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

Un sistema di comunicazione wireless per l'integrazione di robot di servizio in architetture domotiche

Relatore:
Chiar.mo Prof. STEFANO CASELLI

Correlatore:
Prof. VICTOR CALLAGHAN
Dott. Ing. MONICA REGGIANI

Tesi di laurea di:
ELISA LANDINI

Anno Accademico 2003-2004

Indice

Abstract

The fast increasing in the number of elderly people and the lack of assistance is growing into a social issues. So far, most people with physical or cognitive limitations had to rely either on their relatives or on a professional form of care, which is often costly.

The claim for new solutions has led to a wide range of new technology applications. This project focuses in particular on the integration of wireless communication into robotic assistance techniques. These technologies, by providing both physical and information mobility, have the potential of enhancing elderly and disable people's everyday lives in a health resort. In the reference scenario investigated in this project, every person is given a wireless device in which his own profile has been loaded. This device can share data with a number of distributed embedded devices and robots, in order to keep the profile always up-to-date, and detect and avoid emergency situations as soon as possible. This person centric-model is built around the personal wireless device every person is provided with and is made up of a collection of network aware computers and robots which interact with it.

This scenario has been achieved developing an ad-hoc Bluetooth wireless network that allows devices to communicate and interact to attain non-trivial tasks. Since the Bluetooth-based communication system involves a low power request, it can be considered as being sufficiently pervasive. This is an important requirements for such a distributed system which has always to allow for people and robots mobility.

The implementation has been realized with J2ME and Standard Java Bluetooth API (JABWT), whose convergence provides a number of benefits such as portability and mobility.

Capitolo 1

Introduzione

Nel corso degli ultimi cinquanta anni, i miglioramenti della scienza e della medicina hanno indotto un notevole innalzamento dell'età media. Questo fenomeno positivo, ha portato però con sé anche aspetti a valenza negativa, quali il massivo e progressivo aumento delle persone in età geriatrica rispetto ai giovani. Numerose ricerche hanno messo in evidenza l'acuirsi del problema, aggravato da bassi tassi di natalità. Uno studio del CNR [1] stima infatti che nel 2025 gli ultraottantenni rappresenteranno quasi il 7% della popolazione italiana (tabella 1.1).

	1970	1995	2025
Paesi Industrializzati	1.6	3.0	4.3
Italia	1.8	3.6	6.7

Tabella 1.1: *Popolazione ultraottantenne nei Paesi Industrializzati ed in Italia.*

L'assistenza agli anziani e la cura di malattie degenerative a lungo termine quali l'Alzheimer e il morbo di Parkinson, hanno quindi assunto un ruolo fondamentale nello sviluppo delle politiche socio-economiche mondiali. Se il *peso della cronicità* [2] fino a pochi decenni fa rappresentava uno dei tanti aspetti nel dibattito per la riorganizzazione dei sistemi sanitari, oggi coinvolge invece profondamente le ragioni che fondano l'esistenza dei sistemi stessi.

I costi per le cure di questi pazienti *cronici* sottolineano la gravità del problema. In particolare in uno studio condotto sulla demenza [3] (quindi considerando solo il deterioramento cognitivo) si evidenzia l'enorme peso che la cura di questi pazienti ha sulla spesa pubblica: nel 1996 in America queste patologie hanno gravato in media 270 dollari l'anno su ogni cittadino.

Queste cifre rendono l'idea dell'espansione del fenomeno, che colpisce in egual misura tutti i paesi industrializzati. Le soluzioni fino ad ora adottate, basate principalmente sull'assistenza informale (ovvero quella resa da membri della famiglia, amici o lavoratori volontari, anche chiamati *caregiver*), sembrano non sostenerne

più la portata. Ad esempio, in un approfondito articolo di Bianchetti et al. [3] si mostra che

> "mentre da un lato la crescita della popolazione anziana indica che la domanda di assistenza informale crescerà nel futuro, gli altri modelli socio-economici e demografici indicano che la disponibilità di personale che fornisca assistenza a livello informale si sta abbassando".

Da più parti nasce allora la richiesta di trovare nuovi strumenti per risolvere un problema che, sempre più, affligge sia economicamente che eticamente la nostra società.

> "Vi è oggi un notevole spazio che deve essere riempito da studi e ricerche per *inventare* nuove risposte, tenendo conto degli scenari in profonda modifica" [2].

1.1 Introduzione delle nuove tecnologie in ambito assistenziale

L'introduzione delle nuove tecnologie in ambito medico ha permesso la realizzazione di sistemi di assistenza che, pur non sostituendo la figura professionale dell'infermiere e dell'assistente sanitario, possono affiancarlo nelle attività quotidiane [4] [5].

I deficit di un paziente possono essere sia di tipo fisico che cognitivo. Per questo motivo è necessario tener conto di entrambi gli aspetti per lo sviluppo di una soluzione adeguata ed efficiente che risponda alle esigenze dei pazienti e delle loro famiglie.

La robotica in questo senso viene introdotta per ridare al paziente il controllo della dimensione fisica dell'ambiente in cui vive, aspetto fondamentale anche dal punto di vista psicologico. Nelle patologie a lungo termine, infatti

> "la maggior parte degli individui disabili preferisce rimanere nelle proprie case invece di trasferirsi in una struttura istituzionale per ricevere l'assistenza di cui ha bisogno" [3].

Un robot mobile, eventualmente dotato di un braccio meccanico in grado di afferrare oggetti di media grandezza, può compiere attività che il paziente è invece impossibilitato a portare a termine.

Pensando all'esempio di individui paralizzati, la presenza di questo robot può consentire loro di raggiungere una medicina su una mensola o un oggetto caduto a terra. La mobilità del robot permette allora al paziente di acquisire, o meglio

riacquisire, quel controllo e quelle sicurezze sull'ambiente che il deficit aveva pesantemente ridotto. In un certo senso il robot opera come un'estensione del corpo dell'uomo, dove questo non può più arrivare. L'autonomia concessa al paziente da questo tipo di tecnologia produce inoltre una notevole riduzione del tempo di assistenza richiesto ai *caregiver* o agli assistenti specializzati.

L'altra tecnologia utilizzata in questo ambito è la domotica, che oltre a sopperire ai deficit fisici, può assicurare servizi essenziali nell'assistenza a pazienti con deficit cognitivi.

In primo luogo la domotica consente una vasta automatizzazione dell'ambiente, che va dall'apertura di porte e finestre, all'accensione e al controllo remoto dell'impianto di illuminazione e di riscaldamento di tutta la casa. Applicando gli stessi concetti già visti per la robotica, si capisce l'utilità di questi servizi ricordando che nel caso di pazienti disabili anche le operazioni più semplici possono risultare impossibili.

Inoltre la domotizzazione dell'ambiente consente un continuo ed infallibile controllo delle attività quotidiane. È stato rilevato infatti che

> "una notevole percentuale del tempo a disposizione di una persona che fornisce assistenza viene trascorsa a controllare il paziente nelle sue diverse attività, al fine di poter conoscere e curare i disturbi comportamentali e sorvegliarlo per evitare eventuali situazioni di pericolo, come il contatto con l'acqua bollente, il fuoco, ecc.." [3].

Questo servizio di controllo, assolutamente trasparente al disabile, apporta numerosi vantaggi, tra cui i più importanti sono i seguenti:

- *Sicurezza*: il grado di sicurezza è massimo perché il paziente è sottoposto ad una sorveglianza continua, cosa che un *caregiver*, per quanto attento, non potrà mai fornire.
- *Prevenzione*: se il sistema ravvisa una possibile situazione di pericolo, attua le regole a sua disposizione per evitare che questo effettivamente avvenga.
- *Emergenza*: nel caso in cui l'emergenza non posso comunque essere evitata, un sistema di allarme avverte tempestivamente gli addetti responsabili.
- *Personalizzazione*: tutti i comportamenti possono essere registrati ed opportunamente analizzati, per disegnare un profilo il più personalizzato possibile della malattia del paziente. Così lo spazio e l'ambiente in cui vive possono essere adattati alle sue esigenze.

Per chiarire questi concetti si consideri un anziano con deterioramento cognitivo che durante la notte si alza da letto. In assenza di un sistema di sorveglianza automatizzato, l'anziano può trovarsi in difficoltà e richiedere la presenza di un assistente

che lo guidi, accendendo le luci e aprendo le porte al suo passaggio. In un ambiente *pervasivamente* domotizzato invece, i movimenti del paziente sono continuamente sorvegliati. Un sensore può ad esempio rilevare che il paziente non è più sdraiato, ma si sta alzando da letto. In questo caso viene tempestivamente accesa la luce e la porta scorrevole della camera viene aperta. In questo nuovo scenario, la presenza dell'assistente o del *caregiver* viene richiesta solo in caso di emergenza.

La comunicazione gioca un ruolo fondamentale nella realizzazione di uno scenario in cui domotica e robotica interagiscono per fornire servizi mirati a pazienti disabili. Infatti, alla mobilità del robot e del paziente deve corrispondere un mobilità di comunicazioni. Un sistema wireless a basso consumo di potenza e dimensioni ridotte che può seguire il paziente in tutti i suoi movimenti, affiancato da un elevato numero di *embedded device* distribuiti nello spazio in cui il paziente stesso si muove, può rappresentare la migliore risposta al requisito di mobilità.

Con un tale sistema di comunicazione le informazioni vengono sia condivise a livello distribuito, sia rese rapidamente disponibili al sistema centrale. Questo permette di migliorare e personalizzare l'assistenza assecondando le particolarità di ogni individuo e dello spazio in cui vive.

1.2 Contributi della tesi

Questa tesi si inserisce nel progetto, in corso presso il Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Parma e in collaborazione con il Dipartimento di Computer Science dell'Università dell'Essex, per la realizzazione di un sistema domotico di assistenza ad anziani e disabili. Questo sistema è costituito da un robot mobile (*Nomad*) munito di un braccio meccanico (*Manus*) per afferrare e trasportare oggetti di media grandezza, in grado di comunicare (utilizzando la tecnologia Bluetooth) con vari dispositivi in un ambiente dinamico.

Scopo di questa tesi è la realizzazione di un sistema di comunicazione wireless basato sullo standard Bluetooth, che permetta:

- L'inserimento, la modifica e la memorizzazione dei dati personali su un dispositivo wireless che l'individuo possa portare sempre con sé.
- L'aggiornamento in tempo reale dei dati relativi alle attività quotidiane svolte.
- Lo scambio di informazioni tra dispositivi diversi, per consentire anche ad un ambiente distribuito una visione globale e sempre aggiornata del sistema.
- La verifica tempestiva di eventuali anomalie nelle attività terapeutiche dell'individuo.

L'obiettivo perseguito è quello di realizzare uno strumento che sia il più possibile standard, aperto, portabile.

A tal fine, per quanto riguarda la comunicazione, è stato definito un formato per la descrizione dell'individuo e delle attività ad esso assegnate, basato su XML. Questo formato, oltre a godere delle proprietà di estendibilità, leggibilità e portabilità derivanti dall'uso della tecnologia XML, risulta essere sufficientemente flessibile e completo.

Allo scopo di ottenere una buona portabilità dello strumento di visualizzazione, la tecnologia utilizzata è stata Java. In particolare è stata usata J2ME per sfruttare le API per il Bluetooth (JABWT). Queste API sono un insieme di interfacce con approccio *object-oriented*, e forniscono un modello di programmazione sufficientemente semplice e ad alto livello per realizzare applicazioni Bluetooth con buone prestazioni e *platform-independent*.

1.3 Organizzazione della tesi

Il secondo capitolo introduce nella prima parte i concetti di base della domotica e della robotica; successivamente il capitolo descrive alcuni progetti significativi in ambito europeo e mondiale e presenta un sistema basato sullo standard Bluetooth. L'attenzione è stata soffermata soprattutto sulla descrizione dei tipici problemi riscontrati nello sviluppo di un sistema complesso ed articolato e sulle soluzioni a cui sono giunti gruppi di ricerca diversi.

Prendendo spunto dalle tematiche introdotte nel secondo capitolo, il terzo presenta i requisiti formali a cui questa tesi deve rispondere, ed approfondisce l'argomento descrivendo in dettaglio le metodologie e gli strumenti utilizzati.

Il quarto capitolo è dedicato alla descrizione del sistema realizzato, sia dal punto di vista software che da quello hardware, con particolare attenzione alla sinergia tra dispositivi utilizzati e funzioni implementate.

Il quinto capitolo infine riassume i risultati ottenuti e indica i possibili sviluppi futuri.

Capitolo 2

Tecnologie domotiche e robotiche

2.1 Introduzione

La parola domotica deriva dal latino *domus* e significa letteralmente *automazione della casa*. Negli ultimi due decenni tuttavia, al crescere esponenziale di studi e ricerche in ambito di automazione e controllo di ambienti abitativi, il termine ha assunto un significato molto più ampio, quale la progettazione di sistemi per l'intrattenimento, il risparmio energetico e l'assistenza.

Come illustrato nella figura 2.1, sia il mercato globale di prodotti per l'automazione di edifici (hardware e software), che quello specifico per l'automazione della casa, stanno registrando un notevole incremento [6].

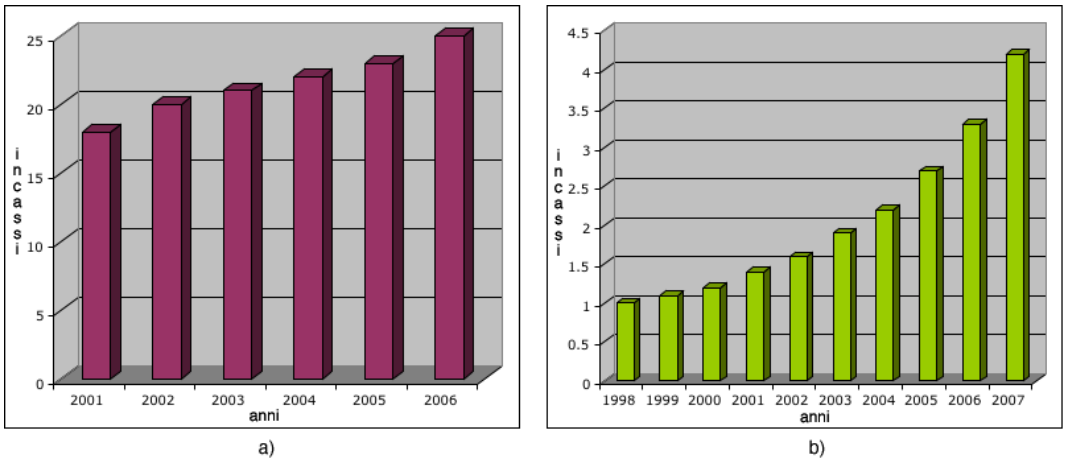

Figura 2.1: *a) Andamento del mercato dei prodotti (hardware e software) per l'automazione di edifici. b) Andamento del mercato specifico per l'automazione della casa (in milioni di $).*

Questo fenomeno ha portato inevitabilmente alla nascita di un numero consistente di gruppi di ricerca, ognuno dei quali ha introdotto un sinonimo di *domotica* per distinguere il proprio progetto e valorizzarne i risultati. Si è arrivati quindi al punto di svalorizzare la parola *domotica* stessa, che per l'eccessivo numero di significati che racchiude, non significare ormai più nulla.

Così, in ambito di automazione e controllo degli ambienti abitativi (dall'ufficio alla macchina fino alle stazioni spaziali) si parla di *disappearing*, *invisible*, *pervasive* o *ubiquitous computing*, *intelligent building*, *intelligent inhabited environment*, *home automation* o *smart home*.

Per avere un'idea più precisa del significato concreto di questi termini, è utile ricordare le parole di Robert Metcalfe, inventore di Ethernet e fondatore di 3Com Corporation, che nel 2001 disse:

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

L'utilizzo di questi dispositivi ha lo scopo di creare un'infrastruttura concettuale e tecnologica che permetta ad ogni individuo (non soltanto al ristretto settore dei tecnici) di assemblare ed utilizzare con semplicità un insieme di prodotti di uso comune provvisti di microprocessore interno e capaci di interagire tramite una rete di comunicazione.

L'acquisizione delle conoscenze necessarie per l'utilizzo di questi dispositivi rappresenta però un barriera molto forte al raggiungimento di questa visione. Si vorrebbe invece creare un sistema in cui l'utilizzatore sia completamente incosciente della presenza delle macchine nell'ambiente che lo circonda. Parafrasando il famoso test di Turing, il test proposto dall'*Intelligent Inhabited Environments Group* dell'Essex [7] per verificare la trasparenza dei computer è:

> "Quale di queste macchine contiene un computer?"

Se non si sa dare risposta a questa domanda, o ancora meglio si risponde

> "Quali macchine?"

significa che i computer possono dirsi *scomparsi*.

Uno dei possibili approcci per affrontare questo problema è quello di inserire degli agenti software (comunemente chiamati *agenti intelligenti*) negli oggetti di uso quotidiano, e trasferire l'apprendimento dalle persone alle macchine. Questo meccanismo, denominato *cognitive disappearance*, libera l'individuo dal dover imparare ad utilizzare il dispositivo. Sarà invece il dispositivo stesso ad imparare ad adattarsi alle esigenze delle persone che ne fanno uso.

Si capisce allora come l'introduzione dell'intelligenza all'interno di questi prodotti debba avvenire in modo trasparente e non-intrusivo per l'individuo.

Emile Aarts (Philips Research) a proposito di questo concetto ha affermato:

> "La casa del futuro sarà più simile a quella del passato che a quella del presente".

Questa previsione mette in evidenza l'evoluzione dei sistemi domotici, nei quali si eviterà l'uso di telecomandi, menu di opzioni e manuali di istruzioni, perché l'ambiente imparerà autonomamente quali sono le nostre preferenze.

Se una visione di questo tipo ha effettivamente assunto un ruolo di primo grado nello sviluppo delle panorama tecnologico mondiale è soprattutto merito delle infrastrutture di comunicazione che permettono di realizzare sistemi di questo tipo.

In particolari i dispositivi possono comunicare ed interagire attraverso una struttura ad-hoc di reti eterogenee [8] [9], e portare così a termine compiti che un solo dispositivo non sarebbe in grado di realizzare. Una rete di questo tipo deve essere in grado di coprire tutto l'ambiente a disposizione, ovvero poter mettere in comunicazione *tutti* i dispositivi che la compongono. Questa visione corrisponde al concetto di *convergenza* indicata da Steve Case (dell'AOL Time Warner) nelle seguenti parole:

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

2.2 Architettura

2.2.1 I limiti dell'IA tradizionale

Le ricerche in ambito domotico hanno prima di tutto dovuto affrontare il problema basilare della definizione di un'architettura in grado di operare in un ambiente reale. Da questo punto di vista, le analogie con la robotica hanno permesso di partire da una base di conoscenze molto vasta.

Il punto di partenza è stato quindi l'esclusione dell'approccio dell'Intelligenza Artificiale (IA) tradizionale, nota per le sue elevate richieste computazionali e di memoria, e troppo fragile per operare in un ambiente real-time. Un altro problema correlato a questo approccio è la mancanza di uno strumento di controllo adeguato, in grado di gestire i quattro principali aspetti che caratterizzano un ambiente domotico reale:

- L'impossibilità di realizzare un modello del mondo sempre aggiornato e coerente.

- L'imprecisione dei sensori.
- Il non-determinismo delle persone e del clima.
- L'enorme numero di variabili in gioco, corrispondenti ad un vettore degli ingressi a più dimensioni.

Questi fattori impediscono l'utilizzo di sistemi di controllo standard tipo fuzzy o PID. Questo limite, aggravato dalle dimensioni compatte che devono avere gli agenti software *embedded* nei dispositivi, richiede l'identificazione di un approccio completamente diverso rispetto a quello proposto dall'IA tradizionale.

2.2.2 Brooks e l'architettura a *Behaviour*

La soluzione che meglio si adatta alle esigenze degli *Intelligent Building* viene dalla robotica, cha già da anni utilizza l'architetture a *Behaviour*, introdotte da Brooks nel 1986 [10] [11] [12].

Constatando l'impossibilità di descrivere il mondo secondo un modello matematico, Brooks afferma che

> "The world is its own best model " [11]
> (Il mondo è il miglior modello di se stesso)

L'utilizzo di un massiccio numero di sensori e feed-back permette quindi una visione sempre aggiornata dell'ambiente in cui il robot (e analogamente il sistema domotico) opera.

Il passo avanti di Brooks consiste poi nella definizione dell'architettura a comportamenti, così chiamata perché ogni agente viene dotato di un certo numero di comportamenti semplici che operano in parallelo. Ai comportamenti di tipo reattivo, dedicati alla reazione immediate a determinati eventi, vengono affiancati anche quelli di tipo deliberativo, più adatti ad una pianificazione a lungo termine. Dalla loro interazione si generano quindi *spontaneamente* comportamenti più complessi.

Questa architettura è fondata su un approccio incrementale: un nuovo *behaviour* viene aggiunto solo quando il precedente è stato completamente testato. Questo approccio *a livelli* permette di facilitare la fase di debug del sistema, che viene concentrata solo sull'ultima parte di codice inserita.

L'agente è costruito su una semplice struttura di base, in cui la complessità viene aggiunta gradualmente. Inoltre, l'interazione e la coordinazione di più agenti [13] permette di portare a termine anche quei compiti che un solo agente non può affrontare.

