://java.sun.com);
- Jini 1.0 o superiore (http://www.jini.org);
- Tomcat o un altro web-server in grado di eseguire le servlet (http://www.apache.org);
- Jess 5.1 o superiore (http://herzberg.ca.sandia.gov/jess);
- JavaMail 1.2 o superiore (http://java.sun.com);
- JavaBeans Activation Framework (http://java.sun.com).

Per una più facile configurazione dell'intero sistema, si consiglia di utilizzare lo schema delle directory mostrato in figura B.1.

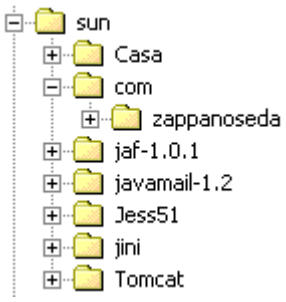

Figura B.1: Struttura delle directory

Elenco delle directory presenti in sun:

- Casa: il cuore dell'agenzia domotica. Contiene i sistemi di telecontrollo versione programma ed e-mail, gestione allarmi e scenari, programmazione azioni e la simulazione di tutti gli elettrodomestici da noi realizzati per testare il sistema.
- com\zappanoseda: è il package che contiene tutti gli schemi delle interfacce delle varie famiglie, oltre a classi di utilità generali come Icona e EventoCasa.
- jaf-1.0.1: JavaBeans Activation Framework, è richiesto da JavaMail.
- javamail-1.2: sistema di gestione e-mail JavaMail.
- Jess5.1: sistema a regole Jess.
- jini: infrastruttura Jini.
- Tomcat: web-server, include, inoltre, le API per realizzare le servlet e la servlet, da noi realizzata, per la versione web del sistema di telecontrollo.

B.2 Configurazione del sistema

Editare i file SetInstallPoint.bat presenti nelle directory sun\Casa, sun\com\zappanoseda, tomcat\... sistemando i cammini esatti per tutte le

variabili presenti (se avete mantenuto la struttura delle directory sopra riportata dovete solamente impostare la variabile RUNTIME_JAR e cambiare il drive delle altre, indicando su quale disco avete installato il sistema). Elenco delle variabili presenti nei file di configurazione:

- *RUNTIME_JAR*: file jar che contiene il runtime java, fornito dal JDK;
- *JINIHOME_BACKSLASH*: percorso in cui è stato installato Jini, utilizzando il formato con backslash (\);
- *JINIHOME_FORWARDSLASH*: percorso in cui è stato installato Jini, utilizzando il formato con forwardslash (/);
- *JESSHOME_BACKSLASH*: percorso in cui è stato installato Jess, utilizzando il formato con backslash (\);
- *JESSHOME_FORWARDSLASH*: percorso in cui è stato installato Jess, utilizzando il formato con forwardslash (/);
- *COMHOME_BACKSLASH*: percorso directory che contiene com/zappanoseda, utilizzando il formato con backslash (\);
- *COMHOME_FORWARDSLASH*: percorso directory che contiene com/zappanoseda, utilizzando il formato con forwardslash (/);
- *ACTIVATION_JAR*: file jar di JavaBeans Activation Framework;
- JAVAMAIL_JAR: file jar di JavaMail;
- *SERVICEHOME_BACKSLASH*: percorso in cui è stata installata l'agenzia domotica, utilizzando il formato con backslash (\);
- *SERVICEHOME_FORWARDSLASH*: percorso in cui è stata installata l'agenzia domotica, utilizzando il formato con forwardslash (/);
- *SERVLET_JAR*: file jar delle servlet.

B.3 Compilazione del sistema

Abbiamo creato dei file batch, in modo da rendere più agevole l'operazione di compilazione dell'intero sistema.

Per prima cosa bisogna aver configurato il sistema come illustrato nel paragrafo precedente. Poi si compila il package com.zappanoseda eseguendo il file c.bat presente nella directory in cui è stato installato (sun\com\zappanoseda). Successivamente si può passare a compilare i dispositivi simulati, i sistemi di telecontrollo (versione programma ed e-mail), di gestione allarmi e scenari e di programmazione azioni, eseguendo il file c.bat presente nella directory sun\casa. Infine, si può compilare la versione web del sistema di telecontrollo eseguendo il file c.bat che si trova sul web-server.

B.4 Avvio del sistema

Per avviare il sistema bisogna eseguire nell'ordine le seguenti operazioni:

1. avviare il web-server con le politiche di sicurezza corrette, in modo che la servlet abbia il permesso di collegarsi alla comunità Jini (eseguire il file avvia.bat presente nella directory sun\tomcat\bin);
2. avviare il demone RMI (RMID) utilizzando il file runRMI.bat che si trova nella directory sun\casa;
3. aspettare qualche secondo e poi far partire il lookup service di Jini (reggie) eseguendo runLUS.bat presente in sun\casa;
4. spostarsi in sun\casa\server-side e mandare in esecuzione i dispositivi che si vuole (rForno.bat, rSMS.bat, rLavatrice.bat, …);

5. avviare il sistema di gestione allarmi e scenari eseguendo il file rGestoreAllarmi.bat;

A questo punto l'agenzia domotica è stata avviata. Per utilizzare il sistema di telecontrollo via web è sufficiente aprire il browser e impostare l'indirizzo della servlet (http://localhost:8080/servlet/ServletCasa); mentre per avviare la versione programma, andare nella directory sun\casa\client-side ed eseguire il file rTeleControllo.bat. Se si vuole attivare la versione e-mail, eseguire rMonitoraggioEmail.bat.

Infine, per utilizzare il sistema di programmazione delle azioni, eseguire il file rToDo.bat che si trova nella cartella sun\casa\client-side.

B.5 Uso del sistema

Nei paragrafi seguenti viene illustrato come utilizzare il sistema di telecontrollo (nelle tre versioni disponibili) e quello di programmazione delle azioni.

B.5.1 Sistema di telecontrollo (versione programma)

La versione programma del sistema di telecontrollo, permette di telecontrollare e monitorare tutti i dispositivi domestici presenti nella casa, per avviare il programma andare nella cartella sun\casa\client-side ed eseguire rTeleControllo.bat. Nella figura B.2 è riportata la schermata principale del programma.

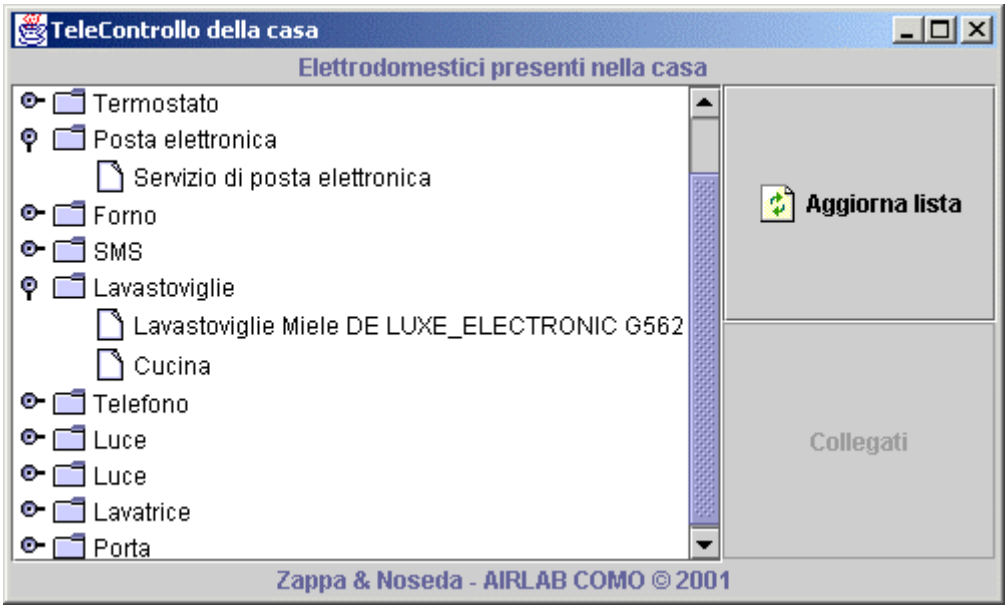

Figura B.2: Finestra principale del programma di telecontrollo

Nella parte di sinistra si trova l'elenco di tutti i dispositivi attivi, con la loro descrizione e la stanza in cui sono situati. Nella parte di destra si hanno due bottoni, il primo aggiorna la lista dei dispositivi, mentre il secondo è attivo quando si è scelto uno degli elettrodomestici dell'elenco e permette di collegarsi al device. Quando questo bottone è premuto viene presentata un nuova finestra con l'interfaccia grafica specifica per monitorare e telecontrollare il dispositivo selezionato (si veda la figure B.3 sotto riportata per degli esempi).

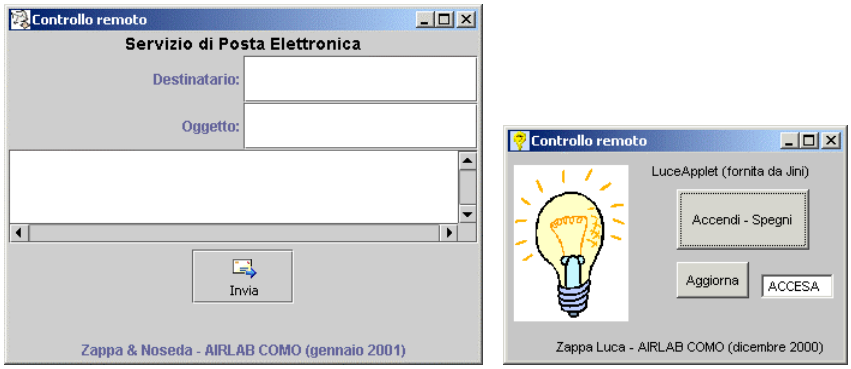

Figura B.3: Esempi di interfacce utenti di telecontrollo

B.5.2 Sistema di telecontrollo (versione web)

Per utilizzare il telecontrollo via web, aprire il browser e impostare l'indirizzo della servlet (http://localhost:8080/servlet/ServletCasa); viene caricata la pagina iniziale (figura B.4) che contiene la lista di tutti i dispositivi, dell'agenzia domotica, al momento attivi.