Queste teorie, traslate in ambito domotico, hanno permesso la definizione di una architettura analoga, in cui i tipici *behaviour* robotici del *wandering* e *wall following* sono stati sostituiti da altri denominati comunemente *goal seeking* o *emergency*.

2.2.3 Analogie tra *Intelligent Building* e robot

Effettivamente non è un caso che le teorie su cui si basano le architetture domotiche siano derivate direttamente dai concetti sviluppati in ambito robotico. Le analogie tra *Intelligent Building* e robot infatti sono molteplici.

Già nel 1921, ad esempio, Le Corbusier affermava che

> "La casa è una macchina in cui vivere"

Come un robot, infatti, essa è composta da un elevato numero di parti meccaniche. Negli edifici domotizzati l'analogia viene ulteriormente rafforzata dalla presenza di sensori e sistemi di comunicazione tipici della robotica.

Anche la domotica, come la robotica, opera in ambienti fortemente dinamici e non modellizzabili, in cui le persone possono avere comportamenti irrazionali ed imprevedibili.

Il gruppo di ricerca dell'Essex [7], ha inoltre formalizzato con il nome *s-map* (ovvero *sensory-map*) un'ulteriore analogia: l'*Intelligent Building* si muove tra configurazioni possibili come il robot *naviga* nello spazio [14].

La figura 2.2 illustra questo concetto.

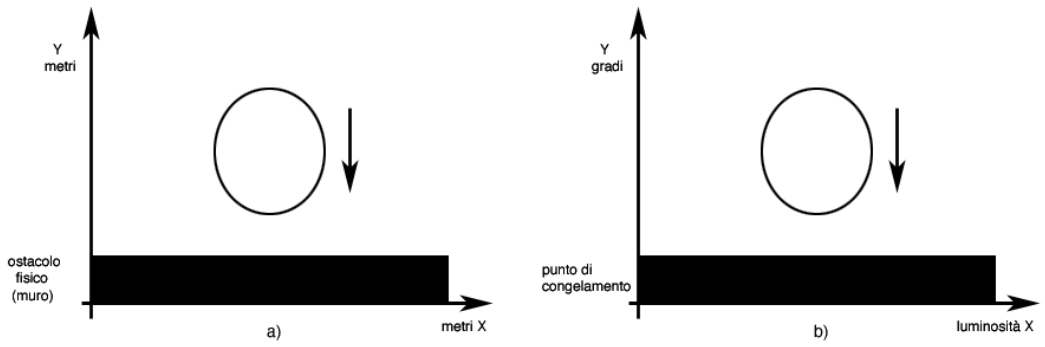

Figura 2.2: *a) Posizione del robot nel piano cartesiano. b) Configurazione dell'Intelligent Building nel piano che mette in relazione temperatura e luminosità*

L'immagine a) rappresenta la posizione di un robot mobile all'interno del piano cartesiano. Il tratto nero indica la presenza di un ostacolo oltre il quale il robot non può andare.

La figura b) rappresenta invece la relazione che il sistema ha appreso tra la luminosità e la temperatura nell'ambiente. In modo analogo alla figura a), anche qui è presente una barriera, identificata dal il punto di congelamento. Questa condizione indica che, fra tutte le possibili configurazioni settabili dal sistema, quelle al di sotto

del tratto nero non sono raggiungibili, o meglio spostano il sistema in un campo di valori *non sicuri*.

L'elevato numero di analogie tra robot ed *Intelligent Building* rende quindi comprensibile l'esistenza di filosofie di pensiero diverse che vedono la robotica come una branca della domotica o viceversa.

Un interessante articolo [15] si inserisce tra queste dissertazioni, introducendo il felice neologismo *immobot*. Questa parola indica tutta quella vasta categoria di sistemi classificabili tra la robotica e la domotica, che va dall'automazione di edifici, ai sistemi installati sulle autovetture per garantire una guida automatica e sicura.

2.3 Problemi aperti

Per realizzare la visione descritta nell'introduzione di questo capitolo, è necessario che il sistema risponda ad alcuni requisiti fondamentali. In questa sezione verranno indicati sia gli aspetti tecnici che quelli metodologici da affrontare nella progettazione di un'architettura domotica [16].

Mentre i primi dipendono sostanzialmente da limiti tecnologici, superabili nell'arco di alcuni anni, i secondi invece rappresentano un ventaglio di domande in continua espansione. Per quel che riguarda questo secondo gruppo, non esiste una soluzione ottimale ai problemi che vengono proposti. Data la complessità di un ambiente domotico e le innumerevoli metodologie a disposizione, è ovvio constatare la presenza di soluzioni diverse proposte da gruppi di ricerca diversi.

2.3.1 Dimensioni e costi

Una delle principali problematiche da affrontare riguarda le dimensioni ed i costi dei dispositivi utilizzati. Per essere trasparenti all'utente, devono infatti essere piccoli e compatti, al limite *wearable*. D'altra parte, essi devono anche avere un costo limitato, per poterli posizionare in modo distribuito nello spazio.

Le caratteristiche fisiche di questi dispositivi implicano inoltre un'autonomia limitata associata a capacità computazionali e di memoria ridotte, che limitano notevolmente la scelta del codice da utilizzare. Ad esempio, non è difficile comprendere le motivazioni che, da questo punto di vista, hanno fatto scartare l'utilizzo dell'intelligenza artificiale tradizionale (notoriamente *computationally demanding*) per lo sviluppo di questi sistemi.

2.3.2 Associazioni e grado di intelligenza

In un sistema distribuito formato da un numero variabile di dispositivi eterogenei, è fondamentale stabilire un meccanismo di associazione [17].

Ogni elemento del sistema infatti opera introducendo nuove funzionalità e scambiando dati sullo stato attuale dell'ambiente e dei suoi abitanti. Le domande a cui è necessario rispondere per progettare un ambiente di questo tipo sono molteplici:

- Cosa accade quando un elemento (ad esempio un cellulare) esce dal sistema?
- La funzione che questo svolgeva e le informazioni che forniva, devono essere rimpiazzate?
- Come riconoscere un nuovo elemento che invece sta entrando?
- La prima comunicazione con il sistema, deve avvenire in modo automatico o tramite l'intervento di un utente esterno?
- Ogni dispositivo comunica con tutti gli altri, o solo con un sottogruppo?
- Come stabilire un sottogruppo che permetta di ottimizzare le ridotte capacità di banda e di I/O dei dispositivi e contemporaneamente permetta una visione abbastanza ampia del sistema?

Un altro importante aspetto è quello dell'intelligenza. Il coesistere di gadget *dumb* (ovvero privi di intelligenza) a fianco di quelli *fully intelligent* (ovvero provvisti di un meccanismo di apprendimento avanzato) mette in luce le difficoltà di interazione che possono sorgere.

2.3.3 Ambiente distribuito

In un'architettura domotica, i dispositivi sono in grado di formare gruppi di lavoro ad-hoc per portare a termine compiti di una certa complessità.

Questo scenario distribuito genera un certo numero di problematiche da affrontare, fra cui:

- È meglio assegnare ad ogni agente software il controllo di un ambiente ben definito (ad esempio uno stanza o un piano) in cui possono essere presenti più individui, o piuttosto assegnare un dispositivo ad ogni individuo, e lasciare che lo segua in tutti i suoi movimenti?
- Come suddividere lo spazio? Gli agenti devono controllare ambienti ben distinti o possono sovrapporsi?
- Qual è la granularità computazionale? Ovvero, i dispositivi sono indipendenti computazionalmente e funzionalmente?
- Le associazioni vengono memorizzate localmente o sono gestite da un sistema centrale?

- C'è un computer che opera da master e controlla il gruppo, o ogni elemento ha lo stesso grado di controllo?
- Qual è il miglior linguaggio di comunicazione, che sia allo stesso tempo compatto ma completo?

2.3.4 Mobilità

Uno dei requisiti basilari per la progettazione di un sistema domotico è la mobilità. Alcuni dispositivi devono infatti poter seguire l'utente in tutte le sue attività quotidiane, instaurando di volta in volta associazioni e comunicazioni nei nuovi sistemi in cui si trovano.

La mobilità può essere realizzata a diversi livelli. Un telefono cellulare ad esempio è estremamente mobile, perché può seguire l'utente in ogni suo movimento. All'estremo opposto, invece, si trovano quegli apparecchi che per loro natura non vengono mai spostati (ad esempio il frigorifero o la televisione).

Ovviamente l'infrastruttura tecnica deve essere in grado di gestire il dinamismo degli elementi, che possono far parte del sistema anche solo per brevi periodi di tempo.

2.3.5 Dimensionalità, temporalità e non-determinismo

Un altro importante aspetto da considerare nell'analisi di un sistema domotico è il numero delle variabili in gioco. I meccanismi di apprendimento, basati sull'osservazione di tutti i fenomeni che avvengono all'interna sistema, devono tener conto dell'elevato numero di configurazioni possibili. Come già accade in ambito robotico, la necessità di aggiornare il più possibile la visione del mondo, rischia di far esplodere l'insieme degli stati.

Un altro problema da affrontare è quello della temporalità. In alcune circostanze infatti il comportamento dell'utente non è determinato dallo stato corrente del mondo , ma piuttosto da una sequenza di eventi che hanno poi portato a quello più recente. In un caso emblematico identificato dal gruppo di ricerca dell'Essex [7] come *s-paradox* [14], si realizza la seguente sequenza di eventi:

- L'utente modifica l'ambiente (ad esempio accendendo la luce);
- Il sistema allora applica una delle regole acquisite nella fase di apprendimento e modifica a sua volta l'ambiente (abbassando le tapparelle);
- L'utente reagisce a sua volta all'ultimo cambiamento verificatosi (accendendo un'altra luce, perché la prima in questa nuova condizione non è più sufficiente).

In uno scenario di questo tipo, in cui è presente una forte interazione tra sistema ed utente, è necessario progettare un'architettura che sia in grado di gestire situazioni di non-determinismo. In altre parole è necessario abbandonare il concetto classico di controllo, basato su un modello matematico (e quindi deterministico) del mondo.

2.4 Studio di un caso: l'*iDorm*

L'*Intelligent Inhabited Environments Group* dell'Università dell'Essex [7] ha realizzato un prototipo di ambiente *pervasivamente* domotizzato in cui testare le proprie ricerche. Come si vede nella figura 2.5, l'*iDorm* [9] [18] [19] [20] a prima vista sembra soltanto una delle tante camere del Campus dell'Università dell'Essex [21].

Figura 2.3: *iDorm*

In realtà si rivela essere una stanza multi-utente e multi-funzione, dotata di un elevato numero di sensori (localizzazione degli occupanti, umidità, luminosità, temperatura, etc) ed attuatori (dimmer delle luci, chiusura motorizzata di porte e finestre, regolatori della temperatura e della ventilazione) ed in grado di apprendere le preferenze e le necessità dei suoi abitanti.

L'apprendimento degli agenti software che gestiscono la stanza di basa su due elementi fondamentali:

- Il campionamento dei sensori ad ogni cambiamento della configurazione attuale (modificata dall'utente);

- La memorizzazione delle configurazioni più significative, identificate dalla frequenza con cui vengono ripetute.

Quando il sensore registra un cambiamento manuale, l'agente effettua lo *snapshot* dei sensori e quindi invia il comando all'attuatore, che infine applica il comando ricevuto. Questo meccanismo, denominato *Evidential Learning*, è determinato dall'attenta osservazione delle coppie stimolo-azione. In altre parole, l'individuo viene continuamente monitorato per registrare gli stimoli che portano ad un cambiamento dell'ambiente. Quando questo avviene (ed è ripetuto con una certa frequenza), la coppia stimolo-azione viene codificata in una nuova regola ed incorporata come parte dei comportamenti dell'agente.

L'aspetto della frequenza è di particolare importanza. Infatti, le ridotte dimensioni dei dispositivi *embedded* su cui devono essere caricati gli agenti, rendono accettabile solo un approccio implementativo minimalista.

Si inserisce in quest'ottica l'utilizzo di soglie e contatori per evitare la memorizzazione di tutte le possibili configurazioni realizzate dagli occupanti: solo quelle ripetute più volte andranno a formare le nuove regole di comportamento del sistema.

Uno degli agenti che gestiscono l'*iDorm* è installato su un dispositivo chiamato SNAP (*Simple Network Application Platform*) [22] mostrato nella figura 2.4. Questo dispositivo permette l'esecuzione di codice scritto in Java ed opera da server (via TCP/IP) per visualizzare con un browser (potenzialmente in ogni parte del mondo) i dati dei sensori ad esso collegati.

Figura 2.4: *Ridotte dimensioni della SNAP.*

Data la massiccia presenza di sensori e dispositivi eterogenei, all'interno dell'*iDorm* sono stati utilizzati diversi protocolli di comunicazione:

- IEEE 802.11b

- Bluetooth
- 1-Wire (proprietario della Dallas Semiconductors)
- IPv6
- LonTalk (della LonWorks)

Inoltre sono stati definiti due schemi di comunicazione:

- Uno schema ad alto livello, basato sullo scambio di messaggi in formato XML e HTML con il server centrale;
- Un secondo sistema di messaggi ACL (Agent Communication Language) a più basso livello basato sull'architettura DIBAL [23].

Per gestire i diversi protocolli e schemi di comunicazione e rendere la struttura scalabile, è stato introdotto un *gateway*, che permette di avere un'unica interfaccia standard tra tutti i dispositivi e gli ambienti collegati alle diverse reti presenti.

Gli abitanti, possono comunicare con il sistema utilizzando l'interfaccia standard rappresentata da interruttori per la luce, il riscaldamento, la chiusura di porte e finestre, oppure con interfacce di altro tipo, quali quella vocale, il WEB, il WAP o quella virtuale VRML.

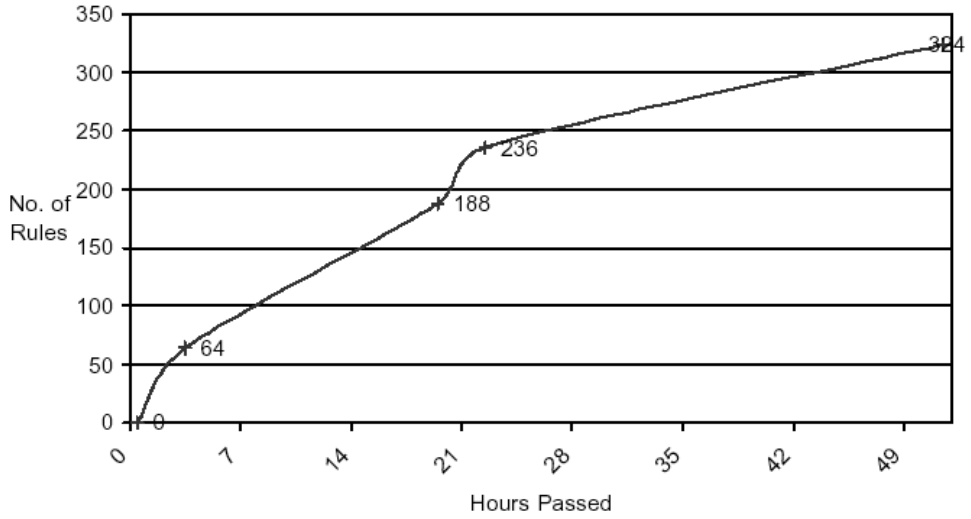

Figura 2.5: *Numero di regole apprese dall'iDorm in funzione delle ore di esperimento.*

Gli esperimenti effettuati sull'*iDorm* hanno messo in evidenza il suo elevato grado di apprendimento. Infatti, in un test di 51 ore, rappresentato nella figura

2.5 partendo da un set di regole inizialmente vuoto, l'agente ha imparato 324 regole. Questo numero, confrontato con le 20736 possibili, evidenzia la capacità di ottimizzazione e di selezione delle solo regole utili ed efficaci.

2.5 Altri progetti a livello internazionale

Negli ultimi anni un numero sempre crescente di progetti domotici si sono affacciati sul panorama mondiale.

In Svezia, Davidsson [24] utilizza tecniche multi-agente per fornire servizi all'interno di edifici. Questi agenti sono sviluppati in conformità a teorie di intelligenza artificiale tradizionale che decompongono il sistema in funzioni piuttosto che in *behaviour*. Questa ricerca non sviluppa alcun meccanismo di apprendimento.

In Colorado, Mozer [25] utilizza invece le reti neurali. Focalizzando l'attenzione unicamente sul controllo intelligente dell'illuminazione della casa, il gruppo ha realizzato un sistema in grado di operare in un ambiente reale. Le prove condotte in questo scenario hanno permesso di constatare una notevole riduzione dei consumi di energia, anche se in alcuni casi sono stati ottenuti a scapito del comfort degli occupanti.

Il MIT ha sviluppato vari progetti in questo contesto. I più significativi da citare sono HAL, HIVE e Internet 0. Il primo si concentra sulla realizzazione di un ambiente in grado di rispondere alle richieste dei suoi occupanti. In questo caso le stanze sono dotate di un elevato numero di sensori e interfacce *intelligenti* in grado di percepire ed interpretare i gesti degli abitanti.

HIVE [26] invece è un esempio particolarmente interessante di un modello distribuito realizzato su un'architettura multi-agente. Questo lavoro differisce da quello sviluppato dall'Intelligent Inhabited Environments Group dell'Essex [7] per quel che riguarda gli agenti: mentre in HIVE sono *soft* (ovvero esclusivamente software), quelli dell'*iDorm* sono strettamente correlati all'hardware del sistema e inseriti al suo interno (ovvero *embedded*). Dato che gli agenti software in HIVE risiedono su server non hanno da tenere in considerazione limiti di memoria e di capacità di calcolo a cui vanno invece incontro quelle dell'*iDorm*.

Lo scopo di Internet0 [27] è invece quello di sviluppare uno standard di comunicazione semplice ed *open-source* che si rifà ai concetti base di Internet, sfruttandone i punti di forza e riducendone al minimo la complessità. Internet 0 dovrebbe permettere di assegnare un indirizzo IP ad ogni apparecchio nella casa (dalle luci all'impianto di riscaldamento) e consentire ad ogni dispositivo di comunicare con tutti gli altri in modo semplice ed automatico. Secondo Gershenfeld, direttore del nuovo centro *Bits and Atoms* del MIT, è giunto il tempo di "lasciare che l'edificio sia il computer, piuttosto che inserire computer al suo interno". La semplicità, correlata ai bassi costi di produzione e di assemblaggio, rappresenta sicuramente uno

dei punti a favore di un sistema di questo tipo. Alcuni, tuttavia, stanno già muovendo delle pesanti critiche, proprio a partire dalla semplicità di questo modello. "Un sistema di questo tipo è in grado di scalare sulle grandi dimensioni? O un aumento delle regole memorizzate sul chip lo porterà dai 3$ attuali ad arrivare fino a 10$ o 20$?" si chiede Ken Sinclain, editore del sito `automatedbuildings.com`.

Per quel che riguarda i progetti europei, non si può dimenticare *The Disappearing Computer* [28], un grande progetto finanziato dall'Unione Europea a partire dal 1° gennaio 2001. Comprende 16 sotto-progetti correlati in varia misura, ed un elevato numero di attività di supporto. Tra questi c'è anche eGadget [29], una collaborazione tra numerose Università Europee tra cui anche quella dell'Essex.

Un'altra iniziativa da citare è *e-Health* [30], che promuove fondi per l'assistenza a distanza. Risulta particolarmente interessante perché riguarda il Comune di Parma ed ha molti aspetti in comune con il progetto realizzato in questa tesi. Infatti, l'Ufficio Progetto Europa del Comune di Parma, insieme all'Assessorato ai Servizi Sociali, ha presentato il 26 aprile scorso una domanda per ottenere contributi per iniziative di e-Health. L'idea del progetto è quella di introdurre nuove tecnologie di assistenza a distanza tramite l'utilizzo di un braccialetto elettronico che rileva i parametri vitali. La sperimentazione verrà effettuata su un campione di anziani in Italia (Parma), Inghilterra (Essex) e Finlandia (Helsinki).

Esistono infine molti altri progetti indirizzati soprattutto all'automazione ed al controllo remoto di edifici. Uno di questi è *Integer House* [31], un progetto inglese in collaborazione con grandi aziende del settore (come Echelon e Cisco Systems) per la realizzazione di case automatizzate e completamente controllabili da remoto. Un progetto analogo è *The Internet Home* [32] realizzato con la partnership di Cisco Systems e della compagnia di costruzione Laing Homes. Una di queste case, in stile tipicamente inglese, si trova a Watford, nei pressi di Londra, ed è aperta al pubblico dal febbraio 2000.

2.6 Domotica e Bluetooth: Child Spotter

Il parco divertimenti Tivoli Garden di Copenaghen, frequentato da molte famiglie con bambini, ha inaugurato la scorsa stagione primaverile presentando il nuovo servizio *Child Spotter*, tramite il quale i genitori potranno sempre sapere in che zona del parco si trovano i figli. Al prezzo di 2 euro infatti, potranno noleggiare un braccialetto con trasmettitore bluetooth da fare indossare ai bambini. Il dispositivo dialogherà poi con una serie di access point disseminati nel parco. Il meccanismo è semplice: inviando ad un certo numero di telefono un SMS con il codice associato al braccialetto, si riceverà in pochi secondi la posizione del braccialetto, ovvero del bambino. Il servizio è erogato dalla danese Bluetags [33], che oltre al dispositivo da polso appena descritto, ne realizza anche uno *wearable*, ovvero indossabile. Il

piccolo apparato (8 x 35 x 38 mm per 11g di peso, batteria compresa) ha un'antenna interna, è conforme allo standard Bluetooth 1.2 ed ha una portata massima di 50 metri.

Capitolo 3

Requisiti e strumenti

3.1 Scenario generale

Il sistema di cui si è realizzato un prototipo è inteso per operare in una struttura assistenziale, quale può essere un ospedale geriatrico a lunga degenza. In questo scenario, ogni paziente viene dotato di un dispositivo (che nel migliore dei casi è un braccialetto o una spilla) in grado di effettuare comunicazioni wireless.

Quando arriva un nuovo paziente, tutti i suoi dati vengono inseriti nel database centrale, comprese le attività terapeutiche che gli sono state assegnate. Da qui vengono poi salvati sul dispositivo personale che il paziente terrà sempre con sé.

Al completamente di un'attività, viene memorizzato sul dispositivo il nuovo stato del profilo del paziente. Questa operazione viene eseguita da un operatore oppure da dispositivi *embedded* associati ad ambienti specifici. Ad esempio possono trovarsi all'ingresso della piscina, dell'ambulatorio o della palestra. Il dispositivo ha anche la funzione di identificare e registrare un'eventuale attività non effettuata. In questo modo si può mantenere uno stato sempre aggiornato delle attività quotidiane del paziente.

Si ipotizza che nella struttura assistenziale sia presenta anche un robot mobile, anch'esso in grado di comunicare senza fili, con la funzione di monitorare le attività terapeutiche dei pazienti ed eventualmente ricordarne la necessità tramite la sintesi vocale. Quando il robot si avvicina ad un paziente, esso riceve dal dispositivo lo stato aggiornato del suo profilo. Questi dati possono poi essere condivisi con altri robot mobili, per permettere ad ognuno di questi di avere una visione il più possibile globale ed attuale dei profili dei pazienti ricoverati, oppure possono essere inviati al sistema centrale per la gestione di eventuali emergenze.

3.2 Requisiti formali

3.2.1 Mobilità ed ambiente distribuito

Data la natura distribuita dell'ambiente in cui si deve operare, la mobilità è un requisito basilare da realizzare. Il sistema deve permettere il maggior grado di mobilità possibile, corrispondente ad una copertura adeguata di tutto lo spazio in cui i pazienti possono muoversi.

Il concetto di mobilità in questo contesto è espandibili in due diverse direzioni, tra loro strettamente correlate. La prima è una mobilità *fisica*, intesa come possibilità di seguire i pazienti in tutti i loro spostamenti. Questi possono infatti muoversi sia all'interno dell'ospedale che all'esterno. La mobilità fisica è ottenuta da una parte con il dispositivo di cui sono dotati in ogni momento tutti i pazienti, in grado di seguirli in ogni loro movimento, e dall'altra con il robot mobile, capace di assisterli in ogni zona.

Il secondo tipo di mobilità si riferisce invece alla *comunicazione*. In questo caso l'attenzione è posta sulla necessità di realizzare comunicazioni wireless tra i vari dispositivi. Infatti questi apparecchi devono seguire sempre il paziente e allo stesso tempo poter comunicare con il robot mobile e i computer *embedded* distribuiti. Una condizione di questo tipo è realizzabile solo con un sistema di comunicazione wireless, in cui lo scambio di informazione avviene senza alcun intervento esterno, anche quando il paziente non è in compagnia di un operatore.

3.2.2 Trasparenza

Il requisito di trasparenza è fondamentale nella progettazione di un'architettura domotica. Nella realizzazione di un sistema di assistenza inoltre assume una valenza ancora più significativa.

I pazienti infatti non possono in alcun modo essere gravati del peso della tecnologia. Per questo motivo è obbligatorio realizzare un sistema il meno intrusivo possibile, il cui le interazioni paziente-macchina siano pressoché nulle. L'apprendimento, secondo il concetto di *cognitive disappearing* già visto nel Capitolo 2, deve essere spostato dall'utente al dispositivo: il paziente non deve *saper usare* il sistema, ma piuttosto il sistema stesso deve adattarsi alle esigenze del paziente (come ad esempio l'esigenza di mobilità, vista nella sezione precedente). Inoltre, la mancanza di trasparenza, ovvero l'intrusione eccessiva del sistema nella vita del paziente, potrebbe condizionare i suoi comportamenti, e falsare l'analisi dei dati ricavati dall'osservazione dei comportamenti stessi.

3.2.3 Ridotte dimensioni

Per ridurre il più possibile l'impatto del dispositivo sul paziente, è necessario ridurne al minimo le dimensioni (al limite renderlo *wearable*). Più il dispositivo è piccolo, meno il paziente avrà coscienza della sua esistenza. In questo senso quindi, la compattezza del dispositivo è strettamente correlata alla trasparenza del sistema.

3.2.4 Interoperabilità

La presenza di dispositivi eterogenei che devono scambiarsi informazioni implica l'adozione di un linguaggio comune, in grado di essere eseguito su tutte le macchine senza creare incompatibilità e senza la necessità di riscrivere un codice diverso su ogni piattaforma.

Inoltre la mancanza di un hardware ben definito da utilizzare come dispositivo da assegnare al paziente ha richiesto l'implementazione di un software *platform-independent* che fosse anche svincolato dall'hardware di base del dispositivo.

3.3 La piattaforma J2ME

Il linguaggio utilizzato per l'implementazione del sistema è stato Java. In particolare è stata scelta la piattaforma J2ME, dedicata a prodotti di consumo quali telefoni cellulari, palmari e una vasta gamma di dispositivi *embedded*. J2ME è formata da un insieme di API standard definite da un gruppo di esperti del settore seguendo il programma *Java Community Process*, e beneficia della tecnologia Java perfezionata per dispositivi a ridotta capacità computazionale, di memoria e di I/O.

L'architettura di J2ME, rappresentata nella figura 3.1 definisce le configurazioni, i profili ed i pacchetti opzionali la cui combinazione permette di ottimizzare le caratteristiche tipiche del dispositivo su cui si vuole operare.

Le configurazioni definiscono il minimo set di librerie e le funzionalità della *virtual machine* supportata per un particolare insieme di dispositivi che condividono caratteristiche simili, come ad esempio la connettività alla rete e la memoria disponibile. Esistono due tipi di configurazioni: CDC (*Connected Device Configuration*) e CLDC (*Connected Limited Device Configuration*).

- CDC: per dispositivi con un minimo di 2MB di memoria disponibile e CPU a 32-bit. Questa configurazione include una *virtual machine* completa (JVM) e un set abbastanza ampio di API (derivante nella maggior parte da quelle della Java 2 Standard Edition).

- CLDC: è la configurazione più piccola, destinata a dispositivi con 128-512 KB di memoria, CPU a 16 o 32 bit e connessione *intermittente* alla rete.

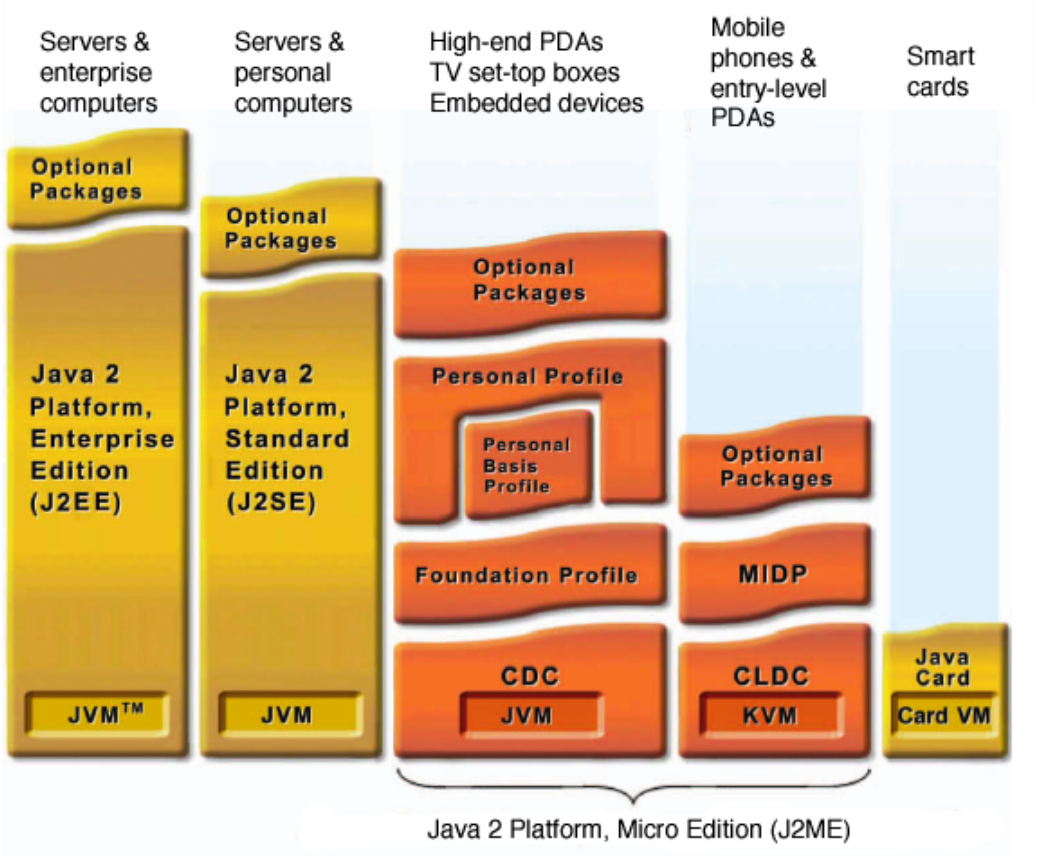

Figura 3.1: *Posizione e caratteristiche della piattaforma J2ME*

> Include una *virtual machine* compatta e veloce, ed un ristretto ma funzionale set di API. Si adatta particolarmente bene al wireless, ma è anche utilizzata dalla maggior parte di dispositivi *embedded* a ridotte dimensioni.

Per fornire un ambiente di sviluppo completo e indirizzato ad una vasta categoria di dispositivi, le configurazioni devono essere combinate con i profili, ovvero con un set di API a più alto livello che definiscono ulteriori caratteristiche quali l'accesso a proprietà specifiche dei singoli dispositivi.

Il profilo di cui è stato dotato il CLDC è il MIDP (*Mobile Information Device Profile*), che offre funzionalità di base richieste dai dispositivi mobili. Presenta infatti API per gestire l'interfaccia con l'utente, la connettività alla rete, la memorizzazione permanente dei dati e la gestione dell'applicazione. Combinato con il CLDC, il MIDP fornisce un ambiente di sviluppo completo che beneficia delle capacità di dispositivi mobili, minimizzando il consumo di memoria e di energia. Questo ambiente fornisce infatti tre pacchetti `java`, ovvero `java.io`, `java.lang` e

`java.util` che ereditano le funzionalità di J2SE adattate alla *Micro Edition*, e il pacchetto `javax` contenente le classi della `microedition`. In particolare, per la `microedition` sono presenti i seguenti quattro pacchetti:

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

Utilizzando il profilo MIDP è possibile sviluppare programmi chiamati *MIDlet*. MID (*Mobile Information Device*) identifica la categoria di dispositivi su cui questi programmi devono essere eseguiti. Il suffisso *let* invece ricorda gli *applet*: semplici applicazioni Java con una struttura bene definita. I *midlet* hanno effettivamente ereditato dagli *applet* molte delle loro caratteristiche: dalla struttura di base, alla semplicità implementativa.

La J2ME può essere ulteriormente estesa da un vasto numero di pacchetti opzionali, da combinare alle configurazioni utilizzate ed ai loro rispettivi profili. Creati per rispondere a specifiche richieste di mercato, questi pacchetti offrono API standard sia per tecnologie già esistenti, che per quelle emergenti. Uno dei pacchetti opzionali più importante è quello per il Bluetooth.

3.4 Bluetooth

Come standard di comunicazione tra i dispositivi eterogenei è stato scelto lo standard Bluetooth. Questa tecnologia è stata ideata nel 1994 dalla ditta svedese Ericsson, come metodo di comunicazione a corto raggio (circa 10 metri) tra dispositivi mobili per rispondere ad una precisa esigenza di mercato. Infatti, con l'aumento dei dispositivi elettronici di uso quotidiano, sono aumentati anche i metodi di connessione, sempre più specifici e proprietari.

Ad esempio, un cavo VGA connette il computer al monitor ed uno USB alla stampante. Una connessione dati veloce richiede Ethernet (ovvero un cavo RJ-45) oppure, accontentandosi di una velocità ridotta, un doppino telefonico con presa RJ-11. A questi vanno poi aggiunte tutte le connessioni a breve termine, come quelle che si realizzano temporaneamente per sincronizzare i dati tra computer e palmare, o per scaricare le foto dalla macchina digitale, quelle via radio che permettono l'uso di

telefoni senza fili, e infine quelle ad infrarossi per il controllo remoto del televisore o dello stereo.

Questa breve panoramica degli standard utilizzati quotidianamente mette in evidenza la necessità di un denominatore comune, per sollevare l'utente dai limiti introdotti dall'hardware proprietario.

Nel 1998 questi concetti si concretizzano nel consorzio SIG *(Special Interest Group)*, formato da Ericsson, IBM, Intel, Nokia e Toshiba per sviluppare specifiche *open-source* e indipendenti dalle grandi case costruttrici. Da allora, più di 2000 società hanno partecipato al SIG Bluetooth, compresi virtualmente tutti i produttori di telefonia, computer e palmari.

Anche se inizialmente è stato utilizzato quasi esclusivamente come sostitutivo delle comunicazioni via cavo e via raggi infrarosso, il Bluetooth offre effettivamente molti altri servizi. Questi vanno dalla sincronizzazione di dispositivi al controllo sull'accesso di edifici basato sul riconoscimento degli utenti provvisti di tecnologia Bluetooth. Le opportunità di sviluppo create da questo standard sono poi rese particolarmente interessanti da tre elementi:

- Il costo ridotto necessario per la realizzazione di dispositivi con Bluetooth integrato.
- La possibilità di riassumere in un unico *telecomando* tutti gli eterogenei dispositivi che ognuno di noi porta sempre con sè, come carte magnetiche e di credito, chiavi di casa, della macchina, dell'ufficio, telecomandi per il cancello e il garage.
- La facilità di installazione e di utilizzo, resa possibile dal riconoscimento automatico di altri dispositivi Bluetooth adiacenti. Questa trasparenza permette all'utente di aprire cancelli e porte senza azionare nessun comando, oppure telefonare con il semplice uso di auricolari, pur tenendo il cellulare in tasca.

Si comprende quindi come questa tecnologia (che al pari di altri standard usa la banda Industrial-Scientific-Medical (ISM) dei 2.4 GHz), sia molto più che un metodo di sostituzione di cavi e rappresenti piuttosto un metodo di comunicazione completamente nuovo tra dispositivi elettronici a breve distanza.

3.4.1 Confronto con altri standard wireless

Per mettere in evidenza i punti di forza ed i limiti di questa tecnologia, risulta utile un breve confronto con due noti standard di comunicazione wireless i cui ambiti vengono spesso ad intersecarsi con quelli del Bluetooth: l'infrarosso e l'802.11b.

Generalmente i dispositivi domestici come i TV o i videoregistratori (anche indicati come *black devices*, in contrasto con gli elettrodomestici come la lavatrice

e la lavastoviglie chiamati *white devices*) vengono controllati con telecomandi a fasci di luce infrarossa.

Questo tipo di comunicazione, pur beneficiando di costi ridotti ed estrema facilità di utilizzo, è però limitata per quel che riguarda il raggio di azione. Inoltre, dato che ricevitore e trasmettitore devono vedersi reciprocamente, la comunicazione può avvenire solo posizionandosi di fronte all'apparecchio a cui si vuole inviare il segnale.

Per quel che riguarda il confronto con lo standard 802.11b, l'aspetto comune più evidente riguarda la banda: entrambi questi protocolli di comunicazione wireless operano nella banda dei 2.4 GHz. Tuttavia è importante non considerare Bluetooth come un sostitutivo dello standard 802.11b nell'ambito, ad esempio, di LAN realizzate senza cavi, perché effettivamente sono stati pensati e realizzati come tecnologie complementari.

Il protocollo 802.11b è stato progettato per connettere dispositivi di dimensioni piuttosto grandi e con elevata disponibilità di potenza e di banda, come computer desktop e portatili. Questi possono comunicare ad una velocità massima di 11 Mbit/sec e raggiungere distanze di 100 metri.

Dall'altra parte Bluetooth è stato realizzato per dispositivi di dimensioni ridotte come palmari e telefoni cellulari, in grado di comunicare ad una velocità massima di 1 Mbit/sec, in un raggio di 10 metri. È quindi ovvio che questo protocollo può operare a potenza molto ridotta.

I miglioramenti delle versioni 1.1 e 2.0 dello standard Bluetooth hanno consentito l'introduzione di nuove classi di potenza, che permettono di raggiungere fino a 100 metri di raggio mantenendo un consumo abbastanza limitato.

Un'altra importante differenza con lo standard 802.11b è relativa al fatto che questo non è stato creato per le comunicazione vocali, mentre ogni connessione Bluetooth supporta sia i dati che la voce.

Riassumento quel che è emerso in questa sezione, è possibile elencare i maggiori vantaggi della tecnologia Bluetooth.

- *È wireless ed opera in modo automatico*: non servono cavi, connettori e connessioni. Inoltre la connessione viene iniziata automaticamente, perché i dispositivi, posti in prossimità, si trovano reciprocamente e cominciano autonomamente a comunicare (eccetto nel caso in cui sia richiesta un'autenticazione).

- *È a consumo ridotto*: questa caratteristica che ne lo strumento più adatto per essere integrato all'interno di apparecchi compatti, in cui non è previsto lo spazio per voluminose batterie.

- *È a basso costo*: attualmente la spesa sostenuta da un'azienda per introdurre

questa tecnologia all'interno di prodotti di dimensioni ridotte è di circa 15 euro, ma nel giro di alcuni anni dovrebbe scendere sotto i 5 euro per unità.

- *La banda ISM è regolata ma libera*: gli stati si sono accordati su un unico standard per permettere agli utenti di utilizzare lo stesso dispositivo Bluetooth virtualmente in ogni parte del mondo.
- *Gestisce sia la voce che i dati*: la sua capacità di gestire simultaneamente dati e voce rende possibili innovazioni quali auricolari da utilizzare a mani libere con applicazioni integrate per l'invio di fax ed email.
- *Il segnale è omnidirezionale e può passare attraverso i muri*: i dispositivi non hanno bisogno di essere a vista e posso anche essere in stanze diverse.

3.4.2 Topologia di rete

Un altro punto di forza del Bluetooth è la topologia di rete, che permette la creazione di reti ad-hoc particolarmente flessibili. I dispositivi possono interagire secondo due diversi schemi.

Lo schema più semplice è chiamato *piconet*. In questa configurazione possono essere contemporaneamente presenti 8 dispositivi: 1 solo *master* e 7 *slave*. Il *master* controlla tutto il traffico e se due *slave* vogliono comunicare tra di loro, il messaggio deve passare attraverso il *master*. La piconet utilizza un semplice meccanismo di *polling*: richiede a turno ad ogni *slave* se ha un messaggio da inviare.

La comunicazione è basata su pacchetti. Quando un'unità riceve un pacchetto controlla prima di tutto il destinatario: se non è diretto a quell'unità, il pacchetto viene rilasciato. In questo modo tutti i dispositivi della rete possono condividere un unico canale logico.

La banda resa disponibile da questo schema è di 1Mbit/s, che può portare fino a 721Mbit/s di dati.

Il secondo schema (*scatternet*), rappresentato nella figura 3.2 è più complesso e permette la sovrapposizione di più *piconet*, fino ad un massimo di 10. In questo modo si può raggiungere una banda complessiva di 10Mbit/sec. Le *piconet* formano una *scatternet* quando hanno un punto in comune, tipicamente realizzato con un dispositivo in comune a diverse *piconet*.

3.4.3 Bluetooth Protocol Stack

Tutte le comunicazioni che avvengono tra due dispositivi Bluetooth passano attraverso il Bluetooth Protocol Stack.

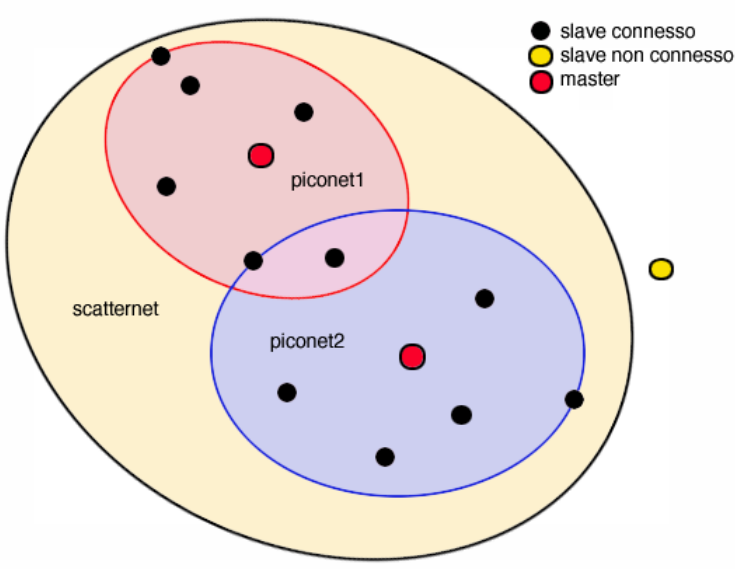

Figura 3.2: *Scatternet*

Lo stack del Bluetooth, rappresentato in figura 3.3 è diviso in due parti: il controllore, tipicamente hardware, e l'*host* software con cui le applicazione ed i servizi interagiscono.