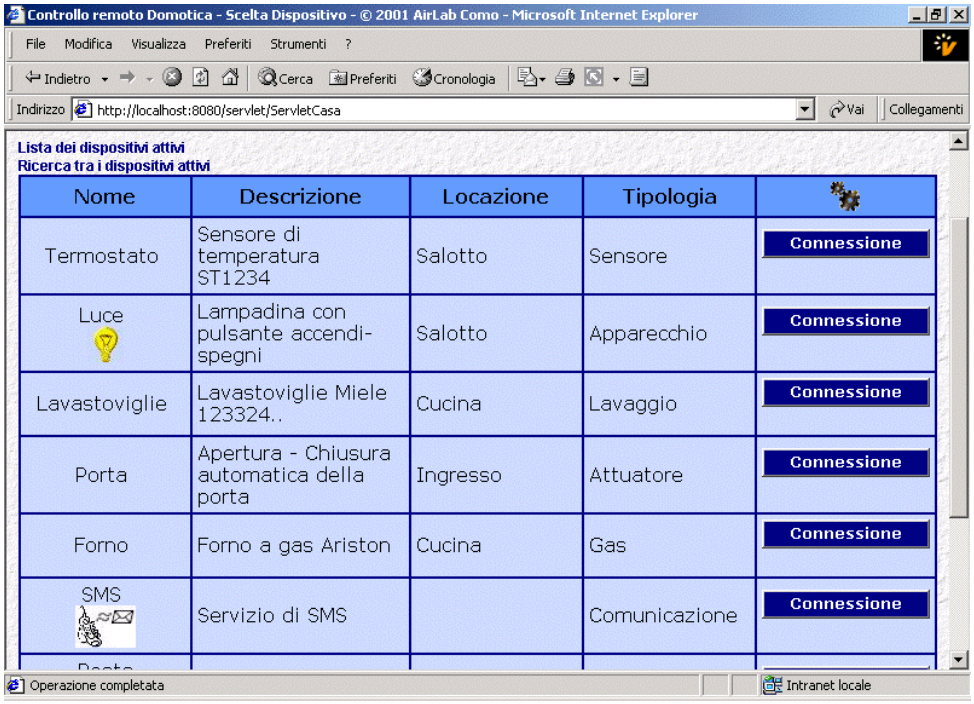

Figura B.4: Pagina principale telecontrollo via web

Premendo il pulsante “Connessione” viene monitorato lo stato del dispositivo corrispondente e l’elenco delle operazioni che è possibile eseguire su tale device (figura B.5).

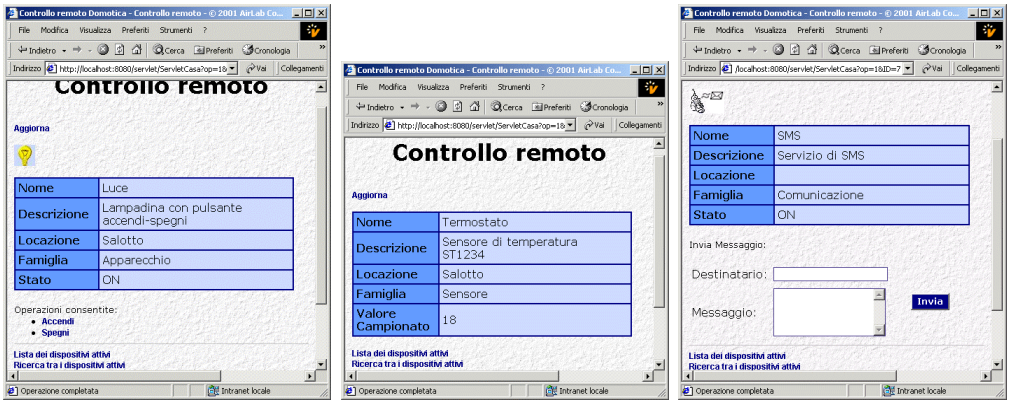

Figura B.5: Esempi di telecontrollo via web

Premendo il link “Ricerca tra i dispositivi attivi” è possibile eseguire una ricerca dei dispositivi attivi che rispettano i criteri impostati (figura B.6).

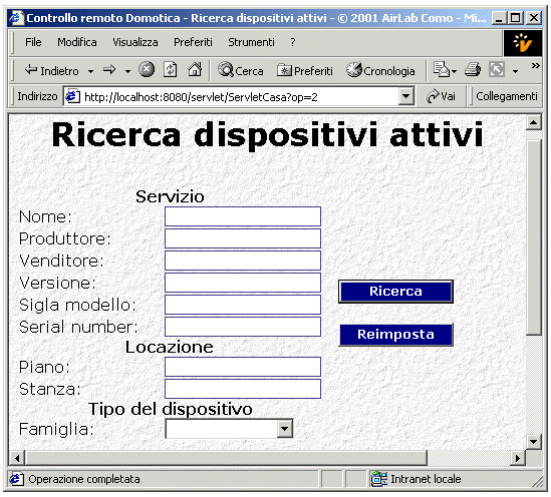

Figura B.6: Pagina di ricerca tra i dispositivi attivi

B.5.3 Sistema di telecontrollo (versione e-mail)

Per attivare il sistema di telecontrollo via e-mail, andare nella cartella sun\casa\client-side ed eseguire il file rMonitoraggioEmail.bat; il programma si collega alla casella di posta elettronica indicata nel file in attesa dei messaggi inviati dall’utente.

Per monitorare e controllare l’agenzia domotica, è sufficiente utilizzare un qualsiasi applicativo di posta elettronica ed inviare degli opportuni messaggi all’indirizzo e-mail della casa (nel nostro esempio controllo.casa@katamail.com). I comandi devono essere inseriti nell’oggetto del messaggio e hanno la seguente sintassi:

- LISTA: risponde al mittente con un messaggio nel quale sono indicati tutti i dispositivi domestici attivi, con il loro nome, la descrizione, la locazione, i vari stati e le operazioni possibili su tale

device, inoltre viene associato ad ogni dispositivo un identificatore numerico (ID) in modo che successivamente possa essere usato per controllarlo (figura B.7). Si possono anche indicare dei parametri opzionali per ricercare un insieme di dispositivi con caratteristiche comuni, i possibili parametri sono:

  - NOME: nome del dispositivo;
  - PRODUTTORE: produttore del dispositivo;
  - VENDITORE: venditore del dispositivo;
  - VERSIONE: versione del dispositivo;
  - MODELLO: modello del dispositivo;
  - SERIAL: numero seriale del dispositivo;
  - PIANO: piano in cui i dispositivi sono situati;
  - STANZA: stanza in cui i dispositivi sono situati;
  - FAMIGLIA: tipologia di dispositivi: sensore, apparecchio, attuatore, intrattenimento, lavaggio, gas, comunicazione;

- CONTROLLO ID=<num> OP=<num>: viene invocata l'operazione indicata sul dispositivo a cui era stato associato l'ID specificato, il mittente riceverà un messaggio nel quale è indicata se l'operazione è andata a buon fine e il nuovo stato assunto dal dispositivo in questione (figura B.8);
- se nell'oggetto non viene inserito nessun carattere il sistema risponde con un messaggio di aiuto nel quale è indicato come utilizzare il sistema.

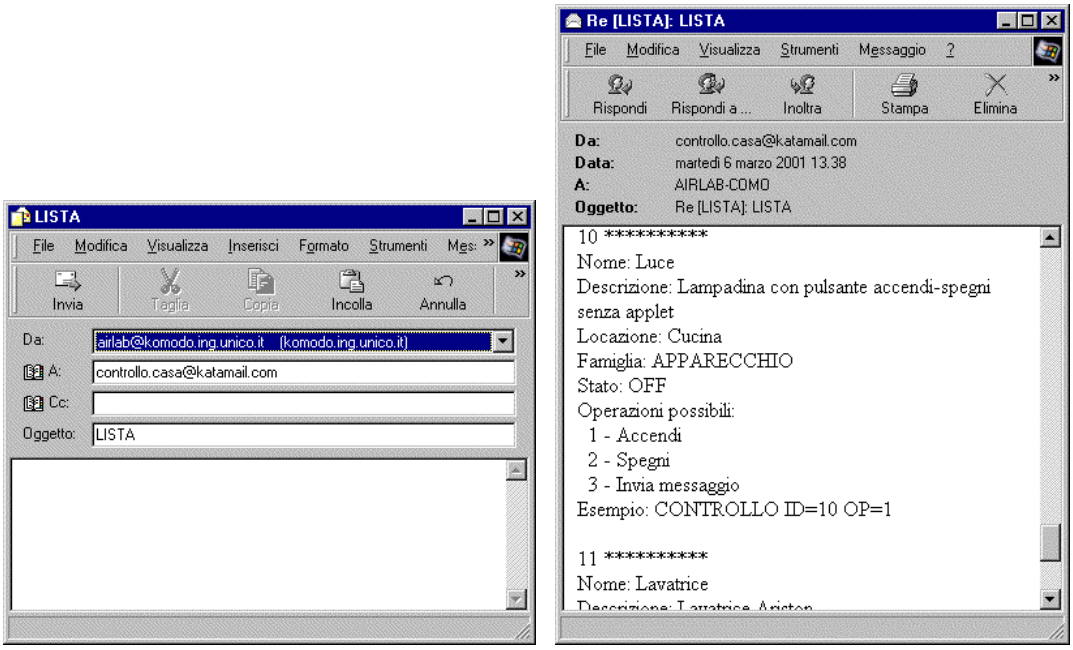

Figura B.7: Richiesta lista dispositivi e risposta

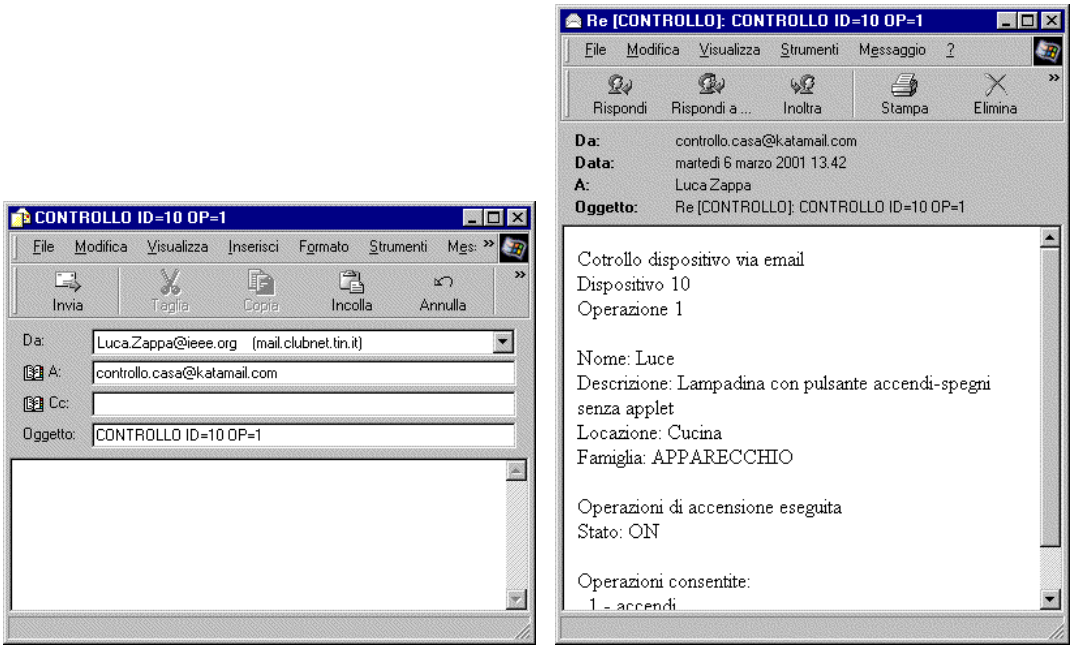

Figura B.8: Richiesta di accensione della luce e risposta

B.5.4 Sistema di programmazione delle azioni

Il sistema di programmazione delle azioni permette di eseguire delle operazioni sugli elettrodomestici della casa in un determinato istante temporale. Per avviare il programma portarsi nella cartella sun\casa\client-side ed eseguire rToDo.bat; nella figura B.9 è riportata la schermata principale del programma.

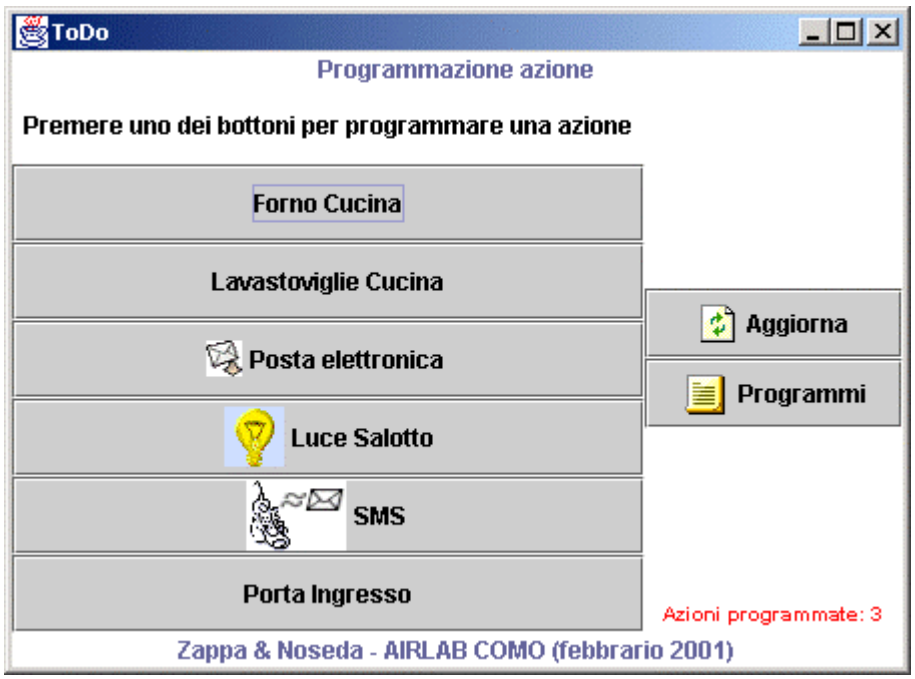

Figura B.9: Finestra principale sistema programmazione azione

Nella parte di sinistra si trova l'elenco dei dispositivi domestici attivi. Premendo il bottone corrispondente si accede alla finestra che permette la creazione di una nuova azione, dove bisogna impostare l'istante temporale (giorno, mese, anno, ora, minuti, secondi) e l'operazione desiderata, che dipende dalla tipologia del dispositivo considerato (figura B.10).

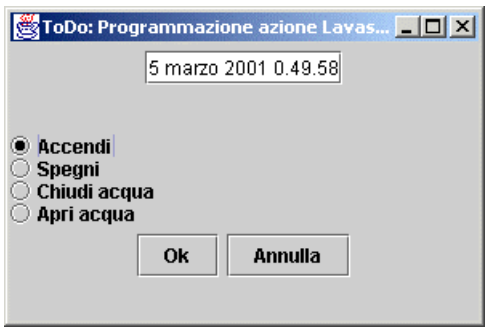

Figura B.10: Impostazione di una nuova azione

Nella parte di destra si hanno due bottoni, il primo serve per aggiornare la lista dei dispositivi, mentre il secondo per vedere le azioni che sono state programmate e che ancora devono essere eseguite (figura B.11). In fondo a destra è inoltre indicato il numero di azioni che sono state programmate e che devono ancora essere eseguite.

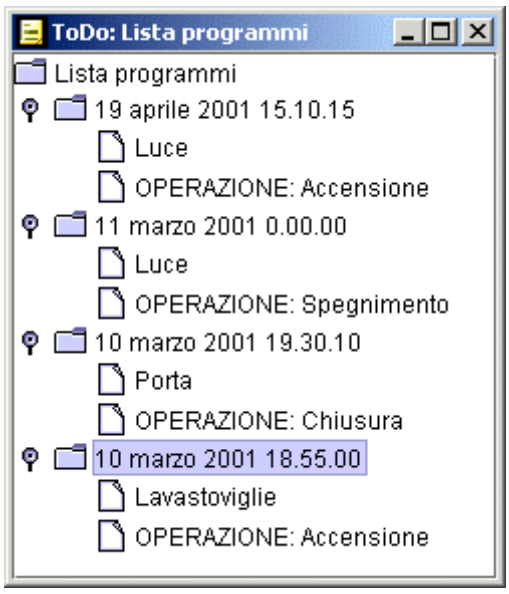

Figura B.11: Lista dei programmi impostati

C

LISTATI

In questa appendice vengono riportati i sorgenti Java più significativi dei sistemi e dei dispositivi domestici simulati che compongono l'agenzia domotica da noi sviluppata. Abbiamo deciso di non riportare tutto il codice sorgente, in quando il sistema è estremamente vasto, come mostrato nella tabella 7.1. in cui viene indicato per ogni modulo il numero di linee di codice (LOC: Lines Of Code).

C.1 Package com.zappanoseda

Il package com.zappanoseda contiene le classi e le interfacce di utilità generale utilizzate da tutti i sistemi dell'agenzia domotica.

C.1.1 L'interfaccia Dispositivo

```
package com.zappanoseda;

import java.rmi.*;
import net.jini.core.event.*;
```

```
import com.zappanoseda.*;

/**
* Dispositivo -- Interfaccia implementata da un generico
dispositivo.
* Il dispositivo è l'elettrodomestico più generico.
* @author Luca Zappa & Alberto Noseda
* @version 1.0
*/

public interface Dispositivo extends Remote {
/**
* Costante che indica l'ok.
*/
  public static final int OK = 0;

/**
* Costante che indica la presenza di un errore.
*/
  public static final int ERRORE = 1;

/**
* Operazione che ritorna il nome dell'elettrodomestico.
*/
  public String nome() throws RemoteException;

/**
* Operazione che ritorna la descrizione dell'elettrodomestico.
*/
  public String descrizione() throws RemoteException;

/**
* Operazione che ritorna la stanza in cui è situato
l'elettrodomestico.
*/
  public String locazione() throws RemoteException;

/**
* Operazione per gestire l'evento.
*/
public EventRegistration addRemoteEventListener(
  RemoteEventListener listener,
  MarshalledObject handback ) throws RemoteException;
}
```

C.1.2 L’interfaccia Sensore

```
package com.zappanoseda;

import java.rmi.*;
import com.zappanoseda.*;

/**
* Sensore -- Interfaccia implementata da un generico sensore.
* Esempi di sensore:
```

```
* sensore di temperatura, sensore di luminosità, sensore di
posizione, ...
* @author Luca Zappa & Alberto Noseda
* @version 1.1
*/

public interface Sensore extends Dispositivo {
/**
* Operazione che ritorna il valore campionato dal sensore.
*/
 public int campionamento() throws RemoteException;
}
```

C.1.3 L'interfaccia Attuatore

```
package com.zappanoseda;

import java.rmi.*;
import com.zappanoseda.*;

/**
* Attautore -- Interfaccia implementata da un dispositivo di tito
attuatore.
* Esempi di attuatori:
* apertura porte, finestre, tapparelle, ...
* @author Luca Zappa & Alberto Noseda
* @version 1.0
*/

public interface Attuatore extends Dispositivo {

/**
* Costante che indica lo stato chiuso.
*/
  public static final int CHIUSO = 0;

/**
* Costante che indica lo stato aperto.
*/
  public static final int APERTO = 1;

/**
* Comando di apertura.
*/
  public void apri() throws RemoteException;

/**
* Comando di chiusura.
*/
  public void chiudi() throws RemoteException;

/**
* Ritorna lo stato: CHIUSO o APERTO.
*/
  public int stato() throws RemoteException;
```

```
}
```

C.1.4 L'interfaccia Apparecchio

```
package com.zappanoseda;

import java.rmi.*;
import com.zappanoseda.*;

/**
* Apparecchio -- Interfaccia implementata da un generico
apparecchio.
* Esempi di apparecchi:
* lampadina con interruttore, ...
* @author Luca Zappa</a> & Alberto Noseda
* @version 1.0
*/

public interface Apparecchio extends Dispositivo {

/**
* Costante che indica lo stato acceso.
*/
  public static final int ON = 1;

/**
* Costante che indica lo stato spento.
*/
  public static final int OFF = 0;

/**
* Comando per accenderere l'apperacchio.
*/
  public void on() throws RemoteException;

/**
* Comando per spegnere l'apperacchio.
*/
  public void off() throws RemoteException;

/**
* Ritorna lo stato dell'aparecchio: ON o OFF.
*/
  public int stato() throws RemoteException;
}
```

C.1.5 L'interfaccia Lavaggio

```
package com.zappanoseda;

import java.rmi.*;
```

```
import com.zappanoseda.*;

/**
* Lavaggio -- Interfaccia implementata da un generico apperecchio ad
acqua.
* Esempi di apparecchi di tipo Lavaggio:
* lavatrice, lavastoviglie, ...
* @author Luca Zappa &  Alberto Noseda
* @version 1.0
*/

public interface Lavaggio extends Apparecchio {

/**
* Costante che indica lo stato aperto del rubinetto dell'acqua.
*/
  public static final int ACQUA_APERTA = 1;

/**
* Costante che indica lo stato chiuso del rubinetto dell'acqua.
*/
  public static final int ACQUA_CHIUSA = 0;

/**
* Operazione che chiude il rubinetto dell'acqua.
*/
  public void chiudi_acqua() throws RemoteException;

/**
* Operazione che apre il rubinetto dell'acqua.
*/
  public void apri_acqua() throws RemoteException;

/**
* Operazione che ritorna lo stato del rubinetto dell'acqua:
ACQUA_APERTA o ACQUA_CHIUSA.
*/
  public int stato_acqua() throws RemoteException;
}
```

C.1.6 L'interfaccia Gas

```
package com.zappanoseda;

import java.rmi.*;
import com.zappanoseda.*;
import net.jini.core.event.*;

/**
* Gas -- Interfaccia implementata da un generico apperecchio a gas.
* Esempio di apperacchi a gas:
* forno, caldaia, piano cottura, scaldabagno, ...
* @author Luca Zappa & Alberto Noseda
* @version 1.0
*/
```

```
public interface Gas extends Apparecchio {
/**
* Costante che indica lo stato aperto del rubinetto del gas.
*/
  public static final int GAS_APERTO = 1;

/**
* Costante che indica lo stato chiuso del rubinetto del gas.
*/
  public static final int GAS_CHIUSO = 0;

/**
* Operazione che chiude il rubinetto del gas.
*/
  public void chiudi_gas() throws RemoteException;

/**
* Operazione che apre il rubinetto del gas.
*/
  public void apri_gas() throws RemoteException;

/**
* Operazione che ritorna lo stato del rubinetto del gas: GAS_APERTO
o GAS_CHIUSO.
*/
  public int stato_gas() throws RemoteException;
}
```