Il sistema realizzato in questa tesi interagisce in modo esplicito solo con i livelli più alti dello stack, analizzati di seguito:

- *HCI*: è il livello software più basso, che si interfaccia direttamente con l'hardware.
- *L2CAP*: gestisce la segmentazione e il raggruppamento dei pacchetti, il protocollo di multiplexing e fornisce informazioni sulla qualità del servizio.
- *SDP*: le applicazioni utilizzano questo livello per identificare servizi Bluetooth disponibili.
- *RFCOMM*: questo livello opera per emulare una connessione seriale via Bluetooth.

In alcuni casi è presente un ulteriore livello al di sopra di tutti questi: l'OBEX (*Object Exchange Protocol*). Originariamente definito dalla *Infrared Data Association (IrDA)* [34], l'OBEX permette lo scambio di oggetti quali file, vCard e vCalendar o la sincronizzazione di dati su dispositivi diversi.

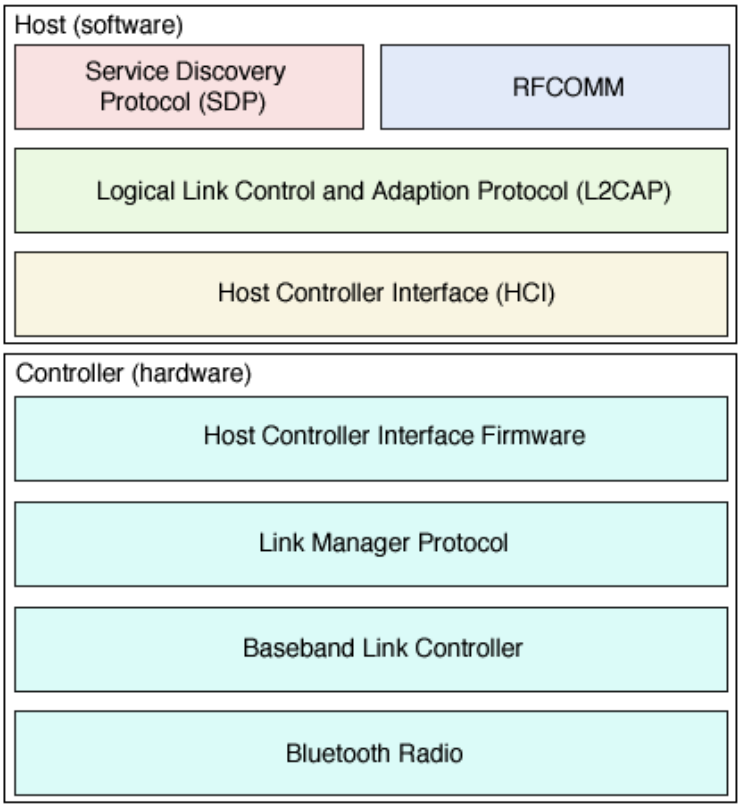

Figura 3.3: *Bluetooth Protocol Stack*

3.4.4 Profili

Ogni profilo descrive una funzionalità specifica dell'uso del Bluetooth come meccanismo di comunicazione per diverse categorie di apparecchi. In questo modo ogni distributore può decidere quali profili rendere disponibili sui propri dispositivi, assicurando interoperabilità e consistenza tra dispositivi diversi. Ovviamente è sempre possibile espandere il contesto di azione di un apparecchio, realizzando profili ulteriori rispetto a quelli supportati.

I profili più comuni per un'applicazione come quella realizzata in questa tesi sono i seguenti:

- *Generic Access Profile (GAP)*: definisce l'uso dei livelli più bassi dello stack, incluse le funzionalità di gestione del dispositivo. Tutti i dispositivi implementano questo profilo.
- *Service Discover Application Profile (SDAP)*: definisce le funzionalità relative al *Service Discovery*, applicate a SDP e L2CAP.

- *Serial Port Profile (SPP)*: definisce i requisiti necessari per l'emulazione di una connessione seriale, quindi specifica l'uso del RFCOMM, L2CAP e SDP per poterla implementare.
- *Generic Object Exchange Profile (GOEP)*: definisce i requisiti di interoperabilità tra OBEX, SPP e GAP per gestire il trasferimento di file di vario tipo e la loro sincronizzazione.
- *File Transfer Profile (FTP)*: definisce i requisiti di interfaccia con l'utente e di interoperabilità per l'uso del GOEP, OBEX e SDP.

3.4.5 Sicurezza

Lo standard Bluetooth garantisce la sicurezza dei dati trasmessi e memorizzati sul dispositivo, con la combinazione di tre meccanismi:

- *Pseudo-random frequency hopping*: questa tecnica di modulazione rende particolarmente difficile *sniffare* (ovvero intercettare e leggere) i dati trasmessi.
- *Autenticazione*: permette ad un utente di limitare la possibilità di connessione ad un ristretto e scelto insieme di dispositivi.
- *Criptazione*: l'utilizzo di chiavi di varia lunghezza per la criptazione dei dati li rende intelleggibili sono alle persone autorizzate.

Inoltre, tutti i dispositivi Bluetooth devono implementare il *Generic Access Profile (GAP)* che definisce tre possibili modalità di sicurezza:

- *Modo 1*: modo insicuro, nessuna modalità di sicurezza viene stabilita.
- *Modo 2 (service-level enforced security)*: quando un dispositivo opera in questa modalità non viene istituito nessun meccanismo di sicurezza prima che il canale di comunicazione venga stabilito. Permette a più applicazione di avere politiche di accesso diverse, ed eseguirle in parallelo.
- *Modo 3 (link-level enforced security)*: in questo caso le procedure di sicurezza vengono stabilite prima che il set-up del canale sia concluso.

È comunque importante mettere in evidenza che il metodo di autenticazione supportato dal Bluetooth autentica il dispositivo ma non l'utente che ne fa uso. Quindi, per la comunicazione di dati sensibili, è sempre consigliabile implementare almeno un ulteriore livello di sicurezza (ad esempio la criptazione dei dati stessi).

3.5 La sinergia di Java e Bluetooth

Java ha fornito la prima e (fino ad ora) unica standardizzazione delle API per il Bluetooth. Le JABWT (*Java API Bluetooth Wireless Technology*) sono il risultato del JSR-82 (*Java Specification Request 82*) [35] e definiscono il pacchetto opzionale relativo alla tecnologia wireless Bluetooth per la J2ME.

Come avviene per ogni *Specification Request*, all'approvazione è stato definito un gruppo di esperti che contribuissero insieme a Motorola, ditta leader nel settore, alla definizione delle API. Alle JABWT hanno collaborato Nokia, SonyEricsson, Mitsubishi, Symbian, Sun Microsystems, Rococo Software e molte altre.

Lo scopo di queste API è quello di definire un'architettura aperta che permetta la creazione di un ambiente di sviluppo per ogni tipo di applicazione Bluetooth.

Le API rappresentano un pacchetto opzionale sviluppato per operare al top del CLDC. Possono essere utilizzate sia come estensione delle capacità dei profili della J2ME (come il MIDP) per lo sviluppo di midlet da eseguire sui cellulari, oppure in diretta connessione al CLDC per applicazioni Bluetooth da eseguire sul desktop (ad esempio per la comunicazioni tra due computer portatili).

La definizione delle JABWT ha portato al rilascio di due pacchetti separati:

- `javax.bluetooth`: il cuore della API per il Bluetooth, per la gestione completa della comunicazione.
- `javax.obex`: dato che Bluetooth può utilizzare anche lo standard OBEX per l'invio di file ed oggetti quali vCard e vCalendar, è stato definito un supporto per questo tipo di operazioni.

Per poter quindi scrivere un'applicazione con le API Java Bluetooth si devono utilizzare le classi ed i metodi definiti in questi due pacchetti o in quelli ugualmente contenuti in `javax` (ovvero `javax.microedition`).

Per eseguire invece questa stessa applicazione dopo averla compilata, è necessario disporre di un'implementazione delle API stesse, resa disponibile dai costruttori di cellulari che supportano queste specifiche API (come il Nokia 6600) oppure da alcune Software House specializzate in questi prodotti (come Rococo Software o Atinav). Disponendo di un'implementazione, l'esecuzione del codice è assolutamente svincolata dal tipo di stack utilizzato.

Con ogni altro linguaggio di programmazione (come il C o il C++) invece, la scrittura di un programma per il Bluetooth dipende totalmente dallo stack utilizzato. Questo in pratica significa che un programmatore C++ che lavora con Linux (e quindi utilizza i metodi e le classi fornite dallo stack *open-source* Bluez) deve completamente riscrivere il proprio codice per eseguire lo stesso programma sul Palm, su Windows o su uno dei tanti cellulari che attualmente supportano le comunicazioni Bluetooth.

Utilizzando invece le JABWT, il codice non subisce alcuna modifica passando da un sistema operativo all'altro, ovvero da uno stack all'altro. Questo è possibile perché, implementando le API Java per il Bluetooth, ogni distributore fornisce al programmatore un set di metodi e di classi che sono definite a priori dal JSR-82. L'interfaccia con lo stack risulta quindi assolutamente trasparente e il programmatore può lavorare ad un livello molto più alto, concentrandosi sulle funzionalità della propria applicazione.

L'introduzione delle Java API per il Bluetooth ha quindi apportato numerosi benefici, come evidenziato da un interessante articolo [36] della Rococo Software [37] (ditta irlandese leader nel settore delle comunicazioni wireless).

- *Portabilità*: il codice Java è ben noto per poter essere portato semplicemente da un sistema operativo all'altro. Questa caratteristica permette agli sviluppatori di scrivere un'unica applicazione per dispositivi diversi. Nella sua migliore accezione, soddisfa completamente lo slogan di "Write once, run anywhere." (si scrive una volta, esegue ovunque). Questa caratteristica assume inoltre un significato particolare se applicata ad un contesto, come quello eterogeneo dei dispositivi wireless, in cui la diversità tra le varie marche sul mercato spinge fortemente all'utilizzo di uno standard comune.

- *Mobilità*: dopo essere stato compilato, un programma scritto in Java può essere inviato direttamente *over-the-air (OTA)* ai dispositivi mobili su cui poi verrà eseguito. Questo permette di fornire servizi attraverso una rete mobile, un portale web o un altro dispositivo che in quel momento opera da ponte. La possibilità di caricare nuove applicazioni al momento (ovvero *on-the-fly*), magari utilizzandole anche solo una volta, è fondamentale per dispositivi con risorse limitate soprattutto di memoria.

- *Networking*: Java è noto per essere stato sviluppato con un'attenzione particolare per le reti, quindi il *networking* è sicuramente uno dei suoi punti di forza. La nascita e il rafforzamento di sistemi di comunicazione wireless non ha quindi richiesto la definizione di un nuovo linguaggio, ma ha semplicemente beneficiato dei punti di forza sul *networking* da sempre presenti in Java.

- *Interoperabilità*: L'effettiva realizzazione di una concreta operabilità tra dispositivi Bluetooth dipende da due fattori. Prima di tutto, gli stack devono interoperare tra di loro ad ogni livello e devono concordare i protocolli dei comandi, gli eventi e le azioni. Inoltre, le applicazioni che usano questi stack devono accordarsi su certi aspetti del loro uso, come ad esempio l'inizializzazione o il significato da attribuire ai dati inviati e ricevuti. Anche se i profili Bluetooth risolvono la maggior parte di questi problemi, solo attraverso uno

standard semanticamente definito e conforme alle API si può garantire una vera interoperabilità.

Capitolo 4

Il sistema CareMate

4.1 Hardware e software di base

La scelta dell'hardware e del software di base per lo sviluppo del sistema CareMate risponde a due esigenze fondamentali:

- La conformità ai requisiti di sistema;
- Le restrizioni introdotte dalla mancanza di implementazione hardware e software delle API Java per il Bluetooth.

Dato che le JABWT sono relativamente recenti (2001), è stata riscontrata una notevole difficoltà nel reperire uno strumento di programmazione (sia hardware che software) che permettesse lo sviluppo di un software conforme a questo standard. Inoltre, nella realizzazione di un sistema in cui hardware e software sono strettamente correlati, questa limitazione ha avuto un enorme peso sulle scelte progettuali.

La fase di analisi di fattibilità del sistema è quindi stata basata principalmente sulla ricerca di uno strumento che fornisse un'implementazione delle API Java per il Bluetooth. Sono stati così identificati alcuni stack Bluetooth che implementano le JABWT, tra cui lo stack fornito da Atinav [38], il progetto *JavaBluetooth* [39] e *Impronto* [40] di Rococosoft.

I primi due, tuttavia, si sono rivelati inadatti. Infatti:

- *Lo stack di Atinav*: fornisce supporto separato per J2ME e J2SE, vendendo questi due stack ad un prezzo molto elevato e, come ogni prodotto da vendita, senza i sorgenti. Inoltre il pacchetto `javax.obex` (necessario per l'invio di file tra dispositivi Bluetooth) non è implementato.
- *Lo stack JavaBluetooth*: ha il vantaggio di fornire anche i sorgenti, ma non ha (ancora) implementato tutte le funzionalità richieste per questo progetto, compreso il supporto per J2ME e il pacchetto `javax.obex` .

La soluzione è stata invece trovata con *Impronto* della Rococosoft, che implementa interamente le JABWT (quindi entrambi i pacchetti `javax.bluetooth` e `javax.obex`) sia per J2SE (*Java 2 Standard Edition*) che per J2ME. Inoltre, è disponibile una versione per Linux a licenza gratuita per le Università.

Questi fattori hanno quindi portato a scegliere *Impronto* come base di sviluppo del software Bluetooth, e Linux come sistema operativo. In particolare è stata scelta una distribuzione Gentoo [41] che si è rilevata la migliore sia per la compilazione di *Impronto*, che per l'installazione dei *tool* di supporto per lo stack Bluez [42] di Linux. Anche sul robot mobile (*Nomad*), già dotato di sistema operativo Linux, è stata installata una distribuzione Gentoo.

Infine, per abilitare la comunicazione Bluetooth sui computer fissi e portatili utilizzati nella realizzazione del progetto, sono stati acquistati degli adattatori Digicom Palladio USB [43] (figura 4.1), che coprono un raggio d'azione di circa 100m.

Figura 4.1: *Adattatore Bluetooth Digicom Palladio USB*

Questi strumenti hardware hanno permesso di sviluppare applicazioni Bluetooth rispondendo in questo modo al requisito di mobilità. Inoltre, dato il supporto per la *Micro Edition*, sono stati soddisfatti anche i requisiti relativi all'interoperabilità e allo sviluppo di un sistema distribuito.

La mancanza di un dispositivo reale che supporti concretamente le JABWT non ha reso possibile da questo punto di vista soddisfare completamente anche i requisiti di compattezza del dispositivo e di trasparenza del sistema. Esistono infatti in commercio due modelli di cellulari (il Nokia 6600 e il SonyEricsson P900) che implementano il pacchetto `javax.bluetooth`. Tuttavia, l'elevato costo e la mancanza di conformità totale allo standard, ne fanno strumenti ancora inadatti per lo sviluppo di un sistema di questo tipo.

Data l'importanza delle scelte adottate, viene di seguito elencato in modo esplicito l'hardware di base utilizzato:

- Postazioni Linux con distribuzione Gentoo;
- Robot mobile (*Nomad* [44], mostrato nella figura 4.2) con sistema operativo Linux (distribuzione Gentoo);

- Adattatore Digicom Palladio USB.

Figura 4.2: *Nomad 200 con braccio meccanico Manus*

Per quel che riguarda l'analisi del software, è importante mettere in evidenza l'utilizzo congiunto di *Impronto* [40] e *Me4Se* [45]. *Me4Se* permette l'esecuzione di applicazioni scritte per la J2ME in ambiente J2SE (ovvero sul *desktop*) e fornisce un valido supporto per quelle piattaforme per cui non è stato ancora realizzato un buon emulatore.

In questo modo si può effettuare la fase di debug di un midlet Bluetooth direttamente sul *desktop*, evitando inutili perdite di tempo nell'invio al cellulare dei file `jar` compilati. Per lo stesso motivo è stato utilizzato anche l'emulatore del cellulare Nokia 6600, nella versione per Linux [46].

In mancanza di un cellulare in grado di eseguire il midlet Bluetooth conforme allo standard JSR-82, lo strumento *Me4Se* è stato utilizzato per dimostrare il

buon funzionamento del sistema CareMate. Diversamente dall'emulatore della Nokia, che opera in un ambiente simulato e non può collegarsi con dispositivi esterni, *Me4Se* permette infatti di realizzare vere comunicazione wireless via Bluetooth.

L'unione di *Impronto* e *Me4Se* consente di sviluppare applicazioni Bluetooth in ambiente non simulato ma reale, aggirando l'ostacolo della mancanza dell'hardware necessario (ovvero di un cellulare con supporto per la J2ME e per le JABWT completamente implementati).

Oltre alle applicazioni correlate al Bluetooth, sono stati utilizzati strumenti di terzi per altri scopi. Questo è il caso di MySql [47], con cui è stato creato il database di tutti i dati dei pazienti e delle attività realizzate nel sistema, e di JDBC [48], con cui è stata sviluppata l'interfaccia di basso livello al database stesso. JDBC rappresenta l'insieme delle API Java che permettono di gestire database relazionali come MySql e fornisce uno strato di accesso ai database completo e soprattutto completamente ad oggetti.

Per realizzare il *parsing* delle informazioni del paziente (memorizzate in un file `xml`) sulla piattaforma J2ME, è stato utilizzato il *parser* kXml [49], che coniuga dimensioni ridotte (circa 6KB) ed una buona velocità. Per quel che riguarda la parte *server* della comunicazione Bluetooth via OBEX, sono stati utilizzati i servizi offerti da KDEBluetooth per Linux. Questo strumento fornisce il supporto necessario per le comunicazioni OBEX realizzate con lo stack Bluez di Linux.

Infine, è significativo ricordare l'utilizzo di Apache Ant [50] per la fase di compilazione. La scrittura di un file `build.xml` ha consentito infatti di semplificare e automatizzare la compilazione dei programmi sviluppati, spesso dipendenti da un elevato numero di parti. Questo strumento si è rivelato particolarmente utile per la flessibilità e la trasparenza che consente nella compilazione di programmi complessi

Riassumento, il software di base utilizzato è stato il seguente:

- *Impronto Developer Kit* della Rococosoft versione 1.2 per Linux;
- *Me4Se*;
- Emulatore Nokia 6600 per Linux;
- MySql e JDBC;
- Parser kXml per J2ME;
- KDEBluetooth;
- Ant per la compilazione di tutti i programmi sviluppati.

4.2 Architettura del sistema

Nel sistema sono presenti quattro differenti programmi eseguiti su macchine diverse. Data l'intrinseca relazione tra dispositivo e software implementato, si assume (ove non specificato diversamente) di indicare con il nome del dispositivo anche l'applicazione eseguita sull'apparecchio stesso.

Seguendo il criterio appena introdotto, il sistema è formato dalle seguenti parti:

- *DataBase*: formato dal database contenente i dati del sistema, dal programma che gestisce il database stesso e presenta un'interfaccia grafica, e dal computer su cui viene eseguito.
- *Device*: con questo nome si intende il dispositivo su cui è memorizzato il profilo del paziente e la relativa applicazione (midlet).
- *Nomad*: formato dal robot mobile e dal suo specifico programma.
- *Embedded*: dispositivo *embedded* che aggiorna le attività dei pazienti, comprensivo della sua applicazione.

Ogni elemento del sistema può essere considerato come un *micro-sistema* interagente con l'esterno, ovvero con gli altri *micro-sistemi*, per generare il sistema CareMate complessivo.

Utilizzando il *Sequence Diagram* di figura 4.3 si può analizzare il funzionamento complessivo del sistema. Questo processo ha inizio con l'inserimento nel *DataBase* dei dati di un nuovo paziente. Quando questa operazione è conclusa, l'infermiere invia il profilo sul *Device* che sarà poi assegnato al nuovo paziente. Dato che insieme al profilo viene spedito anche il midlet (già in formato `jar` eseguibile), il risultato dell'operazione di invio del file corrisponde anche alla creazione del *Device* stesso.

Nel profilo sono contenute le informazioni del paziente e delle attività terapeutiche da eseguire durante tutta la settimana. Queste attività sono inizialmente memorizzate come "incompiute". Quando un'attività viene completata il suo stato diventa "ok", oppure viene settato a "scaduta" quando non viene portata a termine nell'orario prestabilito.

Quando il file (con midlet e profilo) viene ricevuto dal *Device*, questo può cominciare ad interagire con l'*Embedded* e con il *Nomad*. Se uno dei tanti *Embedded* inseriti nell'ambiente registra la presenza di un *Device*, invia a questo il messaggio `embedded`. Il *Device* allora, ricevendo questo messaggio, estrae le informazioni sul paziente e su tutte le attività quotidiane che il paziente deve ancora compiere (ovvero sia quelle incompiute che quelle scadute) e le invia all'*Embedded*. L'*Embedded* può così controllare che il profilo del paziente sia compatibile con l'attività che sta per fare. Se questo controllo dà esito positivo, l'*Embedded* invia un messaggio di

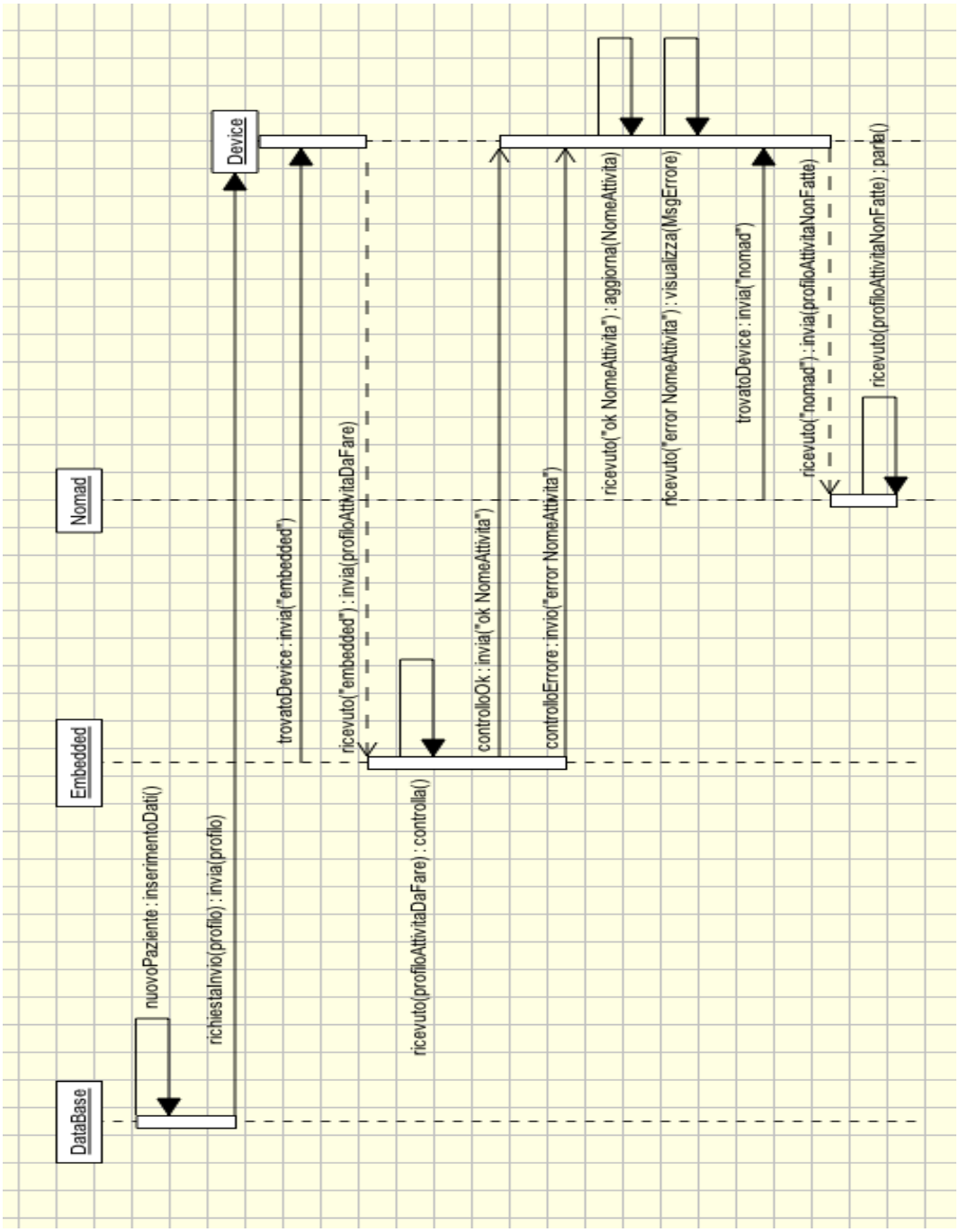

Figura 4.3: *UML Sequence Diagram rappresentante la comunicazione all'interno del sistema CareMate*

conferma, che verrà utilizzato dal *Device* per aggiornare l'attività corrispondente. Nel caso in cui invece il controllo identifichi un'incongruenza tra le attività ancora da eseguire e quella che sta per compiere, l'*Embedded* invia un messaggio di errore. In questo caso il *Device* mette in evidenza l'errore con una scritta sul display oppure con un suono o una luce rossa lampeggiante.

Nel caso in cui sia il *Nomad* a registra la presenza di un *Device*, invia a questo il messaggio `nomad`. Il *Device* allora estrae le informazioni sul paziente e sulle sole attività quotidiane scadute e le invia al *Nomad*. A questo punto il *Nomad* utilizza il sintetizzatore sonoro per avvertire il paziente della presenza delle sue attività scadute, in caso ve ne siano. Nel caso in cui non siano presenti attività scadute, il *Nomad* conferma al paziente il suo buon comportamento.

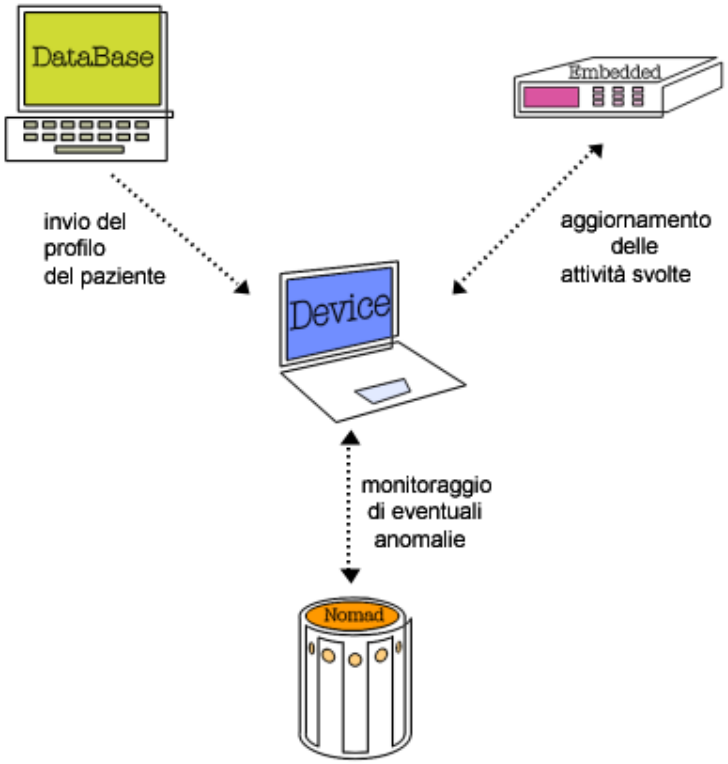

Figura 4.4: *Interazione tra le varie parti del sistema CareMate.*

La figura 4.4 riassume infine l'interazione tra le varie parti del sistema. La figura del paziente è stata volutamente esclusa dallo scenario, per non assegnare allo stesso un ruolo attivo che non gli appartiene: la trasparenza del sistema implica infatti che il paziente non abbia alcuna interazione con il *Device*.

Lo scambio di informazioni avviene sempre ad alto livello. Questo significa che

ogni dispositivo elebora localmente le informazioni di propria competenza ed invia solo i dati già elaborati. In questo modo:

- Si evita l'invio di messaggi troppo lunghi, che richiederebbe tempi elevati incompatibili con la mobilità ed il dinamismo dell'ambiente.
- La responsabilità della comprensione (e quindi dell'elaborazione) dei dati a basso livello è assegnata al dispositivo che ha il maggior numero di informazioni per portarla a termine.

4.3 DataBase

DataBase è un computer fisso utilizzato dagli operatori (ad esempio infermieri) per l'inserimento, la modifica, la cancellazione e la visualizzazione dei dati dei pazienti e delle attività terapeutiche svolte nel sistema. Questi dati sono salvati su un database centrale, a cui si accede tramite un'interfaccia grafica realizzata per facilitare queste operazioni.

L'analisi di questa applicazione è stata suddivisa in quattro parti:

- *Design* del database relazionale per la memorizzazione dei dati del sistema.
- Interazione a basso e medio livello tra il database e l'interfaccia.
- Realizzazione dell'interfaccia grafica per facilitare l'esecuzione delle operazioni sul database.
- Comunicazione Bluetooth.

4.3.1 Design del database relazionale

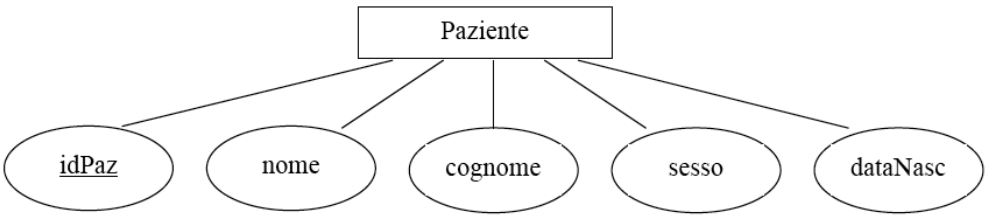

Figura 4.5: *Attributi della tabella "paziente".*

Nella prima fase di *design* del database relazionale contenente i dati del sistema, sono state definite due sole tabelle: *paziente* e *attivita*. Per ogni *paziente* sono stati

definiti gli attribuiti mostrati nella figura 4.5. L'*idPaz* (ovvero l'identificativo unico del paziente) viene assunto come *Primary Key (PK)* della tabella *paziente*, e per questo motivo è stato sottolineato. Per ogni *attività* sono stati invece definiti gli attributi mostrati nella figura 4.6. La *PK* di *attivita* è *idAtt*.

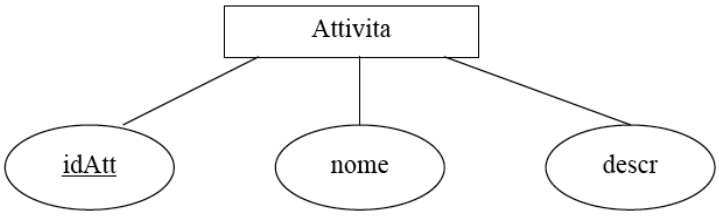

Figura 4.6: *Attributi della tabella "attivita".*

Paziente ed *attivita* sono legati dalla relazione mostrata graficamente nella figura 4.7. Questa relazione può essere letta nel seguente modo:

- Ad ogni paziente possono essere assegnate da 1 ad A attività.
- Ogni attività può essere assegnata ad un numero di paziente compreso tra 1 e P.

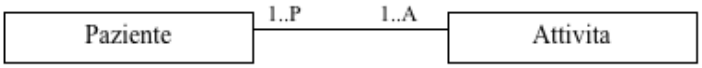

Figura 4.7: *Relazione "molti a molti" tra "paziente" e "attivita".*

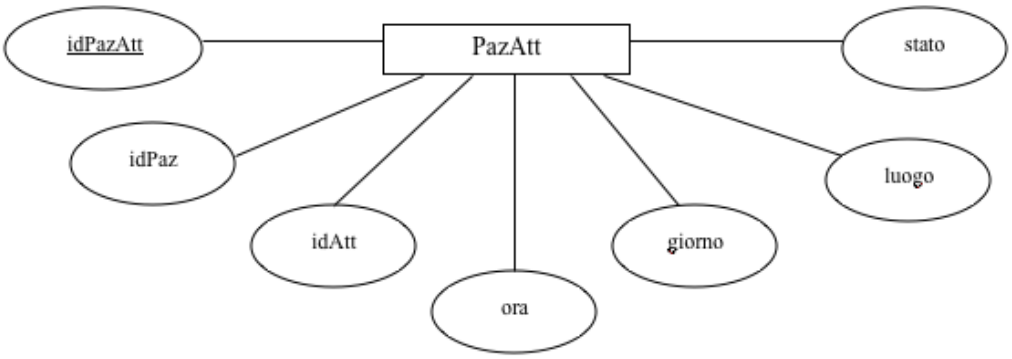

Figura 4.8: *Attributi della tabella "pazAtt".*

La relazione 4.7 richiede l'introduzione di una terza tabella (chiamata *pazAtt*, per indicare l'interazione tra le due precedenti) in cui memorizzare le attività assegnate ai pazienti del sistema, comprensive del giorno, l'ora e il luogo in cui

ogni attività deve avvenire. Questa terza tabella evita l'introduzione di informazioni ridondanti e facilita il mantenimento del sistema in uno stato sempre coerente.

Gli attributi di *pazAtt* sono rappresentati nella figura 4.8. La *PK* è *idAttPaz*, mentre *idPaz* e *idAtt* compaiono in questa tabella come riferimenti esterni (*Foreign Key)*.

Le specifiche così prodotte sono state formalizzate nello script `caremate.sql` che, all'interno dell'ambiente *MySql*, permette la creazione del database `caremate` e lo inizializza con alcuni valori di esempio. In particolare, la creazione del database e delle tabelle sopra analizzate, è realizzata dal codice mostrato nel listato 4.1.

```
### Creazione del database
CREATE DATABASE caremate;

### Connessione al database
USE caremate;

### Creazione delle tabelle
CREATE TABLE paziente
(
        idPaz SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        nome CHAR(40) NOT NULL,
        cognome CHAR(40) NOT NULL,
        sesso ENUM('m', 'f') NOT NULL,
        dataNasc DATE NOT NULL,
        PRIMARY KEY (idPaz)
);

CREATE TABLE attivita
(
        idAtt SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        nome CHAR(30) NOT NULL,
        descr CHAR(80) NOT NULL,
        PRIMARY KEY (idAtt)
);

CREATE TABLE pazAtt
(
        idAttPaz SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        idPaz SMALLINT UNSIGNED NOT NULL,
        idAtt SMALLINT UNSIGNED NOT NULL,
        ora ENUM ('08:00', '09:00', '10:00', '11:00', '12:00', '13:00', '14:00',
            '15:00', '16:00' ,'17:00' ,'18:00', '19:00', '20:00') NOT NULL,
```

```
        giorno  ENUM('lun', 'mar', 'mer', 'gio', 'ven', 'sab', 'dom') NOT NULL,
        luogo   ENUM('ambulatorio', 'camera', 'piscina', 'palestra', 'giardino')
        NOT NULL,
        stato   ENUM('ok', 'incompiuta', 'scaduta')  NOT NULL,
        PRIMARY KEY (idAttPaz),
        FOREIGN KEY (idPaz)
                REFERENCES paziente
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (idAtt)
                REFERENCES attivita
                ON DELETE CASCADE
                ON UPDATE CASCADE
);

### Visualizzazione delle tabelle
SHOW TABLES;

### Visualizzazione delle caratteristiche delle tabelle
SHOW columns FROM paziente;
SHOW columns FROM attivita;
SHOW columns FROM pazAtt;
```

Listato 4.1: *Script* `caremate.sql` *per la creazione del database e la visualizzazione delle caratteristiche delle tabelle*

4.3.2 Interazione tra database e interfaccia grafica

Per realizzare l'interazione tra il database `sql` e l'interfaccia grafica sono state create le classi `MysqlInterface` (che utilizza JDBC) e `DBHandler`. In `MysqlInterface` l'interazione con il database `caremate` avviene in modo diretto, mentre `DBHandler` opera ad un livello intermedio tra la classe di accesso diretto al database (`MysqlInterface`) e l'interfaccia vera e propria (ovvero `CareMate`).

```
Class.forName("org.gjt.mm.mysql.Driver");
connection = DriverManager.getConnection(
                        "jdbc:mysql:// localhost:3306/caremate",
                        "root",
                        "caremate");
```

Listato 4.2: *Parte del codice del costruttore di* `MysqlInterface` *per l'accesso al database* `caremate`

L'accesso al database `caremate` avviene all'interno nel costruttore della classe `MysqlInterface`. Il listato 4.2 mostra il cuore di questa procedura. La prima riga definisce il *Driver mysql* utilizzato, e la seconda riga ottiene la connessione. Il primo argomento del metodo `getConnection` è il nome del database che si vuole aprire, il secondo indica l'utente e il terzo la sua password.

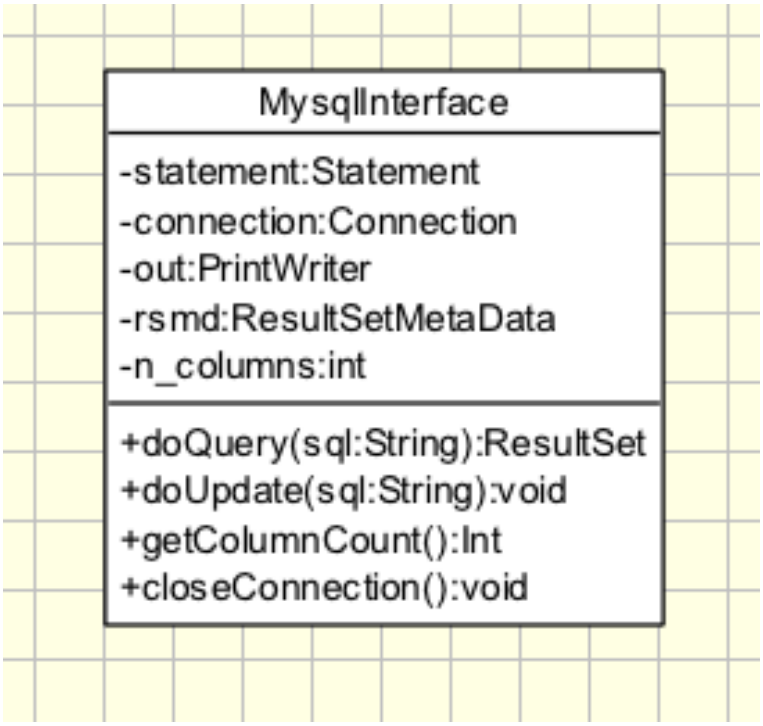

Figura 4.9: *UML Class Diagram della classe* `MysqlInterface`*.*

Il *Class Diagram* di figura 4.9 mette in evidenza i metodi e gli attributi della classe `MysqlInterface`, che fornisce gli strumenti utilizzati poi da `DBHandler`. In particolare:

- `doQuery` permette di inviare una richiesta al database per conoscere le informazioni in esso memorizzate.
- `doUpdate` inoltra una richiesta di aggiornamento.
- `getColumnCount` restituisce il numero di colonne di una data tabella.
- `closeConnection` chiude la connessione con il database.

La classe `DBHandler` riassume nei suoi 27 metodi tutti gli accessi al database necessari per implementare l'interfaccia grafica.

4.3.3 Interfaccia grafica

L'nterfaccia presenta un menu principale con quattro possibili scelte, analizzate di seguito in dettaglio.

Dal menu *File*, come mostrato in figura 4.10, si accede a sette diversi sotto-menu.

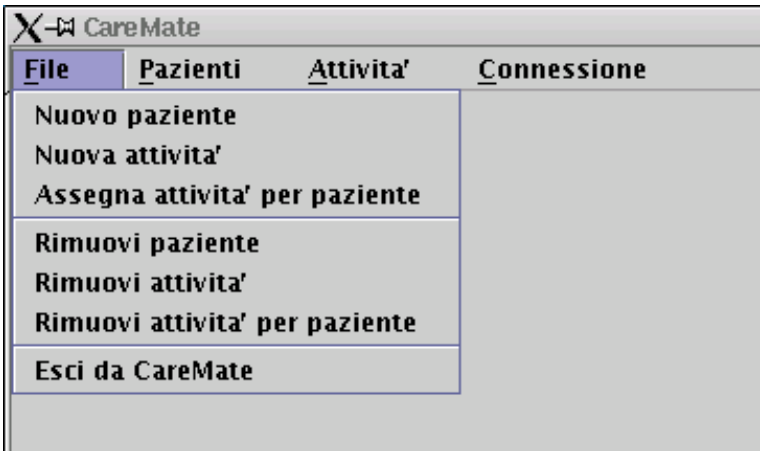

Figura 4.10: *Snapshot della scelta del menu "File"*

Il primo, *Nuovo paziente*, permette la creazione di un nuovo paziente presentando una serie di interfacce che guidano l'utente nell'inserimento dei dati necessari. Il procedimento seguito per la progettazione di questa interfaccia è mostrato nello *State Diagram* di figura 4.11.

Dopo aver selezionato *Nuovo paziente*, compaiono quattro campi (di cui uno a tendina, con la scelta obbligata tra due soli valori possibili) per l'inserimento dei dati del paziente. Quando questa operazione è completata, premendo il bottone *Aggiungi* si ottiene l'effettiva memorizzazione dei dati nel database centrale. Viene inoltre visualizzato un messaggio per confermare l'avvenuto inserimento.

La figura 4.12 mostra la successione di due snapshot dell'interfaccia appena descritta, presi rispettivamente durante l'inserimento dei dati, ed al termine della memorizzazione degli stessi.

Il sotto-menu *Nuova attività* ricalca quello appena descritto per l'inserimento di un nuovo paziente. L'unica differenza da sottolineare consiste nell'impossibilità di creare una nuova attività con nome equivalente a quello di una già esistente. Questa limitazione si manifesta nella visualizzazione di un messaggio di errore, che avverte della mancata memorizzazione della suddetta attività (figura 4.13).