C.1.7 L'interfaccia Comunicazione

```
package com.zappanoseda;

import java.rmi.*;
import com.zappanoseda.*;

/**
* Comunicazione -- Interfaccia implementata da un generico
apperecchio di comunicazione.
* Esempi di apparecchi di tipo comunicazione:
* telefono fisso, telefono mobile, posta elettronica, fax,
* @author Luca Zappa & Alberto Noseda
* @version 1.0
*/

public interface Comunicazione extends Apparecchio {

/**
* Comandi per inviare il messaggio al destinatario.
*/
  public int invia( String destinatario, String messaggio) throws
RemoteException;
}
```

C.1.8 Classe ApparecchioApplet

```
package com.zappanoseda;

import net.jini.core.entry.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.border.*;
import com.zappanoseda.*;

/**
* ApparecchioApplet -- Applet standard per il controllo degli
apparecchi
* @author Luca Zappa & Alberto Noseda
* @version 1.0
*/

public class ApparecchioApplet extends JApplet implements Entry{

  public Apparecchio myServerInterface;
  JButton accendi = new JButton("Accendi");
  JButton spegni = new JButton("Spegni");
  JTextField stato;
  JLabel et1 =
    new JLabel("<html><b> Apparecchio ", SwingConstants.CENTER);
  JLabel et2 =
    new JLabel(" Zappa & Noseda - AIRLAB COMO (gennaio 2001) ",
SwingConstants.CENTER);
  JLabel etNome, etDescrizione , etLocazione;


  public ApparecchioApplet () {
        // Costruttore
  }

  public ApparecchioApplet (Apparecchio myServerInterfaceIn) {
    myServerInterface = myServerInterfaceIn;
  }


  public void init() {
    Container cp = getContentPane();
    cp.removeAll();
    cp.setLayout(new BorderLayout());
    cp.add(BorderLayout.NORTH,et1);
    cp.add(BorderLayout.SOUTH,et2);
    try {
      JPanel panCentr = new JPanel( new GridLayout(2,1) );
      JPanel p3 = new JPanel(new GridLayout(3,1));
      etNome = new JLabel("Nome Apparecchio: " +
myServerInterface.nome() );
      p3.add(etNome);
      etDescrizione = new JLabel("Descrizione: " +
myServerInterface.descrizione() );
      p3.add(etDescrizione);
```

```
      etLocazione = new JLabel("Stanza: " +
myServerInterface.locazione() );
      p3.add(etLocazione);
      panCentr.add(p3);

      JPanel p1 = new JPanel();
      p1.setBorder(new javax.swing.border.TitledBorder("Accensione
elettrica"));
      if ( myServerInterface.stato() == Apparecchio.OFF ) {
        stato = new JTextField( "Spento" ,8);
      }
      else {
        stato = new JTextField( "Acceso" ,8);
      }
      stato.setHorizontalAlignment(JTextField.CENTER);
      stato.setEditable(false);
      p1.add(stato);

      accendi.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          try {
            myServerInterface.on();
            if ( myServerInterface.stato() == Apparecchio.OFF ) {
              stato.setText( "Spento" );
            }
            else {
              stato.setText( "Acceso" );
            }
          } catch (Exception ex) { System.out.println("Apparecchio:
ApparecchioApplet.init(): Exception " + ex); }
        } } );
      p1.add(accendi);

      spegni.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          try {
            myServerInterface.off();
            if ( myServerInterface.stato() == Apparecchio.OFF ) {
              stato.setText( "Spento" );
            }
            else {
              stato.setText( "Acceso" );
            }
          } catch (Exception ex) { System.out.println("Apparecchio:
ApparecchioApplet.init(): Exception " + ex); }
        } } );
      p1.add(spegni);
      panCentr.add(p1);

      cp.add(BorderLayout.CENTER,panCentr);
    } catch (Exception e) { System.out.println("Apparecchio:
ApparecchioApplet.init(): Exception " + e); }
  }

}
```

C.1.9 Classe EventoCasa

```
package com.zappanoseda;

import java.rmi.*;
import com.zappanoseda.*;

/**
* Evento -- Interfaccia che assegna gli ID degli allarmi.
* @author Luca Zappa & Alberto Noseda
* @version 1.0
*/

public final class EventoCasa {
/**
* Costante che rappresenta l'evento: "sconnessione del servizio da
Jini".
*/
  public static final long SCONNESSIONE = 0;

/**
* Costante che rappresenta l'evento: "rottura dispositivo".
*/
  public static final long ROTTURA_DISPOSITIVO = 1;

/**
* Costante che rappresenta l'evento: "perdita acqua".
*/
  public static final long PERDITA_ACQUA = 2;

/**
* Costante che rappresenta l'evento: "perdita gas".
*/
  public static final long PERDITA_GAS = 3;

/**
* Costante che rappresenta l'evento: "mancanza linea di
comunicazione".
*/
  public static final long MANCANZA_LINEA = 4;

/**
* Costante che rappresenta l'evento: "temperatura troppo alta".
*/
  public static final long TEMPERATURA_ALTA = 10;

/**
* Costante che rappresenta l'evento: "temperatura troppa bassa".
*/
  public static final long TEMPERATURA_BASSA = 11;

/**
* Costante che rappresenta l'evento: "rilevazione fumo".
*/
  public static final long RILEVAZIONE_FUMO = 12;

/**
```

```
* Costante che rappresenta l'evento: "rilevazione movimento".
*/
  public static final long RILEVAZIONE_MOVIMENTO = 13;

/**
* Costante che rappresenta l'evento: "rilevazione gas in casa".
*/
  public static final long RILEVAZIONE_GAS = 14;

/**
* Costante che rappresenta l'evento: "rilevazione acqua in casa".
*/
  public static final long RILEVAZIONE_ACQUA = 15;


/**
* Funzione che ritorna la descrizione dell'evento associata ad
IDevento.
*/
  public static String descrizioneEvento( long IDevento ) {
    if ( IDevento == SCONNESSIONE ) {
      return "sconnessione";
    }
    else if ( IDevento == ROTTURA_DISPOSITIVO ) {
      return "rottura dispositivo";
    }
    else if ( IDevento == PERDITA_ACQUA ) {
      return "perdita acqua";
    }
    else if ( IDevento == PERDITA_GAS ) {
      return "perdita gas";
    }
    else if ( IDevento == MANCANZA_LINEA ) {
      return "mancanza linea";
    }
    else if ( IDevento == TEMPERATURA_ALTA ) {
      return "temperatura alta";
    }
    else if ( IDevento == TEMPERATURA_BASSA ) {
      return "temperatura bassa";
    }
    else if ( IDevento == RILEVAZIONE_FUMO ) {
      return "rilevato fumo";
    }
    else if ( IDevento == RILEVAZIONE_MOVIMENTO ) {
      return "rilevato movimento";
    }
    else if ( IDevento == RILEVAZIONE_GAS ) {
      return "rilevato gas";
    }
    else if ( IDevento == RILEVAZIONE_ACQUA ) {
      return "rilevata acqua";
    }
    else {
      return "sconosciuto";
    }
  }
}
```

C.1.10 Classe Icona

```
package com.zappanoseda;

import net.jini.core.entry.*;
import javax.swing.*;
import java.awt.*;

/**
* Icona -- Classe che permette di inserire negli attributi del
servizio un'icona
* @author Luca Zappa & Alberto Noseda
* @version 1.0
*/

public class Icona implements Entry {

  public ImageIcon icon;

  public Icona () {
  }

  public Icona (String file) {
    icon =  new ImageIcon(file);
  }

  public Image getImage () {
    return  icon.getImage();
  }

  public ImageIcon getIcon () {
    return  icon;
  }
}
```

C.2 Simulazione dei dispositivi domestici

In questo paragrafo viene illustrato come realizzare un dispositivo domestico a partire dal package com.zappanoseda presentato nel paragrafo precedente.

C.2.1 Luce

```
import net.jini.core.entry.*;
import net.jini.core.lookup.*;
import net.jini.core.discovery.*;
import net.jini.lookup.entry.*;
import net.jini.discovery.*;
import com.sun.jini.lookup.*;
import com.sun.jini.lease.*;
import net.jini.core.event.*;
import com.sun.jini.lookup.entry.LookupAttributes;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.applet.*;
import com.zappanoseda.*;

/**
* Luce -- Jini Server Class della luce
* @author Luca Zappa & Alberto Noseda
* @version 1.1
*/

public class Luce extends UnicastRemoteObject
 implements Apparecchio, ServiceIDListener, Serializable {

 protected static BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
 protected long countEvent = 0L;
 // Elenco dei servizi a cui spedire gli eventi
 protected Dictionary listeners = new Hashtable();

 static String nome = "Luce";
 static String descrizione = "Lampadina con pulsante accendi-
spegni";
 static String stanza = "Salotto";

 static Entry[] aeAttributes =  { // Descrizione del servizio
       // aeAttributes[0]
     new Name( nome ),       // Nome (.name)
       // aeAttributes[1]
     new ServiceInfo(nome,   // Nome          (.name)
```

```
          "Zappa & Noseda", // Produttore    (.manufacturer)
          "AIRLAB COMO",    // Venditore     (.vendor)
          "1.0",            // Versione      (.version)
          "on-off",         // Sigla modello (.model)
          "LU001"),         // Serial number (.serialNumber)
      // aeAttributes[2]
    new Location("0",       // Piano    (.floor)
                 stanza,    // Stanza   (.room)
                 "Casa1"),  // Edificio (.building)
      // aeAttributes[3]
    new Comment( descrizione ),    // Descrizione (.comment)
      // aeAttributes[4]

  }; // 4 attributi registrati, l'applet viene aggiunta nel main

int Stato = OFF; // Stato lampadina: ON - OFF


public Luce() throws RemoteException {
  super ();
  Stato = OFF;
}


public String nome() throws RemoteException {

  System.out.println ("Luce: richiesta nome");
  return nome;
}


public String descrizione() throws RemoteException {

  System.out.println ("Luce: richiesta descrizione");
  return descrizione;
}


public String locazione() throws RemoteException {

  System.out.println ("Luce: richiesta locazione");
  return stanza;
}


public int stato() throws RemoteException {

  System.out.println ("Luce: richiesta stato"+Stato);
  return Stato;
}


public void on() throws RemoteException {

  if (Stato == ON)
    System.out.println ("Luce: richiesta di accensione INUTILE");
  else
    System.out.println ("Luce: richiesta di accensione");
  Stato = ON;
```

```
 }


 public void off() throws RemoteException {

   if (Stato == OFF)
     System.out.println ("Luce: richiesta di spegnimento INUTILE");
   else
     System.out.println ("Luce: richiesta di spegnimento");
   Stato = OFF;
 }


 public void run() throws IOException {

   // Terminazione del servizio
   System.out.println ("Premi invio per disattivare il servizio");
   System.out.flush();
   String s = in.readLine();
   // generazione dell'evento sconnessione
   genera_evento( EventoCasa.SCONNESSIONE );
 }

 // Procedura di generazione di un evento
 private void genera_evento( long IDevento ) throws IOException {
   int i = 0;
   Enumeration enum = listeners.keys();
   while ( enum.hasMoreElements() )
   {
     i++;
     RemoteEventListener listener = (RemoteEventListener)
enum.nextElement();
     RemoteEvent event = new
RemoteEvent(this,IDevento,countEvent,(MarshalledObject)listeners.get
(listener));
     try
       { listener.notify(event); }
     catch (UnknownEventException e)
       { e.printStackTrace(); }
     catch (RemoteException e)
       { e.printStackTrace(); }
   }
   System.out.println ("Luce: Evento ( " + IDevento + "-"
       + EventoCasa.descrizioneEvento(IDevento) + " ) inviato a " +
i + " servizi.");
   countEvent++;
 }

 // Aggiunta di un servizio alla lista dei servizi a cui consegnare
gli eventi generati
 public EventRegistration addRemoteEventListener(
RemoteEventListener listener,
                                                 MarshalledObject
handback) throws RemoteException {

   try
     {
       listeners.put(listener,handback);
       System.out.println ("Luce: RemoteEventListener aggiunto.");
```

```
       return new
EventRegistration(EventoCasa.SCONNESSIONE,this,null,countEvent);
     }
   catch (Exception e)
     {
       e.printStackTrace();
       return null;
     }
 }


 public void serviceIDNotify (ServiceID serviceId) { } //
ServiceIDListener


 public static void main (String[] args) {
  int i;
  int iPort;
  String sHost;
  Luce myServer;
  LookupLocator lookup;
  JoinManager joinmanager;
  ServiceID id;
  ServiceMatches matches;
  ServiceRegistrar registrar;


  try {
/* Impostazione del security manager per permettere a RMI
   class loader di raggiungere il codebase per le classi
   che non sono disponibili localmente
   ============================================  */
   System.setSecurityManager (new RMISecurityManager ());

/* Crezione degli attributi (un array di oggetti entry)
   che descrivono il servizio, usato per registrare il
   servizio presso il lookup service.
   ============================================== */
   myServer = new Luce();

   // Aggiunta dell'attributo che contiene l'applet Java per il
controllo in remoto

   Entry[] attributoApplet = {
     new LuceApplet (myServer)
   };

   aeAttributes =
LookupAttributes.add(aeAttributes,attributoApplet);

   // Aggiunta dell'attributo che contiene l'icona
   Icona ic = new Icona ("luce.gif");
   if ( ic.icon.getIconHeight() < 0 ) // L'icona non è stata trovata
     System.out.println("Luce: ICONA NON TROVATA!");
   else {
     Entry[] attributoIcona = {
       ic
     };
```

```
     aeAttributes =
LookupAttributes.add(aeAttributes,attributoIcona);
   }

   joinmanager = new JoinManager
   (
      myServer,
      aeAttributes,
      myServer,
      new LeaseRenewalManager ()
   );

   System.out.println("Luce: JoinManager = " + joinmanager);

/* Ricerca del Jini Lookup Service (reggie), sul localhost
   o, se presente, dove indicato nella riga di comando.
   ================================================  */
   System.out.println ();
   if ( args.length > 0 ) {
     System.out.println ("ARGOMENTO : " + args[0]);
     lookup = new LookupLocator ( args[0] );
   }
   else {
     lookup = new LookupLocator ( "jini://localhost" );
   }
   sHost = lookup.getHost ();
   iPort = lookup.getPort ();
   System.out.println ("Luce: LookupLocator = " + lookup);
   System.out.println ("Luce: LookupLocator.host = " + sHost);
   System.out.println ("Luce: LookupLocator.port = " + iPort);

/* Caricamento del ServiceRegistar (la classe che permette
   di interagire con il lookup service).
   ================================================ */
   registrar = lookup.getRegistrar();
   id = registrar.getServiceID ();
   System.out.println ("Luce: ServiceRegistrar = " + registrar);
   System.out.println ("Luce: ServiceID = " + id);
   System.out.println ("*----------------------------*");

   myServer.run();

   try {Thread.sleep (2000);} catch (Exception e) {}
   // Sconnessione dalla comunità Jini
   joinmanager.terminate();
   System.exit(0);

  } catch (Exception e) { System.out.println("Luce: Luce.main():
Exception " + e); }
 }
}
```

C.3 Sistema di telecontrollo e monitoraggio

C.3.1 Versione programma

```
import net.jini.core.entry.*;
import net.jini.core.event.*;
import net.jini.core.lookup.*;
import net.jini.core.discovery.*;
import net.jini.lookup.entry.*;
import com.sun.jini.lookup.*;
import java.rmi.*;
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
import javax.swing.tree.*;
import com.zappanoseda.*;

/**
* TeleControllo -- Telecntrollo dispositivi
* @author <a href="mailto:Luca.Zappa@ieee.org">Luca Zappa</a> & <a
href="mailto:anoseda@komodo.ing.unico.it">Alberto Noseda</a>
* @version 2.0
*/
class TeleControllo extends JFrame {

  static JFrame finestra;
  JLabel
    titolo = new JLabel( " Elettrodomestici presenti nella casa " ,
SwingConstants.CENTER ),
    fine = new JLabel( " Zappa & Noseda - AIRLAB COMO © 2001 " ,
SwingConstants.CENTER );
  JButton aggiorna = new JButton( "Aggiorna lista" , new
ImageIcon("aggiorna.gif"));
  JScrollPane pannelloCentrale;
  JPanel panEast = new JPanel(new GridLayout(2,1));
  JButton bottone = new JButton( "Collegati" );
  JTree tree;
  DefaultTreeModel model;
  Container cp;

  ServiceMatches listaDisp;
  ServiceRegistrar registrar;


  class SelezioneDispositivo implements TreeSelectionListener {
      int precedente = -1;
```

```
      public void valueChanged(TreeSelectionEvent e) {
        DefaultMutableTreeNode chosen = (DefaultMutableTreeNode)
tree.getLastSelectedPathComponent();
        if ( tree.getSelectionPath().getPathCount() == 3 )
          chosen = (DefaultMutableTreeNode) chosen.getParent();
        int sel =  model.getIndexOfChild(model.getRoot(), chosen);
        if ( sel < 0 ) bottone.setEnabled(false);
        if ( ( sel >= 0 ) && ( sel != precedente ) ) {
          precedente = sel;
          try {
            panEast.remove(bottone);
            bottone = new JButton( "Collegati" );
            bottone.setEnabled(true);
            Dispositivo dispSel = (Dispositivo)
listaDisp.items[sel].service;
            System.out.println ("selezione  numero " + sel + " " +
dispSel.nome());

            Azione al = new Azione();
            Object object;

/*  Cilco attributi dei servizi trovati.
   ====================================================  */

            for (int j = 0; j <
listaDisp.items[sel].attributeSets.length; j++)
            {
              object = listaDisp.items[sel].attributeSets[j];
              System.out.println ("TeleControllo: " + sel + ":" + j
+ ": attrib Set: "
                + object);

/*  Se la classe di un attibuto è istanza di una Applet associa