Lo *State Diagram* che rappresenta l'evoluzione dell'interfaccia per l'inserimento di una nuova attività è mostrato in figura 4.14.

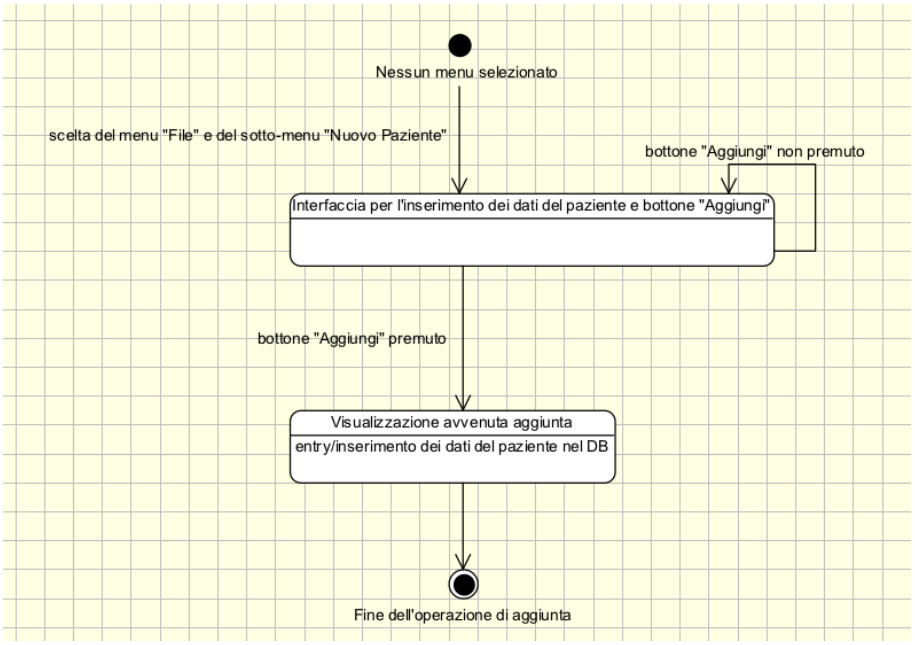

Figura 4.11: *UML State Diagram dell'interfaccia per l'inserimento di un nuovo paziente*

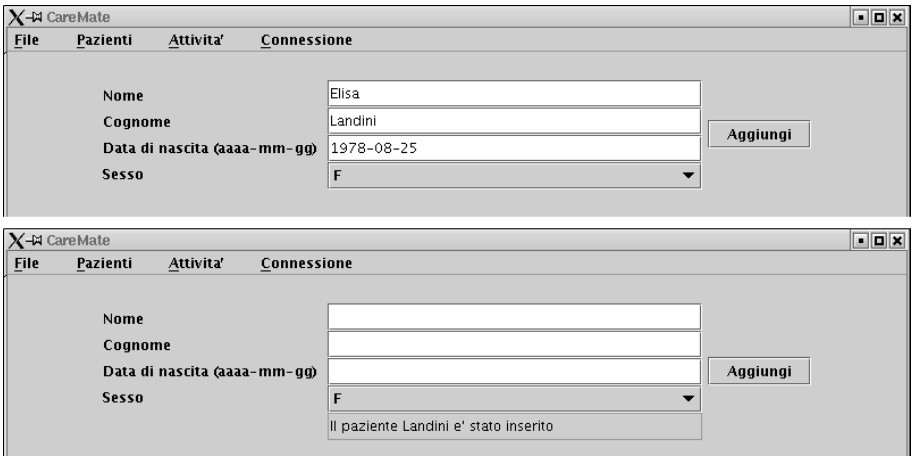

Figura 4.12: *Snapshot in successione: inserimento dei dati di un nuovo paziente e conferma dell'avvenuto inserimento.*

Il programma permette poi di assegnare un'attività ad un paziente, specificando anche il giorno, l'ora e il luogo in cui si deve verificare. Questa operazione è realizzata dal sotto-menu *Assegna attività per paziente*.

Scegliendo l'opzione *Assegna attività per paziente*, vengono visualizzati cinque menu a tendina (figura 4.15) corrispondenti ai cinque tipi di dati da scegliere (ov-

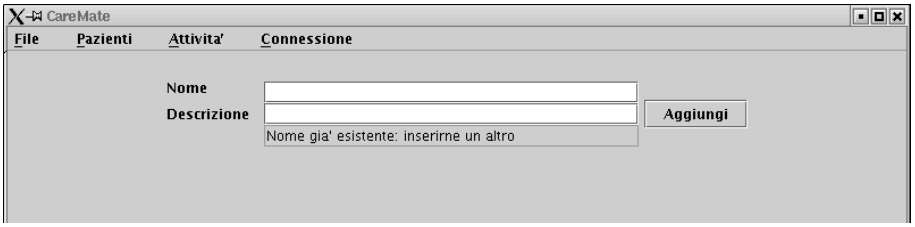

Figura 4.13: *Snapshot della visualizzazione di un messaggio di errore corrispondente al tentativo di inserimento di un'attività con nome uguale ad una già esistente.*

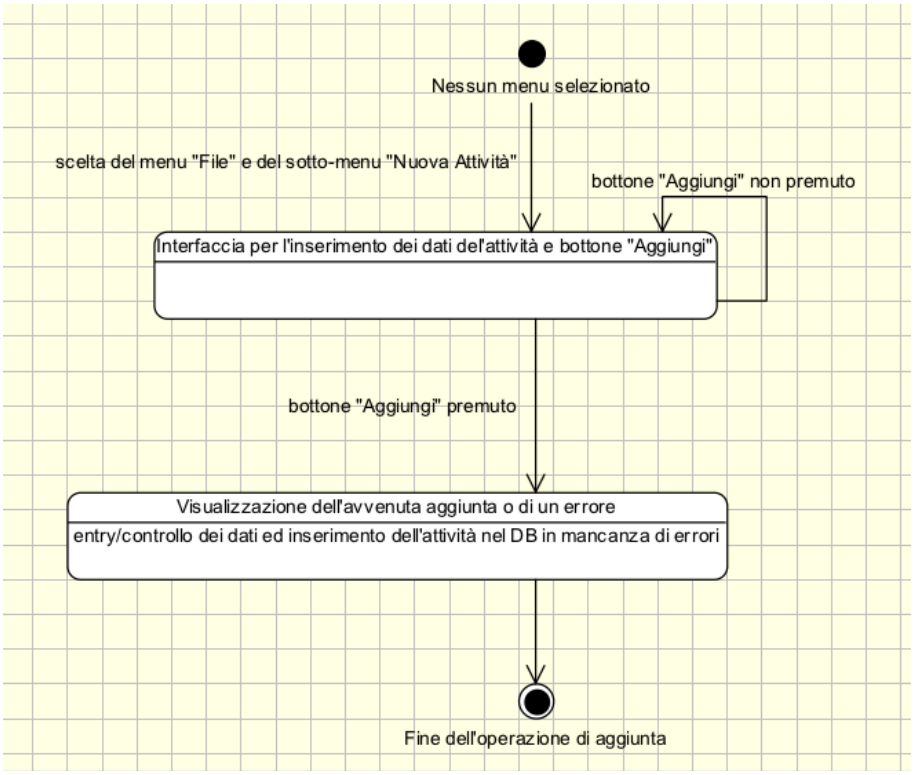

Figura 4.14: *UML State Diagram dell'interfaccia per l'inserimento di una nuova attività*

vero paziente, attività, giorno, ora e luogo). Come evidenziato nello *State Diagram* di figura 4.16, quando la scelta è stata completata, premendo *Aggiungi* si ottiene la memorizzazione dell'associazione tra il paziente e l'attività, in data, ora e luogo specificati. Dato che l'associazione implica anche la definizione di questi ultimi valori, la stessa attività assegnata ad esempio in orari o giorni diversi viene correttamente memorizzata con due diverse *entry*.

Figura 4.15: *Snapshot dell'interfaccia di scelta per assegnare un'attività ad un paziente, specificando anche giorno, ora e luogo.*

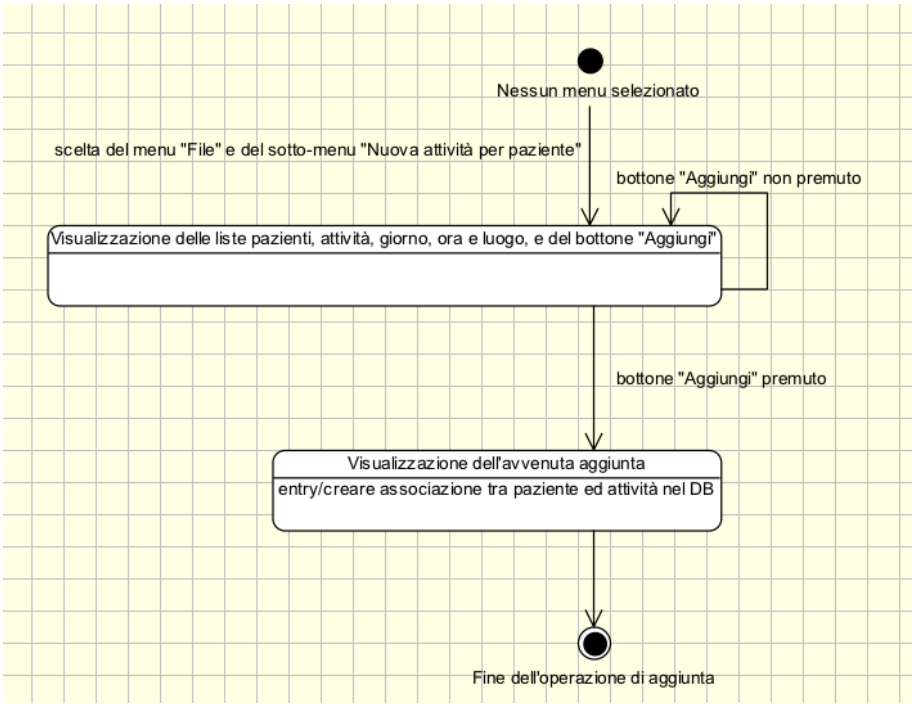

Figura 4.16: *UML State Diagram dell'interfaccia per l'assegnazione di un'attività ad un paziente*

In questa configurazione è possibile modificare più volte lo stesso menu a tendina (ad esempio scegliendo come giorno *lun* e poi, anche dopo aver selezionato altri dati, cambiarlo a *gio*). Questo è possibile perché l'acquisizione dell'insieme di valori scelti viene effettuata solo dopo aver premuto il bottone *Aggiungi*.

Le operazioni di rimozione dei dati di un paziente o di un'attività, rispettivamente identificate dal sotto-menu *Rimuovi paziente* e *Rimuovi attività*, seguono un *pattern* comune, come mostrano gli *State Diagram* a) e b) di figura 4.17.

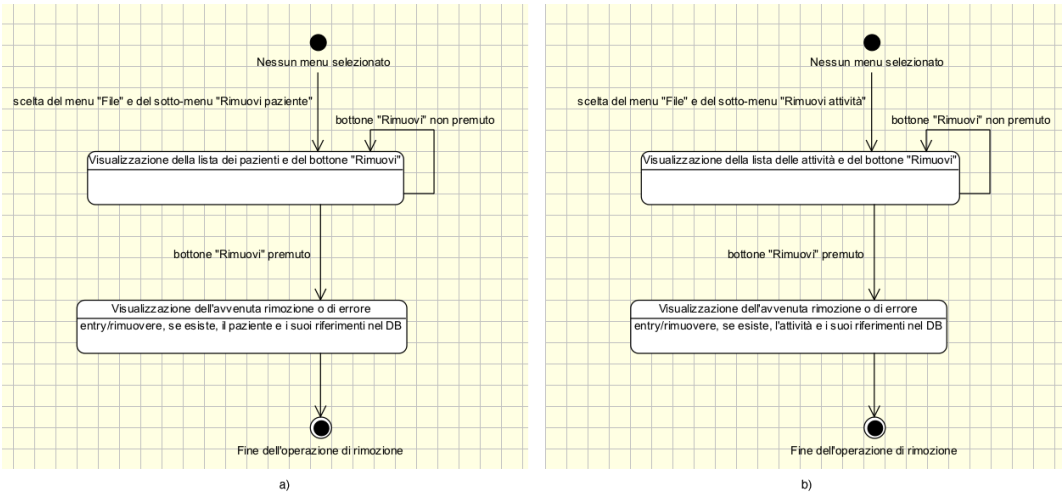

Figura 4.17: *a) UML State Diagram per la rimozione dei dati di un paziente. b) UML State Diagram per la rimozione dei dati di un'attività*

Data l'analogia tra queste due interfacce, analizziamo di seguito solo la rimozione di un'attività.

Seguendo l'evoluzione della grafica mostrata nello *State Diagram* 4.17 b), si identificano due fasi:

- La scelta dell'elemento da rimuovere fra tutti i possibili attualmente nel sistema, seguita dalla pressione del bottone *Rimuovi*.
- La visualizzazione di un messaggio di conferma o di errore.

La figura 4.18 mostra queste due fasi in successione. In particolare, in questo caso la seconda interfaccia visualizza un messaggio di conferma, corrispondente all'avvenuta eliminazione dell'elemento dal database.

La presentazione di un insieme di valori attraverso un menu a tendina implica il caricamento dei soli valori attuali del sistema. Questa in pratica significa che un dato (ad esempio un paziente), se rimosso in una precedente operazione, non comparirà più nella visualizzazione di un'operazione successiva. Quando tuttavia l'utente non cambia menu, ma rimane all'interno dello stesso, questo aggiornamento non è possibile. Osservando la lista dopo aver già rimosso un elemento, si vedrà che questo è comunque ancora presente. La figura 4.19 mette in evidenza il messaggio di errore visualizzato nel caso in cui si sia tentato di rimuovere un dato già cancellato dal database.

Quando un'attività viene effettivamente rimossa dal sistema, vengono anche cancellate tutte le sue occorrenze. Quindi vengono anche rimosse tutte le associazio-

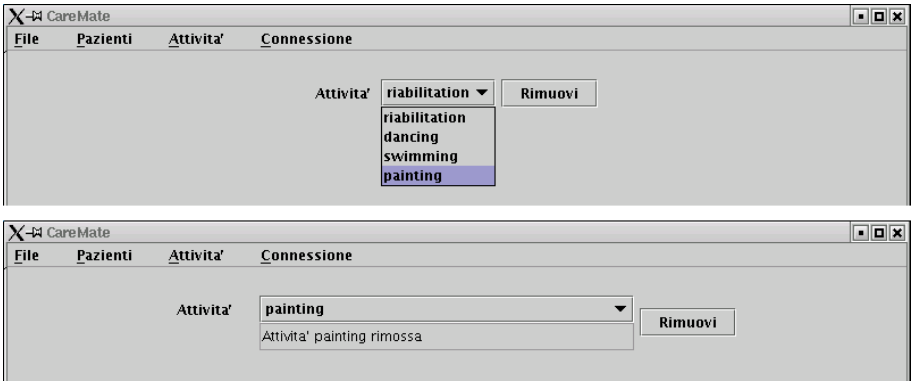

Figura 4.18: *Snapshot relativi alla rimozione (riuscita) di un'attività dal database*

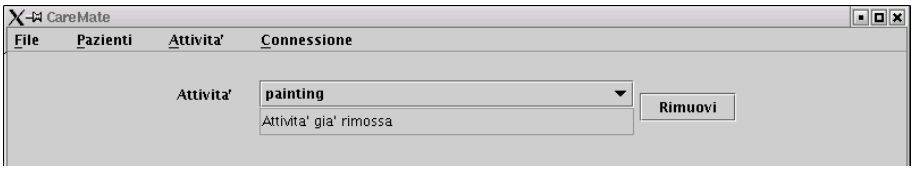

Figura 4.19: *Snapshot relativi al tentativo di rimozione di un'attività già cancellata dal database*

ni tra i pazienti e quella attività, indipendentemente dal giorno, ora e luogo definiti. Come già detto, per la rimozione di un paziente è stato usato lo stesso approccio.

All'interno del menu principale *File* è possibile anche scegliere la rimozione di una delle attività assegnate ad un paziente. Come visto nel caso di assegnazione di un'attività, queste sono definite non solo dal loro nome, ma anche dal giorno, l'ora e il luogo in cui devono essere portare a termine.

Nella fase di *design* di questa interfaccia è stato allora deciso di introdurre un passaggio ulteriore rispetto a quelli visti in precedenza per la rimozione dei dati di un paziente o di un'attività.

Come rilevato nello *State Diagram* di figura 4.20, la prima scelta da effettuare è relativa al paziente (primo snapshot di figura 4.21). Quando questa selezione è stata fatta, si impedisce all'utente di tornare indietro e modificarla, a meno che non sia riselezionata questa opzione nel menu principale, come indicato nel messaggio visualizzato al top della pagina (secondo snapshot di figura 4.21). Questa condizione si ottiene fissando il menu a tendina sul valore precedentemente scelto, ed eliminando il bottone *Scelto*, che verrà sostituito da *Rimuovi*. Questa limitazione è stata introdotta per permettere la presentazione dell'elenco delle attività (con definizione anche del giorno, ora e luogo stabiliti) del solo paziente selezionato. Ogni paziente è infatti associato ad un set di attività variabili, e solo queste possono essere selezionate per la rimozione, facilitando la scelta dell'utente. Premendo *Rimuovi*, se

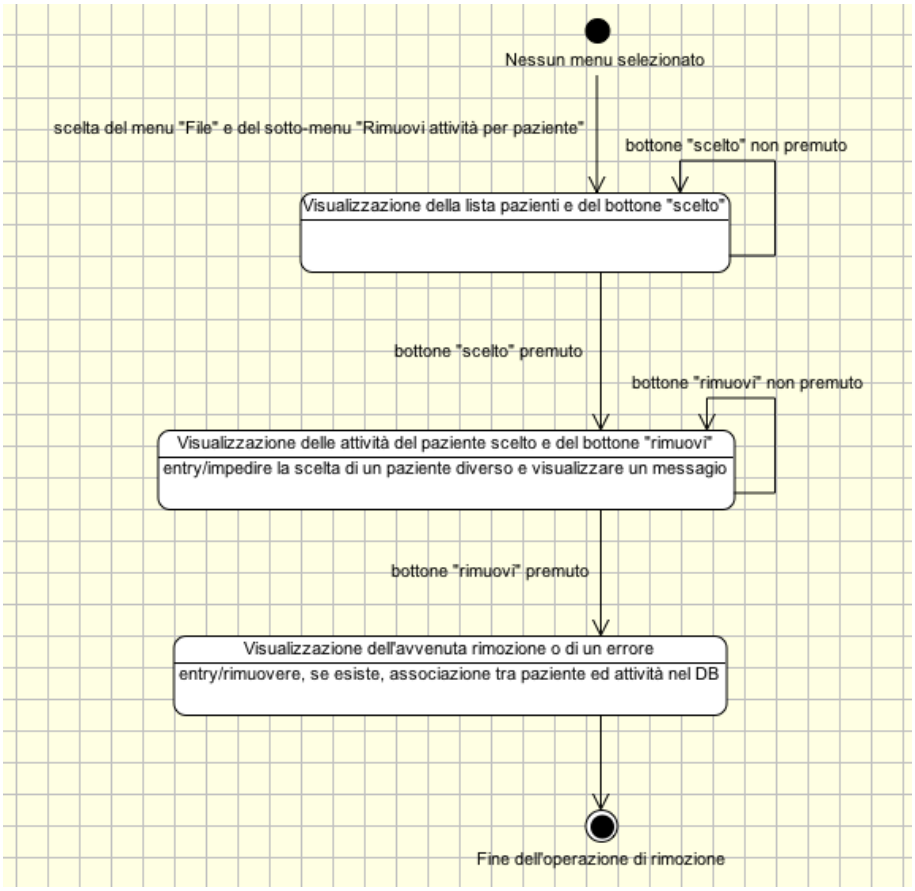

Figura 4.20: *UML State Diagram corrispondente alla rimozione di una o più attività assegnate al paziente*

l'attività esiste, viene eliminata dal database e viene visualizzato un messaggio di conferma (terzo snapshot di figura 4.21). In caso contrario compare un messaggio di errore, come già visto nei precedenti casi di eliminazione. Si ricorda a tale proposito, che la presenza di un'attività già rimossa nel menu a tendina può verificarsi solo se si rimane all'interno della stessa interfaccia per due rimozioni successive.

L'ultimo sotto-menu di *File* è *Esci da CareMate*, che semplicemente chiude l'applicazione.