    al bottone l'apertura di tale Applet.
    Se invece si tratta di una icona la visualizza.
   ====================================================  */

              if (object instanceof Applet) {
                System.out.println ("TeleControllo: Caricamento
Applet specifica");
                al.applet = (Applet) object;
              }
              else if (object instanceof Icona) {
                System.out.println ("TeleControllo: Caricamento
icona specifica");
                Icona icona = (Icona) object;
                bottone.setIcon( icona.getIcon() );
                al.iconcina = icona.getImage() ;
              }
            }

/* Se negli attributi del servizio non è stata trovata nessuna
   Applet viene carica quella di default dell'interfaccia
   implementata dal servizio
   ====================================================  */

            if ( al.applet == null ) {    // nessuna applet trovata
              System.out.println ("TeleControllo: Caricamento Applet
di default");
```

```
              if (dispSel instanceof Sensore) {
                al.applet = new SensoreApplet ( (Sensore) dispSel );
                System.out.println ("SensoreAppelt caricata");
              }
              else if (dispSel instanceof Apparecchio) {
                if (dispSel instanceof Gas) {
                  al.applet = new GasApplet ( (Gas) dispSel );
                  System.out.println ("GasAppelt caricata");
                }
                else if (dispSel instanceof Lavaggio) {
                  al.applet = new LavaggioApplet ( (Lavaggio)
dispSel );
                  System.out.println ("LavaggioAppelt caricata");
                }
                else if (dispSel instanceof Intrattenimento) {
                  al.applet = new IntrattenimentoApplet (
(Intrattenimento) dispSel );
                  System.out.println ("IntrattenimentoAppelt
caricata");
                }
                else if (dispSel instanceof Comunicazione) {
                  al.applet = new ComunicazioneApplet (
(Comunicazione) dispSel );
                  System.out.println ("ComunicazioneAppelt
caricata");
                }
                else {
                  al.applet = new ApparecchioApplet ( (Apparecchio)
dispSel );
                  System.out.println ("ApparecchioAppelt caricata");
                }
              }
              else if (dispSel instanceof Attuatore) {
                al.applet = new AttuatoreApplet ( (Attuatore)
dispSel );
                System.out.println ("AttuatoreAppelt caricata");
              }
              else {
                System.out.println ("DispositivoAppelt caricata");
              }
            }
            bottone.addActionListener(al);
            panEast.add( bottone );
          }
          catch (Exception ex) { System.out.println ("TeleControllo:
exception: " + ex); }
        }
        finestra.validate();
      }
  }

  class Azione implements ActionListener {
    public Applet applet;
    public Image iconcina;


    public void actionPerformed(ActionEvent e) {
      try {
```

```
        System.out.println ("TeleControllo: Bottone premuto");

      TeleControlloFrame frame;
        applet.init ();
        System.out.println ("TeleControllo: Creazione di una
finestra per ospitare l'applet");
        frame = new TeleControlloFrame ();
        frame.addWindowListener (frame);
        frame.setSize (400, 300);
        frame.getContentPane().add ("Center", applet);
        frame.setIconImage(iconcina);
        frame.show ();
      }
      catch (Exception ex) {
        System.out.println ("TeleControllo:
TeleControllo.ListaDispositivi() exception: " + ex);
      }
    }
  }


  class aggiornaListaDispositivi implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      try {
        Class[] c = { Dispositivo.class };
        ServiceTemplate template = new ServiceTemplate (null, c,
null);
        ServiceMatches matches = registrar.lookup(template, 50);

        cp.remove(pannelloCentrale);
        pannelloCentrale.removeAll();
        pannelloCentrale = listaDispositivi(matches);
        cp.add(BorderLayout.CENTER, pannelloCentrale );
        bottone.setEnabled(false);
        finestra.validate();

        System.out.println ("TeleControllo: AGGIORNA");
       }
       catch (Exception ex) {
          System.out.println ("TeleControllo:
TeleControllo.aggiornaListaDispositivi() exception: " + ex);
       }
    }
  }

  public TeleControllo( String[] args ) {
    super("TeleControllo della casa");
    cp = getContentPane();
    cp.setLayout(new BorderLayout());
    cp.add(BorderLayout.NORTH, titolo);
    cp.add(BorderLayout.SOUTH, fine);
    aggiorna.addActionListener(new aggiornaListaDispositivi());
    panEast.add(aggiorna);
    bottone.setEnabled(false);
    panEast.add(bottone);
    cp.add(BorderLayout.EAST,panEast);
    ricercaDispositiviJini(args);
  }
```

```
  public JScrollPane listaDispositivi (ServiceMatches lista) {
    int i, j, numDisp;
    Object object;
    Dispositivo disp;
    String testo;
    JPanel p2;
    DefaultMutableTreeNode root, child;

    listaDisp = lista;

    root = new DefaultMutableTreeNode("Lista dispositivi");
    tree = new JTree(root);

    System.out.println ("TeleControllo: Dispositivi trovati " +
lista.totalMatches);

    model =(DefaultTreeModel) tree.getModel();

    for (i = (lista.totalMatches-1); i >= 0; i--) {
      try {
        disp = (Dispositivo) lista.items[i].service;
        child = new DefaultMutableTreeNode( disp.nome() );
        testo = disp.descrizione();
        if ( testo.compareTo("") != 0 ) child.add(new
DefaultMutableTreeNode(disp.descrizione()) );
        testo = disp.locazione();
        if ( testo.compareTo("") != 0 ) child.add(new
DefaultMutableTreeNode(disp.locazione()) );
        model.insertNodeInto(child, root, 0);
      }
      catch (Exception e) { System.out.println ("TeleControllo:
exception: " + e); }
    }
    tree.expandRow(0);
    tree.addTreeSelectionListener( new SelezioneDispositivo() {
    });
    return (new JScrollPane(tree));
  }

  public void ricercaDispositiviJini ( String[] args ) {
    int i, j;
    int iPort;
    String sHost;
    LookupLocator lookup;
    ServiceID serviceID;
    ServiceMatches matches;
    ServiceTemplate template;
    Applet applet;
    Object object;

    try {

/* Impostazione del security manager per permettere a RMI class
   loader di raggiungere il codebase per le classi che non sono
   disponibili localmente
==================================================  */
      System.setSecurityManager (new RMISecurityManager ());

/* Ricerca del Jini Lookup Service (reggie), sul localhost o, se
```

```
   specificato come argomento, ad su un certo IP
   ================================================  */
      System.out.println ();
      if ( args.length > 0 ) {
        System.out.println ("ARGOMENTO : " + args[0]);
        lookup = new LookupLocator ( args[0] );
      }
      else {
        lookup = new LookupLocator ( "jini://localhost" );
      }
      sHost = lookup.getHost ();
      iPort = lookup.getPort ();
      System.out.println ();
      System.out.println ("TeleControllo: LookupLocator      = " +
lookup);
      System.out.println ("TeleControllo: LookupLocator.host = " +
sHost);
      System.out.println ("TeleControllo: LookupLocator.port = " +
iPort);

/* Ricerca del Jini Lookup Service (reggie), sul localhost o, se
   specificato come argomento, ad su un certo IP
   ================================================  */

      registrar  = lookup.getRegistrar ();
      serviceID  = registrar.getServiceID ();
      System.out.println ("TeleControllo: ServiceRegistrar   = " +
registrar);
      System.out.println ("TeleControllo: ServiceID          = " +
serviceID);
    }
    catch (Exception e) {
      System.out.println ("TeleControllo: Lookup non attivo,
exception: " + e);
    }


/* Ricerca dei dispositivo iscritti presso il Jini Lookup Service
(reggie)
  ================================================  */

    try {
      Class[] c = { Dispositivo.class };
      template = new ServiceTemplate (null, c, null);
      matches = registrar.lookup(template, 50);

/* Creazione del pannello centrale contenente i bottoni per accedere
   ai servizi trovati.
   ================================================  */
      pannelloCentrale = listaDispositivi(matches);
      cp.add(BorderLayout.CENTER, pannelloCentrale );
    }
    catch (Exception e) {
      System.out.println ("TeleControllo: TeleControllo.main()
exception: " + e);
    }
  }

  public static void main (String[] args) {
```

```
    finestra = new TeleControllo(args);
    WindowListener l = new WindowAdapter () {
      public void windowClosing (WindowEvent e) {
        System.exit(0);
      }
    };
    finestra.addWindowListener(l);
    finestra.setBounds(100,80,500,300);
    finestra.setVisible(true);
  }
}

/*********************************************************

TeleControlloFrame -- Finestra independente per contenere l'Applet
che rappresenta l'interfaccia utente di controllo del dispositivo

 *********************************************************/

    class TeleControlloFrame extends JFrame implements
WindowListener
    {

      public TeleControlloFrame () {
        super("Controllo remoto");
      }

      public void windowOpened (WindowEvent e)
      {
      }

      public void windowClosing (WindowEvent e)
      {
        System.out.println ("TeleControllo: Chiusura applet");
        //System.exit (0);
      }

      public void windowClosed (WindowEvent e)
      {
      }

      public void windowIconified (WindowEvent e)
      {
      }

      public void windowDeiconified (WindowEvent e)
      {
      }

      public void windowActivated (WindowEvent e)
      {
      }

      public void windowDeactivated (WindowEvent e)
      {
      }
    }
```

C.4 Sistema di gestione allarmi e scenari

C.4.1 Motore inferenziale

```
;     Motore Inferenziale - 2a versione
;     Agenzia Domotica
;     Copyright 2001 Noseda Alberto & Zappa Luca - AirLab Como


; Collegamento alla classe
(bind ?cl (new GestoreAllarmi))


; Strutture per la gestione allarmi
(defglobal ?*temperatura* = 21)

(deftemplate allarme
      (slot tipo)
      (slot stanza)
)

(deftemplate allarme_temp
      (slot a_b)
      (slot stanza)
      (slot temperat)
)

(deftemplate liv_allarme
      (slot tipo)
      (slot livello)
      (slot stanza)
)

(deftemplate dominus
      (slot locazione)
)

(deftemplate stato_dispositivo
      (slot dispositivo)
      (slot stanza)
      (slot stato)
)


; Avvisi
(defrule chiama_dominus_fuori_casa
      ?c <- (avvisa_dominus ?allarme)
      (not (dominus (locazione "casa")))
      =>
      (retract ?c)
      (call ?cl mail ?allarme)
```

```
      (call ?cl sms ?allarme)
)

(defrule chiama_dominus_in_casa
      ?c <- (avvisa_dominus ?allarme)
      (dominus (locazione "casa"))
      =>
      (retract ?c)
      (call ?cl suona)
)

(defrule chiama_pompieri
      ?c <- (avvisa_pompieri ?allarme)
      =>
      (retract ?c)
      (call ?cl telefono "pompieri" ?allarme)
)

(defrule chiama_sicurezza
      ?c <- (avvisa_sicurezza ?allarme)
      =>
      (retract ?c)
      (call ?cl telefono "sicurezza" ?allarme)
)


; Gas
(defrule allarme_gas_1
      ?g <- (allarme (tipo "gas") (stanza ?stanza))
      (not (liv_allarme (tipo gas) (livello ?) (stanza ?stanza)))
      =>
      (retract ?g)
      (assert (liv_allarme (tipo gas) (livello 1) (stanza ?stanza)))
)

(defrule allarme_gas_2
      ?g <- (allarme (tipo "gas") (stanza ?stanza))
      ?g1 <- (liv_allarme (tipo gas) (livello 1) (stanza ?stanza))
      =>
      (retract ?g ?g1)
      (assert (liv_allarme (tipo gas) (livello 2) (stanza ?stanza)))
      (call ?cl chiudi "gas" ?stanza)
      (call ?cl chiudi "corrente" ?stanza)
)

(defrule allarme_gas_3
      ?g <- (allarme (tipo "gas") (stanza ?stanza))
      ?g2 <- (liv_allarme (tipo gas) (livello 2) (stanza ?stanza))
      =>
      (retract ?g ?g2)
      (assert (liv_allarme (tipo gas) (livello 3) (stanza ?stanza)))
      (call ?cl chiudi "gas" "0")
      (assert (avvisa_dominus "gas"))
)

(defrule allarme_gas_4
      ?g <- (allarme (tipo "gas") (stanza ?stanza))
      ?g3 <- (liv_allarme (tipo gas) (livello 3) (stanza ?stanza))
      =>
```

```
      (retract ?g ?g3)
      (assert (liv_allarme (tipo gas) (livello 4) (stanza ?stanza)))
      (assert (avvisa_pompieri "gas"))
      (assert (avvisa_sicurezza "gas"))
      (call ?cl apri_porte ?stanza)
)


; Acqua
(defrule allarme_acqua_1
      ?a <- (allarme (tipo "acqua") (stanza ?stanza))
      (not (liv_allarme (tipo acqua) (livello ?) (stanza ?stanza)))
      =>
      (retract ?a)
      (assert (liv_allarme (tipo acqua) (livello 1) (stanza
?stanza)))
)

(defrule allarme_acqua_2
      ?a <- (allarme (tipo "acqua") (stanza ?stanza))
      ?a1 <- (liv_allarme (tipo acqua) (livello 1) (stanza ?stanza))
      =>
      (retract ?a ?a1)
      (assert (liv_allarme (tipo acqua) (livello 2) (stanza
?stanza)))
      (call ?cl chiudi "acqua" ?stanza)
      (call ?cl chiudi "corrente" ?stanza)
)

(defrule allarme_acqua_3
      ?a <- (allarme (tipo "acqua") (stanza ?stanza))
      ?a2 <- (liv_allarme (tipo acqua) (livello 2) (stanza ?stanza))
      =>
      (retract ?a ?a2)
      (assert (liv_allarme (tipo acqua) (livello 3) (stanza
?stanza)))
      (call ?cl chiudi "acqua" "0")
      (assert (avvisa_dominus "acqua"))
)

(defrule allarme_acqua_4
      ?a <- (allarme (tipo "acqua") (stanza ?stanza))
      ?a3 <- (liv_allarme (tipo acqua) (livello 3) (stanza ?stanza))
      =>
      (retract ?a ?a3)
      (assert (liv_allarme (tipo acqua) (livello 4) (stanza
?stanza)))
      (assert (avvisa_pompieri "acqua"))
)


; Temperatura
(defrule allarme_temperatura_alta
      ?t <- (allarme_temp (a_b "alta") (stanza ?stanza) (temperat
?temp))
      =>
      (retract ?t)
      (call ?cl setta_temp ?stanza (- ?*temperatura* ?temp))
)
```

```
(defrule allarme_temperatura_bassa
      ?t <- (allarme_temp (a_b "bassa") (stanza ?stanza) (temperat
?temp))
      =>
      (retract ?t)
      (call ?cl setta_temp ?stanza (+ ?*temperatura* ?temp))
)


; Intruso
(defrule allarme_intruso
      ?i <- (allarme (tipo "movimento") (stanza ?stanza))
      (not (dominus (locazione "casa")))
      =>
      (retract ?i)
      (call ?cl chiudi_porte ?stanza)
      (call ?cl chiudi "corrente" ?stanza)
      (assert (avvisa_sicurezza "intruso"))
      (assert (avvisa_dominus "intruso"))
)


; Fumo
(defrule allarme_fumo_1a
      ?f <- (allarme (tipo "fumo") (stanza ?stanza))
      (not (allarme_temp (a_b "alta") (stanza ?stanza) (temperat
?temp)))
      (not (liv_allarme (tipo fumo) (livello ?) (stanza ?stanza)))
      =>
      (retract ?f)
      (assert (liv_allarme (tipo fumo) (livello 1) (stanza
?stanza)))
)

(defrule allarme_fumo_1b
      ?f <- (allarme (tipo "fumo") (stanza ?stanza))
      (allarme_temp (a_b "alta") (stanza ?stanza) (temperat ?temp))
      (not (liv_allarme (tipo fumo) (livello ?) (stanza ?stanza)))
      =>
      (retract ?f)
      (assert (liv_allarme (tipo fumo) (livello 2) (stanza
?stanza)))
      (call ?cl chiudi "gas" ?stanza)
      (call ?cl chiudi "corrente" ?stanza)
)

(defrule allarme_fumo_2
      ?f <- (allarme (tipo "fumo") (stanza ?stanza))
      ?f1 <- (liv_allarme (tipo fumo) (livello 1) (stanza ?stanza))
      =>
      (retract ?f ?f1)
      (assert (liv_allarme (tipo fumo) (livello 2) (stanza
?stanza)))
      (call ?cl chiudi "gas" ?stanza)
      (call ?cl chiudi "corrente" ?stanza)
)

(defrule allarme_fumo_3
```

```
      ?f <- (allarme (tipo "fumo") (stanza ?stanza))
      ?f2 <- (liv_allarme (tipo fumo) (livello 2) (stanza ?stanza))
      =>
      (retract ?f ?f2)
      (assert (liv_allarme (tipo fumo) (livello 3) (stanza
?stanza)))
      (call ?cl chiudi "gas" "0")
      (call ?cl apri_porte ?stanza)
      (assert (avvisa_dominus "fumo"))
)

(defrule allarme_fumo_4
      ?f <- (allarme (tipo "fumo") (stanza ?stanza))
      ?f3 <- (liv_allarme (tipo fumo) (livello 3) (stanza ?stanza))
      =>
      (retract ?f ?f3)
      (assert (liv_allarme (tipo fumo) (livello 4) (stanza
?stanza)))
      (assert (avvisa_pompieri "fumo"))
)


; Gestione casa - Scenari
(defrule uscita
      ?d <- (dominus (locazione "casa"))
      ?d1 <- (dominus (locazione "uscito"))
      =>
      (retract ?d ?d1)
      (call ?cl spegni_luci "0")
      (call ?cl abbassa_tapparelle "0")
      (call ?cl chiudi_porte "0")
      (call ?cl setta_temp "0" (- ?*temperatura* 3))
)

(defrule entrata
      ?d <- (dominus (locazione "rientrato"))
      =>
      (retract ?d)
      (assert (dominus (locazione "casa")))
      (call ?cl setta_temp "0" (?*temperatura))
)

(defrule poca_luce1
      ?m <- (allarme (tipo "movimento") (stanza ?stanza))
      ?d <- (dominus (locazione "casa"))
      ?l <- (allarme (tipo "luce_bassa") (stanza ?stanza))
      ?sd <- (stato_dispositivo (dispositivo "tenda") (stanza
?stanza) (stato "chiusa"))
      =>
      (retract ?m ?l)
      (call apri_tenda ?stanza)
)

(defrule poca_luce2
      ?m <- (allarme (tipo "movimento") (stanza ?stanza))
      ?d <- (dominus (locazione "casa"))
      ?l <- (allarme (tipo "luce_bassa") (stanza ?stanza))
      ?sd <- (stato_dispositivo (dispositivo "tenda") (stanza
?stanza) (stato "aperta"))
```

```
      =>
      (retract ?m ?l)
      (call accendi_luci ?stanza)
)
```