Il menu *Pazienti* contiene 5 sotto-menu, due dei quali possono essere ulteriormente espansi. La figura 4.22 mostra il primo di questi: *Elenco, ordinato per*, che permette la visualizzazione dei pazienti in ordine di *id* (l'identificatore che li contraddistingue in modo univoco) o alfabetico secondo il *cognome*. Sotto al menu aperto, si può osservare parte della visualizzazione corrispondente all'elenco dei pazienti in ordine di *id*. La semplicità dello *State Diagram* (figura 4.23) rispecchia

Figura 4.21: *Snapshot delle fasi di rimozione di una delle attività assegnate ad un paziente.*

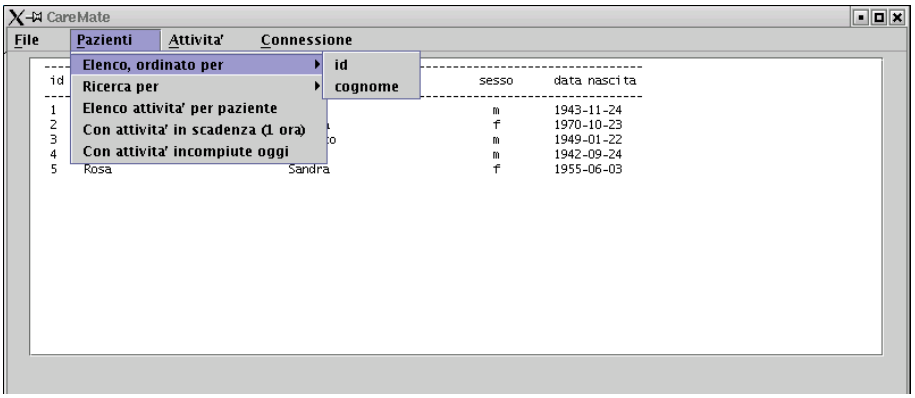

Figura 4.22: *Snapshot del menu "Pazienti" con il sotto-menu "Elenco, ordinato per", e visualizzazione dell'elenco dei pazienti (in parte coperto dal menu).*

la presenza di una sola schermata di interfaccia.

Il sotto-menu successivo è *Ricerca per*, che permette di effettuare la ricerca secondo quattro possibili chiavi: *cognome*, *sesso*, *data di nascita* e *attività assegnata*.

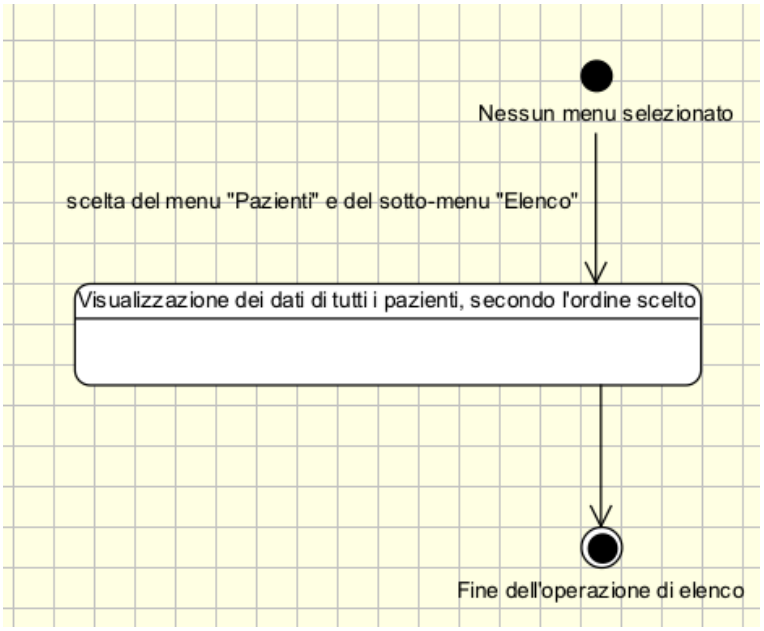

Figura 4.23: *UML State Diagram dell'operazione di visualizzazione dell'elenco dei pazienti secondo l'ordine specificato*

Scegliendo una delle prime tre, viene presentato un campo di input in cui l'utente può digitare la chiave. Scegliendo invece come chiave l'*attività assegnata* viene fatto scegliere in un menu a tendina tra i possibili valori delle attività attualmente presenti nel sistema.

Le figure 4.24 e 4.25 mostrano rispettivamente lo *State Diagram* ed lo snapshot dell'interfaccia finale della ricerca appena descritta. Nell'elenco può essere visualizzato un solo paziente (come è probabile accada in una ricerca per *data di nascita*), più pazienti (come nel caso di una ricerca di tutti i pazienti maschi), oppure un messaggio con cui l'utente viene informato del risultato negativo della ricerca stessa.

Un'utile funzionalità è quella di visualizzare tutte le attività di un certo paziente. Questa operazione viene svolta scegliendo il menu *Elenco attività per paziente*, il cui *State Diagram* viene presentato in figura 4.26.

Seguendo lo schema fornito da questo diagramma, si osserva che la prima interfaccia presentata all'utente è costituita dalla lista dei pazienti del sistema. Scegliendo il nominativo del paziente di cui si vogliono elencare le attività, e premendo il bottone *Elenca attività*, viene presentata la schermata di attività richieste o un messaggio per indicare che il paziente scelto non ha attività assegnate.

La figura 4.27 mostra due snapshot consecutivi dell'interfaccia presentata all'utente.

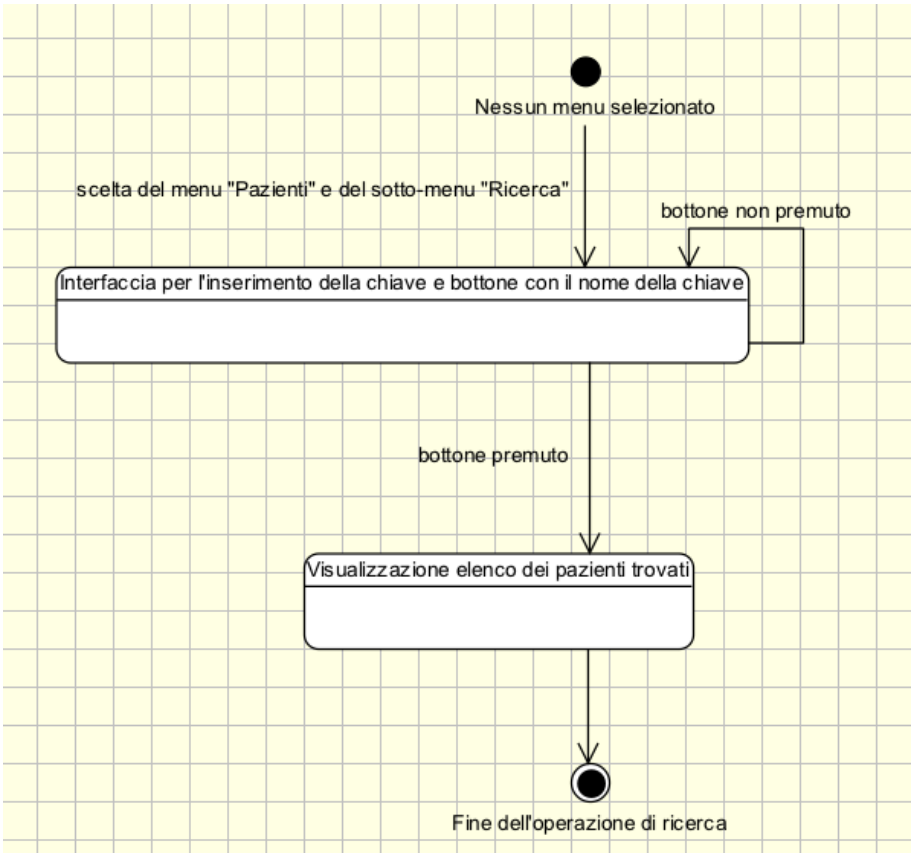

Figura 4.24: *UML State Diagram dell'interfaccia scelta dal sotto-menu "Ricerca per" specificando la chiave di ricerca*

Gli ultimi due sotto-menu di *Pazienti* sono *Con attività in scadenza (1ora)* e *Con attività incompiute oggi*. Dato che questi due menu presentano la stessa interfaccia (con dati diversi), sono stati riassunti in un solo *State Diagram* (figura 4.28). La scelta di uno dei due sotto-menu porta direttamente alla visualizzazione dei dati richiesti, senza interfacce intermedie. Questa caratteristica viene messa in evidenza nello *State Diagram*, formato da un solo stato semplice. La figura 4.29 mostra inoltre il messaggio di uscita visualizzato nel caso in cui non siano presenti nel sistema attività con scadenza entro 1 ora.

Proseguendo nell'analisi delle funzionalità fornite dal programma, si incontra il menu *Attività*, che permette di visualizzare l'elenco di tutte le attività del sistema, ordinate per *id* o in ordine alfabetico secondo il *nome* (figura 4.30). Lo *State Diagram* dell'interfaccia presentata da questo sotto-menu è mostrato in figura 4.31.

L'ultimo menu selezionabile è *Connessione*, che permette di inviare via Bluetooth il profilo di un paziente scelto. In questo modo si inizializza un *Device* prima

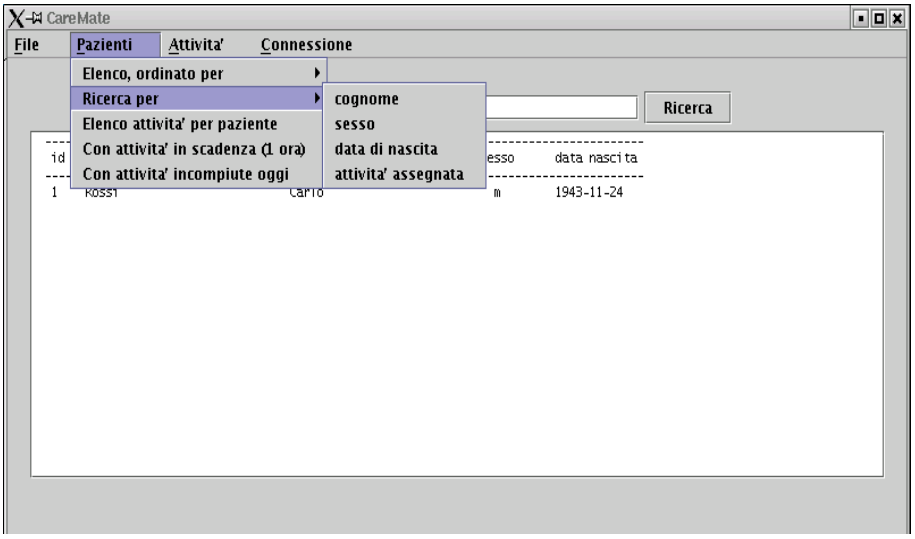

Figura 4.25: *Snapshot della scelta del sotto-menu "Ricerca per " nel menu "Pazienti", e della visualizzazione dei dati del paziente cercato.*

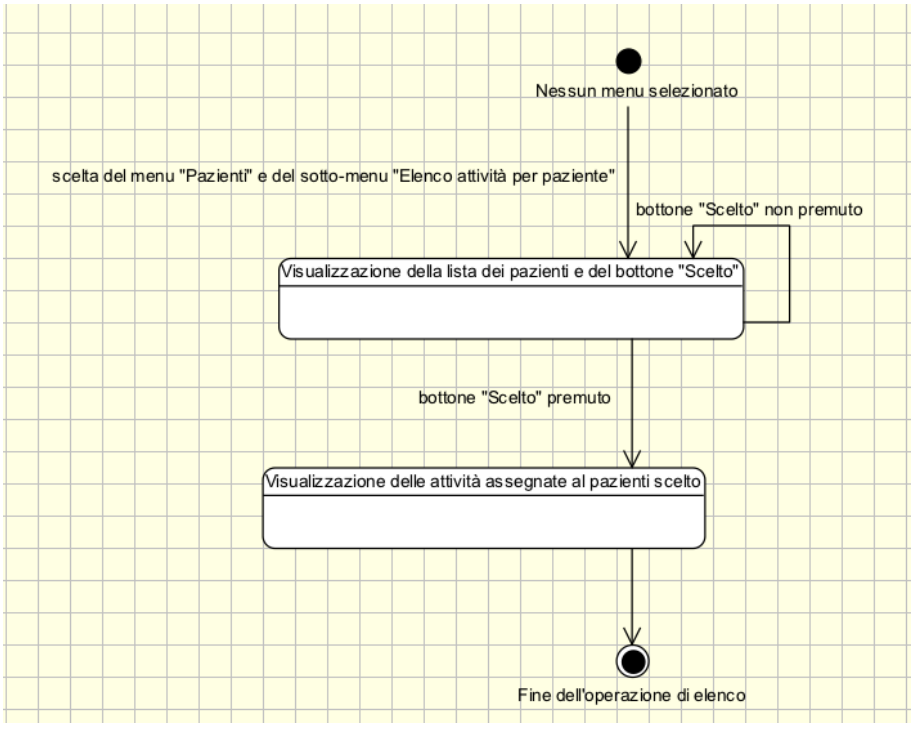

Figura 4.26: *UML State Diagram dell'interfaccia fornita dal sotto-menu "Elenco attività per paziente"*

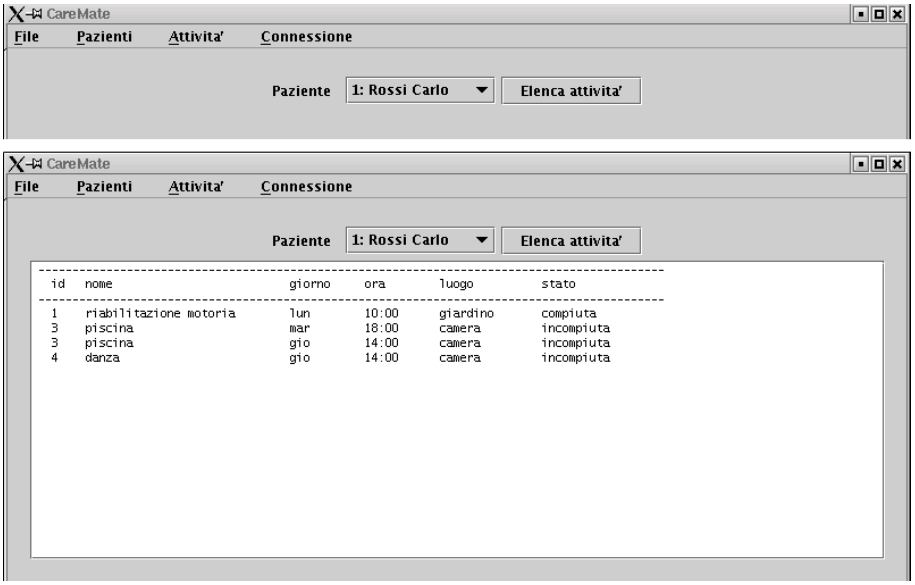

Figura 4.27: *Snapshot consecutivi di scelta del paziente e visualizzazione di tutte le sue attività nel sotto-menu "Elenco attività per paziente".*

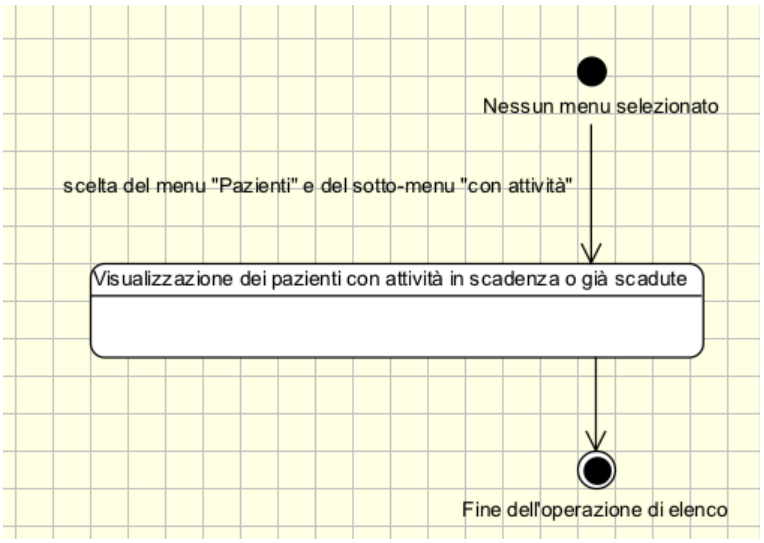

Figura 4.28: *UML State Diagram dell'interfaccia presentata sia dal sotto-menu "Con attività in scadenza (1 ora)" che dal sotto-menu "Con attività incompiute oggi".*

di assegnarlo ad un paziente.

Per evidenziare il fatto che la trasmissione dei dati avviene via Bluetooth, nel menu è stata anche introdotta l'icona con cui si identificano comunemente le comu-

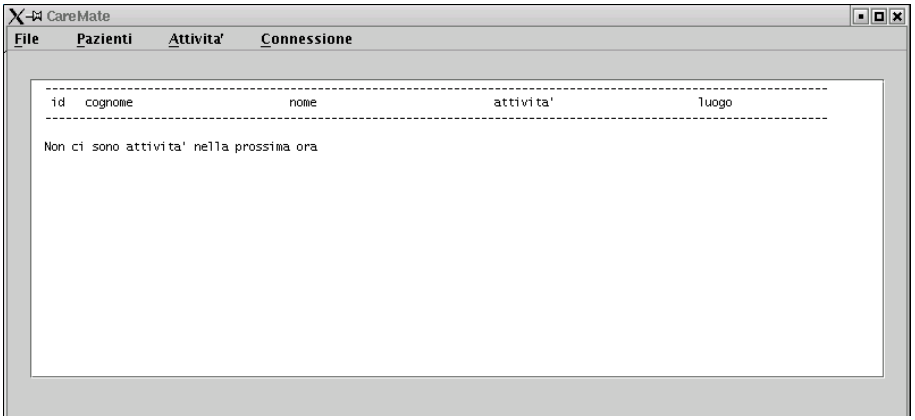

Figura 4.29: *Snapshot dell'interfaccia di visualizzazione delle attività in scadenza entro 1 ora, nel caso in cui nessuna soddisfi questa condizione.*

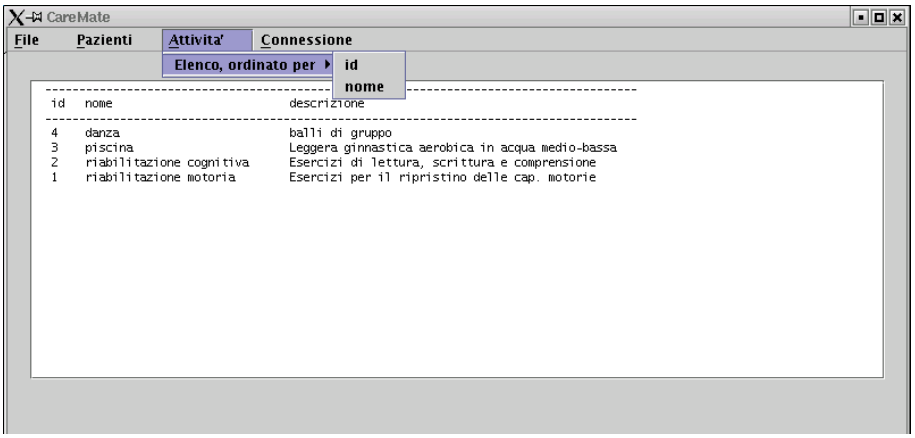

Figura 4.30: *Snapshot della scelta del sotto-menu "Elenco, ordinato per nome" e della relativa visualizzazione in ordine alfabetico*

nicazione che utilizzano questo standard (figura 4.32).

La figura 4.33 mostra lo *State Diagram* del menu *Connessione*. Alla scelta del menu *Connessione* viene presentata la lista dei pazienti inseriti nel sistema (figura 4.34). Alla scelta di un paziente (ovvero dopo aver premuto il bottone *Scelto*), viene creato il file `caremate.xml` con tutte le informazioni relative al paziente scelto, ovvero i suoi dati personali e quelli sulle attività terapeutiche che deve svolgere. Nel file non viene memorizzato lo stato delle attività, perché sono considerate tutte "incompiute". Questa informazione sottintesa non riduce la comprensione dei dati da parte del *Device* ed è coerente con l'intenzione di minimizzare le dimensioni del file inviato.

L'utilizzo del formato `xml` presenta notevoli benefici:

- È facilmente leggibile e comprensibile;
- Permette un'elevata portabilità.
- Consente un'ottima scalabilità del sistema (il numero delle attività elencate non ha un limite prefissato).
- Esistono numerosi programmi per effettuare il *parsing* in modo semplice e trasparente per il programmatore.
- A seconda dello stile ad esso applicato (HTML, WAP, VRML) può essere visualizzato in molti modi diversi.