C.5 Sistema di programmazione azioni

```
import java.io.*;
import java.util.*;
import java.text.*;
import javax.mail.*;
import javax.mail.event.*;
import javax.mail.internet.*;
import net.jini.core.entry.*;
import net.jini.core.lookup.*;
import net.jini.core.discovery.*;
import net.jini.lookup.*;
import net.jini.lookup.entry.*;
import net.jini.discovery.*;
import java.rmi.*;
import javax.swing.*;
import java.awt.*;
import javax.swing.event.*;
import java.awt.event.*;
import javax.swing.tree.*;
import com.zappanoseda.*;


/**
* ToDo -- Permette di programmare delle operazioni da svolgere in un
determinato istante
* @author Luca Zappa & Alberto Noseda
* @version 1.1
*/

public class ToDo extends JFrame {

  static int periodo = 10; // Intervallo in secondi di scansione
della lista delle azione

  class AzioneDifferita implements Comparable {

    Date data;
    Dispositivo dispositivo;
    int operazione;


    public AzioneDifferita (Date d,
                            Dispositivo disp,
                            int op ) {
      data = d;
      dispositivo = disp;
      operazione = op;
```

```
    }

    public int compareTo(Object o) {
      AzioneDifferita a = (AzioneDifferita) o;
      int comp =  data.compareTo(a.data);
      if ( comp == 0 ) { comp = -1; }
      return comp;
    }
  }


  class Azione implements ActionListener {
    public Dispositivo dispositivo;
    JTextField data;
    JFrame frame;
    Image iconcina;

    ButtonGroup bgroup = new ButtonGroup();
    JRadioButton accendi = new JRadioButton("Accendi");
    JRadioButton spegni = new JRadioButton("Spegni");
    JRadioButton chiudiAcqua = new JRadioButton("Chiudi acqua");
    JRadioButton apriAcqua = new JRadioButton("Apri acqua");
    JRadioButton chiudiGas = new JRadioButton("Chiudi gas");
    JRadioButton apriGas = new JRadioButton("Apri gas");
    JRadioButton chiudiAtt = new JRadioButton("Chiudi");
    JRadioButton apriAtt = new JRadioButton("Apri");


    public void actionPerformed(ActionEvent e) {
      try {
        System.out.println ("ToDo: Bottone premuto");
        System.out.println ("ToDo: Creazione di una finestra");
        frame = new JFrame ("ToDo: Programmazione azione " +
dispositivo.nome());
        frame.setSize (300, 200);
        frame.setIconImage(iconcina);
        Container cpFrame = frame.getContentPane();
        cpFrame.setLayout(new FlowLayout());
        Date dataAttuale = new Date();
        data = new JTextField
(DateFormat.getDateTimeInstance(DateFormat.LONG,DateFormat.MEDIUM).f
ormat(dataAttuale) );
        JPanel panFrame1 = new JPanel();
        panFrame1.add(data);
        cpFrame.add(panFrame1);
        cpFrame.setLayout( new GridLayout(3,1) );

        JPanel panFrame2 = new JPanel();

        if (dispositivo instanceof Apparecchio) {
          bgroup.add(accendi);
          bgroup.add(spegni);
          panFrame2.add(accendi);
          panFrame2.add(spegni);
          panFrame2.setLayout( new GridLayout(2,1) );

        }
        if (dispositivo instanceof Lavaggio) {
          bgroup.add(chiudiAcqua);
```

```
          bgroup.add(apriAcqua);
          panFrame2.add(chiudiAcqua);
          panFrame2.add(apriAcqua);
          panFrame2.setLayout( new GridLayout(4,1) );
        }
        if (dispositivo instanceof Gas) {
          bgroup.add(chiudiGas);
          bgroup.add(apriGas);
          panFrame2.add(chiudiGas);
          panFrame2.add(apriGas);
          panFrame2.setLayout( new GridLayout(4,1) );
        }
        if (dispositivo instanceof Attuatore) {
          bgroup.add(chiudiAtt);
          bgroup.add(apriAtt);
          panFrame2.add(chiudiAtt);
          panFrame2.add(apriAtt);
          panFrame2.setLayout( new GridLayout(2,1) );
        }
        cpFrame.add(panFrame2);

        JButton botOk = new JButton("Ok");
        botOk.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          try {
            int operazione;
            if ( accendi.isSelected() )
              operazione = 1;
            else if ( spegni.isSelected() )
              operazione = 2;
            else if ( apriAcqua.isSelected() )
              operazione = 3;
            else if ( chiudiAcqua.isSelected() )
              operazione = 4;
            else if ( apriGas.isSelected() )
              operazione = 5;
            else if ( chiudiGas.isSelected() )
              operazione = 6;
            else if ( apriAtt.isSelected() )
              operazione = 7;
            else if ( chiudiAtt.isSelected() )
              operazione = 8;
            else
              operazione = 0;

            if (operazione !=0 ) {
              AzioneDifferita ad = new AzioneDifferita(
                DateFormat.getDateTimeInstance(DateFormat.LONG,
DateFormat.MEDIUM).parse(data.getText()),
                dispositivo,
                operazione);
              tabella.add(ad);
              System.out.println ("ToDo: Azione programmata
memorizzata");
              frame.dispose();
              aggiornaNumeroProgrammi();
            }
          } catch (Exception ex) { System.out.println("ToDo [Ok]:
Exception " + ex); }
```

```
        } } );

        JButton botAnnulla = new JButton("Annulla");
        botAnnulla.addActionListener( new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            frame.dispose();
        } } );

        JPanel panFrame3 = new JPanel();
        panFrame3.add(botOk);
        panFrame3.add(botAnnulla);
        cpFrame.add(panFrame3);
        frame.show ();
      }
      catch (Exception ex) {
        System.out.println ("ToDo: ToDo.ListaDispositivi()
exception: " + ex);
      }
    }
  }


  class aggiornaListaDispositivi implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      try {
        ricercaApparecchiAttuatori();

        cp.remove(pannelloCentrale);
        pannelloCentrale.removeAll();
        pannelloCentrale = listaApparecchiAttuatori();
        cp.add(BorderLayout.CENTER, pannelloCentrale);
        finestra.pack();
        finestra.repaint();
        System.out.println ("ToDo: AGGIORNA");
       }
       catch (Exception ex) {
          System.out.println ("ToDo: ToDo.aggiornaListaDispositivi()
exception: " + ex);
       }
    }
  }


  class listaProgrammi implements ActionListener {
    public void actionPerformed(ActionEvent e) {

        System.out.println ("ToDo: Creazione di una finestra");
        JFrame frameProg = new JFrame ("ToDo: Lista programmi");
        frameProg.setSize (250, 150);
        frameProg.setLocation(50,50);
        frameProg.setIconImage(new
ImageIcon("programmi.gif").getImage());
        Container cpFrame = frameProg.getContentPane();

        cpFrame.setLayout(new BorderLayout());
        JTree alberoProg;
        DefaultMutableTreeNode root, child;
        root = new DefaultMutableTreeNode("Lista programmi");
        alberoProg = new JTree(root);
```

```
        DefaultTreeModel model =(DefaultTreeModel)
alberoProg.getModel();
        Iterator it = tabella.iterator();
        AzioneDifferita a;
        while ( it.hasNext() ) {
          try {
            a = (AzioneDifferita) it.next();
            child = new
DefaultMutableTreeNode(DateFormat.getDateTimeInstance(DateFormat.LON
G,DateFormat.MEDIUM).format(a.data));
            child.add(new
DefaultMutableTreeNode(a.dispositivo.nome()) );
            String strOP = "OPERAZIONE: ";
            if ( a.dispositivo instanceof Lavaggio ) {
              switch ( a.operazione ) {
                case 1: strOP += "Accensione";
                  break;
                case 2: strOP += "Spegnimento";
                  break;
                case 3: strOP += "Apertura acqua";
                  break;
                case 4: strOP += "Chiusura acqua";
                  break;
              }
            }
            else if ( a.dispositivo instanceof Gas ) {
              switch ( a.operazione ) {
                case 1: strOP += "Accensione";
                  break;
                case 2: strOP += "Spegnimento";
                  break;
                case 5: strOP += "Apertura gas";
                  break;
                case 6: strOP += "Chiusura gas";
                  break;
              }
            }
            else if ( a.dispositivo instanceof Apparecchio ) {
              switch ( a.operazione ) {
                case 1: strOP += "Accensione";
                  break;
                case 2: strOP += "Spegnimento";
                  break;
              }
            }
            else if ( a.dispositivo instanceof Attuatore ) {
              switch ( a.operazione ) {
                case 7: strOP += "Apertura";
                  break;
                case 8: strOP += "Chiusura";
                  break;
              }
            }
            child.add(new DefaultMutableTreeNode(strOP) );
            model.insertNodeInto(child, root, 0);
          }
          catch (Exception ex) {
```

```
            System.out.println ("ToDo: lista programmi exception: "
+ ex);
          }
        }
        alberoProg.expandRow(0);
        cpFrame.add(BorderLayout.CENTER, new JScrollPane(alberoProg)
);
        frameProg.show ();
    }
  }

  public static SortedSet tabella;ì
  private static ServiceRegistrar registrar;
  private ServiceMatches matches;
  static JFrame finestra;
  JPanel pannelloCentrale;
  JButton aggiorna = new JButton( "Aggiorna" , new
ImageIcon("aggiorna.gif"));
  JButton prog = new JButton( "Programmi" , new
ImageIcon("programmi.gif"));
  static JTextField numProg = new JTextField( "Azioni programmate:
0" );
  JLabel
    titolo = new JLabel( "Programmazione azione" ,
SwingConstants.CENTER ),
    fine = new JLabel( "Zappa & Noseda - AIRLAB COMO (febbrario
2001)" , SwingConstants.CENTER );
  Container cp;


  public ToDo(String[] args) {

    super("ToDo");
    tabella = new TreeSet();
    cp = getContentPane();
    connessioneJini();

    cp.setBackground(Color.white);
    cp.setLayout(new BorderLayout());
    cp.add(BorderLayout.NORTH, titolo);
    cp.add(BorderLayout.SOUTH, fine);
    JPanel p2 = new JPanel( new GridLayout(2,1) );
    p2.setBackground(Color.white);
    JPanel p21 = new JPanel(new BorderLayout());
    p21.setBackground(Color.white);
    aggiorna.addActionListener(new aggiornaListaDispositivi());
    p21.add(BorderLayout.SOUTH, aggiorna);
    p2.add(p21);
    JPanel p31 = new JPanel(new BorderLayout());
    p31.setBackground(Color.white);
    prog.addActionListener(new listaProgrammi());
    p31.add(BorderLayout.NORTH, prog);
    numProg.setEditable(false);

numProg.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    numProg.setFont(new java.awt.Font ("SansSerif", 1, 10));
    numProg.setBorder( new
javax.swing.border.LineBorder(Color.white) );
    numProg.setForeground( Color.red );
```

```
    numProg.setBackground( Color.white);
    p31.add(BorderLayout.SOUTH, numProg);
    p2.add(p31);
    cp.add(BorderLayout.EAST, p2);

    ricercaApparecchiAttuatori();
    pannelloCentrale = listaApparecchiAttuatori();
    cp.add(BorderLayout.CENTER, pannelloCentrale);
  }

  public static void aggiornaNumeroProgrammi() {
    numProg.setText( "Azioni programmate: " + tabella.size() );
  }

  private void connessioneJini () {

/*  Connessione alla comunità jini per ricercare i servizi
disponibili
    ================================================  */

    int iPort;
    String sHost;
    Entry[] aeAttributes;
    LookupLocator lookup;
    ServiceID id;

    try {
/*
 Impostazione del security manager per permettere a RMI class loader
 di raggiungere il codebase per le classi che non sono disponibili
 localmente
=================================================  */
      System.setSecurityManager (new RMISecurityManager ());

/*
  Ricerca del Jini Lookup Service (reggie), sul localhost
=================================================  */

      lookup = new LookupLocator ("jini://localhost");
      sHost = lookup.getHost ();
      iPort = lookup.getPort ();

      System.out.println ("LookupLocator = " + lookup);
      System.out.println ("LookupLocator.host = " + sHost);
      System.out.println ("LookupLocator.port = " + iPort);

/*
  Ricerca del Jini Lookup Service (reggie), sul localhost o, se
  specificato come argomento, ad su un certo IP
 ================================================  */

      registrar = lookup.getRegistrar ();
      id = registrar.getServiceID ();
      System.out.println ("ServiceRegistrar = " + registrar);
      System.out.println ("ServiceID = " + id);

    }
    catch (Exception e) {
      System.out.println ("exception: " + e);
```

```
      System.out.println ("Impossibile trovare il lookup service");
    }
    System.out.println();
  }

  public void ricercaApparecchiAttuatori () {

    ServiceTemplate template;
    Class[] c = { Dispositivo.class };
    template = new ServiceTemplate (null, c, null);
    try {
      matches = registrar.lookup(template,50);
    } catch (Exception e) { System.out.println ("exception: " + e);
}
  }

  public JPanel listaApparecchiAttuatori () {
    int i, j, numDisp;
    Object object;
    Dispositivo disp;
    Azione al;
    String testo;
    JButton bottone;

    JPanel pannello = new JPanel();
    pannello.setBackground(Color.white);
    JLabel et = new JLabel("  Premere uno dei bottoni per
programmare una azione  ", SwingConstants.CENTER );
    et.setForeground(Color.black);
    pannello.add(et);

    System.out.println ("ToDo: ServiceMatches     = " + matches);
    System.out.println ("ToDo: Number of matches  = "
      + matches.totalMatches);
    numDisp=0;
    for (i = 0; i < matches.totalMatches; i++)
    {
      System.out.println ("ToDo: " + i + ":   svc item:   "
        + matches.items[i]);
      System.out.println ("ToDo: " + i + ":   svc ID:     "
        + matches.items[i].serviceID);
      //serviceID = matchesitems[i].serviceID;
      System.out.println ("ToDo: " + i + ":   svc object: "
        + matches.items[i].service);
      try {

        if (matches.items[i].service instanceof Dispositivo) {
          disp= (Dispositivo) matches.items[i].service;

          testo=disp.nome() + " " + disp.locazione() ;
          bottone = new JButton( testo );
          bottone.setToolTipText( disp.descrizione() );  //Helpino

          al = new Azione();
          al.dispositivo = disp;
          numDisp++;

/*   Ciclo di stampa degli attributi dei servizi trovati.
    ==============================================  */
```

```
          for (j = 0; j < matches.items[i].attributeSets.length;
j++)
          {
             object = matches.items[i].attributeSets[j];
             System.out.println ("ToDo: " + i + ":" + j + ": attrib
Set: "
               + object);
             if (object instanceof Icona) {
               System.out.println ("TeleControllo: Caricamento icona
specifica");
               Icona icona = (Icona) object;
               //bottone.setIcon( icona.icon );
               bottone.setIcon( icona.getIcon() );
               al.iconcina = icona.getImage() ;
             }
          }

          bottone.addActionListener(al);
          pannello.add( bottone );
        }
      }
      catch (Exception e) {
          System.out.println ("ToDo: ToDo.matchesDispositivi()
exception: " + e);
      }
    }
    if ( numDisp != 0 ) {
      pannello.setLayout(new GridLayout(numDisp+1,1) );
    }
    else {
      pannello.setLayout( new GridLayout(2,1) );
      JLabel no = new JLabel("Nessun dispositivo trovato!",
SwingConstants.CENTER);
      no.setForeground(Color.red);
      pannello.add( no );
    }
    return pannello;
  }

  private static void controllaAzioni () {

    Date dataAttuale = new Date();
    Iterator it = tabella.iterator();
    AzioneDifferita a;
    System.out.println("Numero di azioni programmate: " +
tabella.size() );
    while ( it.hasNext() ) {
      a = (AzioneDifferita) it.next();
      if ( dataAttuale.compareTo(a.data) >= 0 ) {
        // Azione da attivare
        it.remove();
        try {
          if ( a.dispositivo instanceof Lavaggio ) {
            Lavaggio lav = (Lavaggio) a.dispositivo;
            switch ( a.operazione ) {
              case 1:
                System.out.println("ToDo: Accensione " +
a.dispositivo.nome() );
```

```
                lav.on();
                break;
              case 2:
                System.out.println("ToDo: Spegnimento " +
a.dispositivo.nome() );
                lav.off();
                break;
              case 3:
                System.out.println("ToDo: Apertura acqua " +
a.dispositivo.nome() );
                lav.apri_acqua();
                break;
              case 4:
                System.out.println("ToDo: Chiusura acqua " +
a.dispositivo.nome() );
                lav.chiudi_acqua();
                break;
            }
          }
          else if ( a.dispositivo instanceof Gas ) {
            Gas gas = (Gas) a.dispositivo;
            switch ( a.operazione ) {
              case 1:
                System.out.println("ToDo: Accensione " +
a.dispositivo.nome() );
                gas.on();
                break;
              case 2:
                System.out.println("ToDo: Spegnimento " +
a.dispositivo.nome() );
                gas.off();
                break;
              case 5:
                System.out.println("ToDo: Apertura gas " +
a.dispositivo.nome() );
                gas.apri_gas();
                break;
              case 6:
                System.out.println("ToDo: Chiusura gas " +
a.dispositivo.nome() );
                gas.chiudi_gas();
                break;
            }
          }
          else if ( a.dispositivo instanceof Apparecchio ) {
            Apparecchio app = (Apparecchio) a.dispositivo;
            switch ( a.operazione ) {
              case 1:
                System.out.println("ToDo: Accensione " +
a.dispositivo.nome() );
                app.on();
                break;
              case 2:
                System.out.println("ToDo: Spegnimento " +
a.dispositivo.nome() );
                app.off();
                break;
            }
          }
```

```
          else if ( a.dispositivo instanceof Attuatore ) {
            Attuatore att = (Attuatore) a.dispositivo;
            switch ( a.operazione ) {
              case 7:
                System.out.println("ToDo: Apertura " +
a.dispositivo.nome() );
                att.apri();
                break;
              case 8:
                System.out.println("ToDo: Chiusura " +
a.dispositivo.nome() );
                att.chiudi();
                break;
            }
          }
        }
        catch (Exception e) {
          System.out.println ("ToDo: ToDo.controllaAzioni()
exception: " + e);
        }
      }
      else {
        return;
      }
    }
  }

  public static void main(String[] args) {

    finestra = new ToDo(args);

    WindowListener l = new WindowAdapter () {
      public void windowClosing (WindowEvent e) {
        System.exit(0);
      }
    };
    finestra.addWindowListener(l);
    finestra.pack();
    finestra.setLocation(100,200);
    finestra.setVisible(true);

    for ( ; ; ) {
      try {Thread.sleep (periodo * 1000);} catch (Exception e) {}
      controllaAzioni();
      aggiornaNumeroProgrammi();
    }

  }

}
```