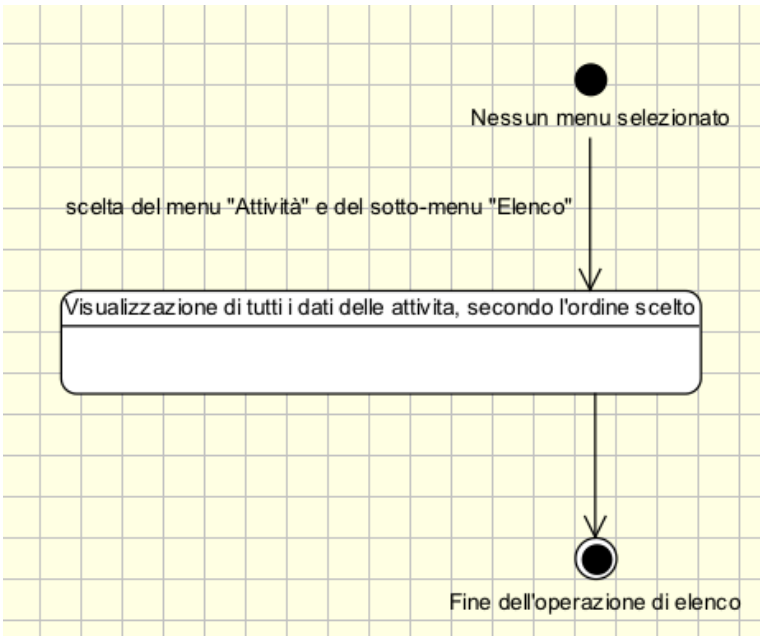

Figura 4.31: *UML State Diagram dell'interfaccia presentata dal sotto-menu "Elenco, ordinato per" per la visualizzazione delle attività del sistema*

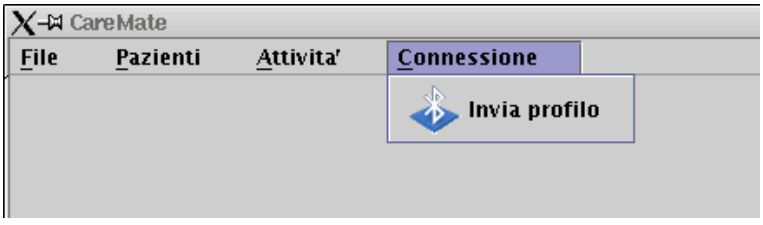

Figura 4.32: *Snapshot del menu "Connessione"*

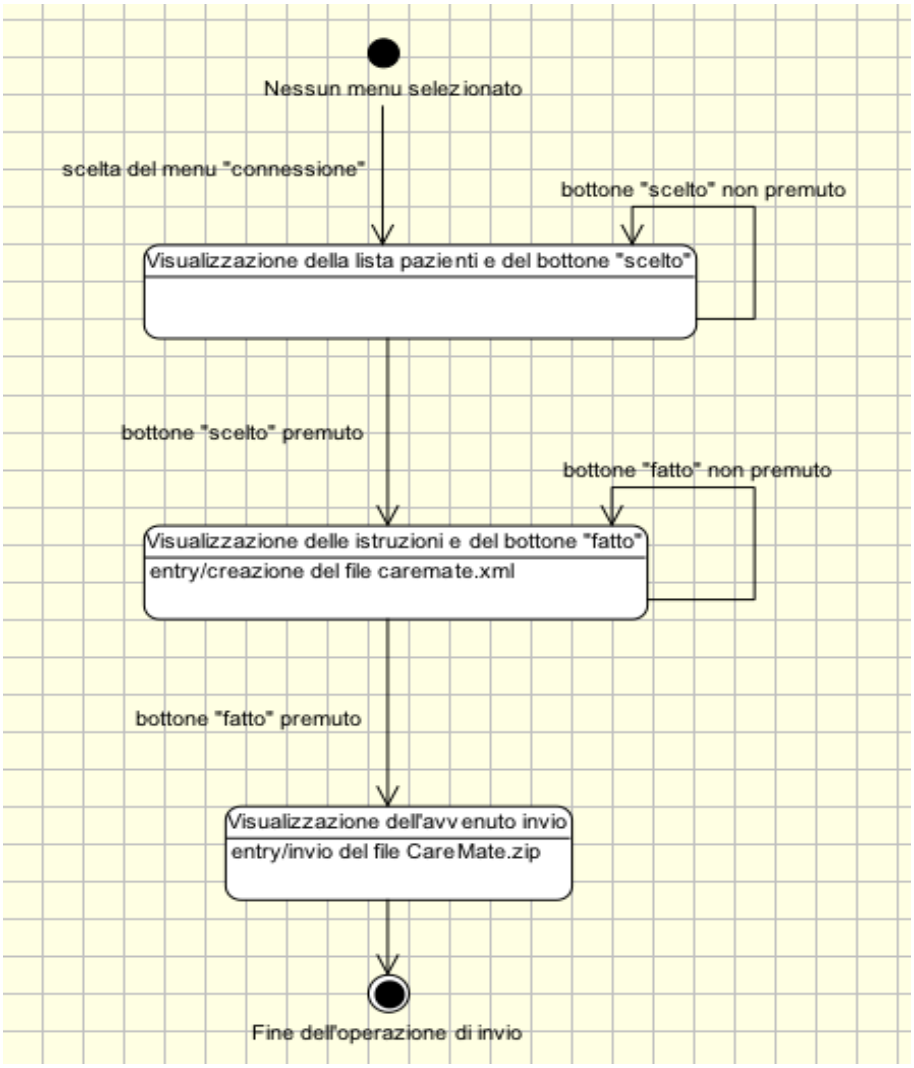

Figura 4.33: *UML State Diagram dello stato dell'interfaccia dopo la scelta del menu "Connessione"*

Figura 4.34: *Snapshot della visualizzazione della lista dei pazienti*

Queste caratteristiche fanno di `xml` lo strumento più adatto per lo scambio di informazioni tra piattaforme diverse. Un esempio del file `caremate.xml` creato per un ipotetico paziente "Carlo Rossi" potrebbe essere quello mostrato nel listato 4.3.

Al termine della creazione del file viene visualizzato un messaggio per confermare l'avvenuta scrittura del profilo e per dare all'utente le istruzioni necessarie per proseguire (figura 4.35).

```
<?xml version="1.0"?>
<paziente>
        <idP>1</idP>
        <cognP>Rossi</cognP>
        <nomeP>Carlo</nomeP>
                < attivita  value="0">
                        <nomeA>swimming</nomeA>
                        <giornoA>lun</giornoA>
                        <oraA>10:00</oraA>
                </ attivita >
                < attivita  value="1">
                        <nomeA>rehabilitation</nomeA>
                        <giornoA>mar</giornoA>
                        <oraA>09:00</oraA>
                </ attivita >
                < attivita  value="2">
                        <nomeA>swimming</nomeA>
                        <giornoA>mer</giornoA>
                        <oraA>12:00</oraA>
                </ attivita >
                < attivita  value="3">
                        <nomeA>rehabilitation</nomeA>
                        <giornoA>gio</giornoA>
                        <oraA>13:00</oraA>
                </ attivita >
</paziente>
```

Listato 4.3: *Esempio di un possibile file* `caremate.xml`

Le istruzioni indicate nella visualizzazione sono le seguenti:

```
cd /usr/java/CareMate/careMateMIDlet/build
./build
```

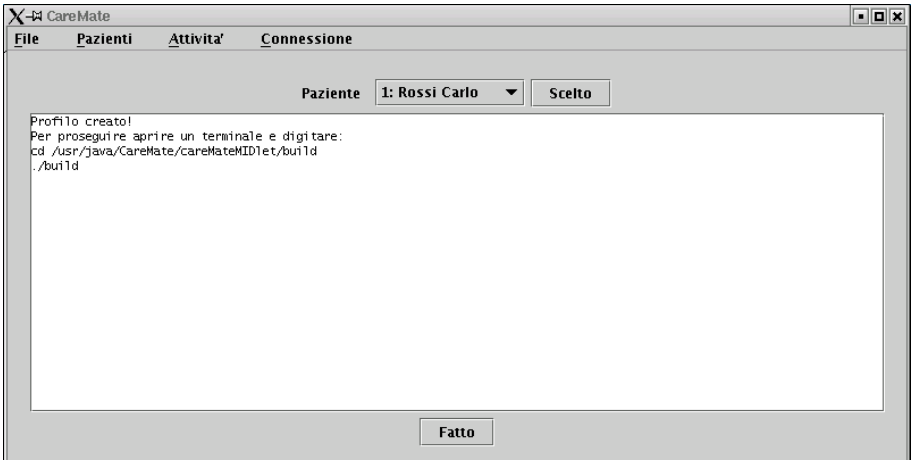

Figura 4.35: *Snapshot della visualizzazione delle istruzioni necessarie per proseguire*

Con la prima istruzione l'utente entra nella cartella che contiene il file `build` per compilare il programma (ovvero il midlet) da spedire al *Device* insieme al profilo `caremate.xml`, precedentemente salvato nella cartella delle risorse del midlet stesso.

Lo script `build` permette la creazione dei file eseguibili `CareMate.jar` e `CareMate.class`, necessari per l'esecuzione del midlet sul *Device*. Questi due file vengono infine *zippati* per generare un unico file `CareMate.zip`, che sarà quello effettivamente inviato al *Device*. Dopo aver seguito le istruzioni ed aver premuto il bottone *Fatto*, il file `CareMate.zip` viene effettivamente inviato. Terminato l'invio del file, viene visualizzato un messaggio di conferma e il sistema è pronto per effettuare un nuovo invio (figura 4.36).

A questo punto è importante mettere in evidenza che la creazione del file `zip` contenente sia `CareMate.jar` che `CareMate.class`, è necessaria solo perché il midlet nel nostro prototipo verrà eseguito su un computer portatile. Disponendo infatti di un cellulare, sarebbe stato sufficiente il file `CareMate.jar` per l'esecuzione del midlet.

In ogni modo, modificare questa parte del programma per inviare solo il file `jar` non implica la scrittura di nuovo codice: è sufficiente cambiare il nome del file inviato in `CareMate.jar`. Le modifiche apportate per eseguire il sistema nel prototipo realizzato, non limitano in alcun modo la sua esecuzione su qualsiasi altro dispositivo dotato di di piattaforma J2ME e di implementazione delle JABWT.

Figura 4.36: *Snapshot della conferma dell'avvenuto invio del profilo ed eventuale scelta di nuova connessione*

4.3.4 Comunicazione Bluetooth

La comunicazione Bluetooth è stata realizzata scrivendo la classe `OBEXClient` che invia un file utilizzando gli standard OBEX e Bluetooth. L'OBEX è un protocollo spesso affiancato al Bluetooth per fornire un meccanismo a livello piuttosto alto per l'invio di file. La comunicazione avviene quindi utilizzando il canale Bluetooth e il meccanismo di invio dei dati dell'OBEX.

Come mostrato nel *Class Diagram* di figura 4.37, la classe `OBEXClient` implementa l'interfaccia `DiscoveryListener` che permette di identificare dispositivi abilitati Bluetooth e ricevere la lista dei servizi che questi eventualmente forniscono.

Quando il *client* identifica un dispositivo che supporta un *server* OBEX (ovvero un *server* che ha *pubblicato* il servizio corrispondente al codice `0x1105` ed è in grado di accettare un file), chiama la classe `SenderClient` che si occupa dell'effettivo invio del file. `SenderClient` è definita all'interno della classe `OBEXClient` che la chiama.

`OBEXClient` segue un *pattern* ben definito per tutti i *client* Bluetooth, corrispondente alla funzionalità di *inquiry*, ovvero identificazione di eventuali dispositivi adiacenti e ricerca dei servizi disponibili. Fino a questo punto, non c'è alcuna differenza tra un *client* OBEX e, ad esempio un *client* L2CAP per la trasmissione di semplici messaggi. È quindi `SenderClient` che implementa le funzionalità dell'OBEX. In particolare, il cuore della classe `SenderClient` è mostrato nel listato 4.4.

Come prima cosa viene creato un *header* con il nome del file, da inviare prima del file stesso. L'applicazione destinataria potrà così accertarsi della coerenza tra il

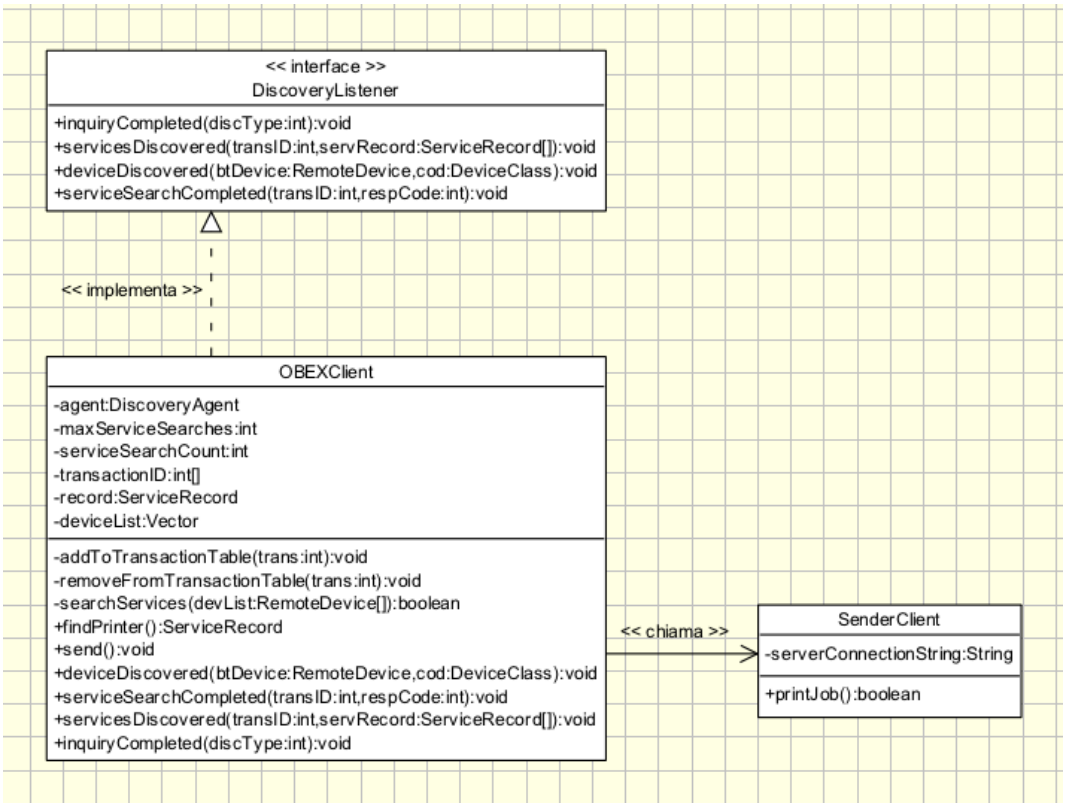

Figura 4.37: *UML Class Diagram della classe MysqlInterface.*

nome del file inviato e quello scritto nell'*header*.

Successivamente viene definita l'operazione che si vuole eseguire (in questo caso una *put*) e anche questo dato veine scritto all'interno dell'*header*. Dopo aver aperto il file di output su cui scrivere e quello di input da cui leggere, un semplice ciclo *while* permette l'invio dei dati (un byte alla volta). Un'eccezione viene lanciata se il file di ingresso non esiste (o più semplicemente non è stato inserito nella stessa cartella del file eseguibile `OBEXClient.class`).

A questo punto gli *stream* aperti possono essere chiusi e il *server* sconnesso.

Nella figura 4.38 viene mostrato uno snapshot dell'output scritto sul terminale dal *client* OBEX. Come descritto nella nell'analisi precedentemente effettuata, dopo aver identificato i dispositivi adiacenti, si effettua una ricerca sui servizi che questi forniscono. Se tra questi servizi è presente un *server* OBEX, allora il file viene effettivamente inviato al dispositivo che supporta questo servizio. Ovviamente dall'altra parte deve essere presente un *server* OBEX in grado di accettare il file inviato.

```
/* crea l'header da inviare */
HeaderSet head = conn.createHeaderSet ();

/* setta il nome del file da inviare */
head.setHeader(HeaderSet.NAME, "CareMate.zip");

/* definisce la richiesta di un'operazione put */
Operation op = conn.put(head);

/*apre lo stream di output per processare la richiesta */
DataOutputStream out = op.openDataOutputStream();

InputStream in = null;
try
{
        /*apre il file da inviare */
        in = new FileInputStream("CareMate.zip");

        /* legge dallo stream di input */
        /* e scrive sullo stream di output*/
        int data;
        while ((data = in.read()) != -1)
        {
                out.write((byte) data);
        }
}
catch(IOException e)
{
        /*eccezione in mancanza del file di input */
        System.out.println("FileIOException");
}

/* chiusura degli stream */
out.close();
in.close();
op.close();

/* disconnette il server */
conn.disconnect(head);
```

Listato 4.4: *Cuore della classe SenderClient*

La scrittura di *server* OBEX da eseguire sul portatile utilizzato come *Device*

non è stata però considerata utile ai fini della tesi, per i seguenti due motivi:

- Il pacchetto`javax.obex` non è implementato su alcun cellulare o dispositivo esistente che supporti la piattaforma J2ME (nemmeno su quelli che forniscono l'implementazione per le JABWT);
- D'altra parte, dato che in questi apparecchi la comunicazione Bluetooth viene sfruttata soprattutto per lo scambio di file, è sempre presente un *server* OBEX fornito dalla ditta costruttrice.

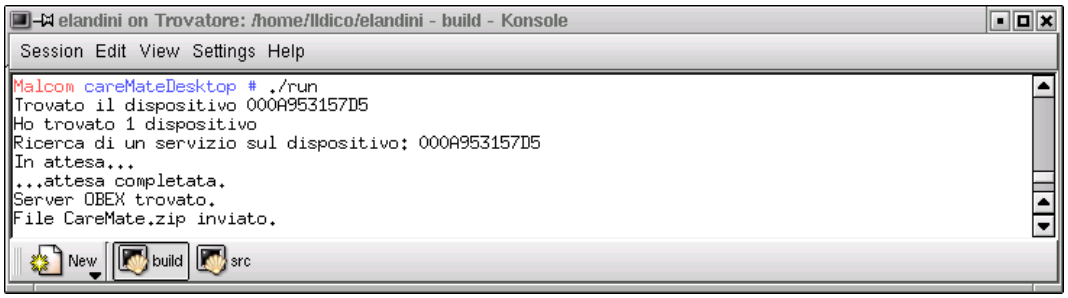

Figura 4.38: *Output scritto sul terminale all'invio del file* `CareMate.zip`.

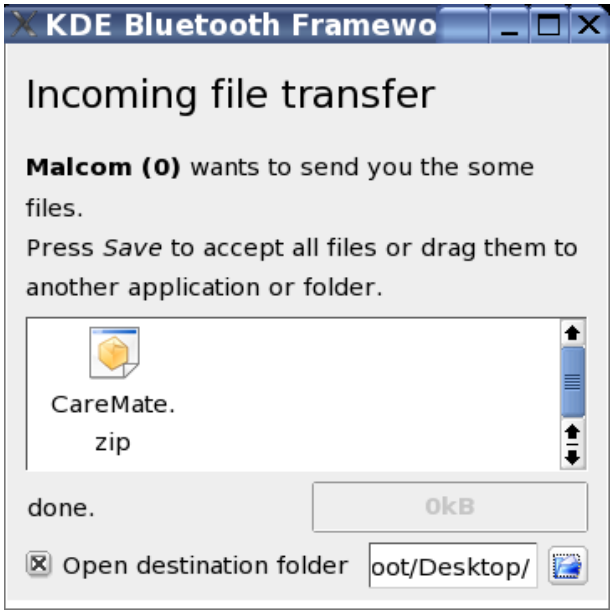

Figura 4.39: *Finestra di KDEBletooth che appare alla ricezione del file* `CareMate.zip`.

Per questo motivo per il prototipo realizzato, si è preferito utilizzare gli strumenti OBEX forniti da KDEBluetooth [51]. La figura 4.39 mostra la finestra di KDEBletooth che appare alla ricezione del file `CareMate.zip`.

Per evidenziare l'interoperabilità dell'applicazione sviluppata, la figura 4.40 mostra la stessa finestra in ambiente Mac. In questo caso non è stato necessario installare un *server* OBEX (come fatto su Linux utilizzando KDEBluetooth) perché il sistema operativo MacOSX supporta già questa funzionalità. Questo significa che il *client* OBEX realizzato per questa tesi può inviare file ad un qualsiasi dispositivo Bluetooth che implementa un *server* OBEX.

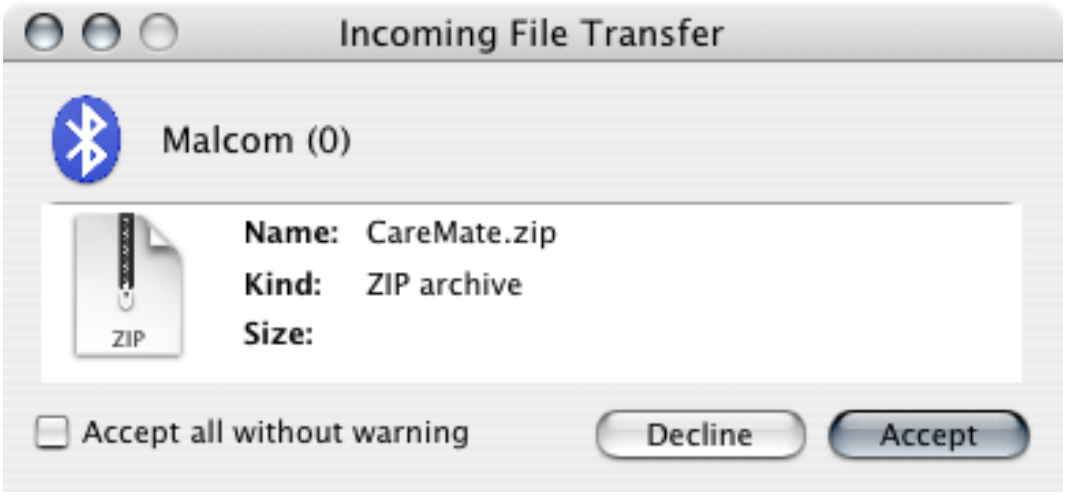

Figura 4.40: *Finestra che appare alla ricezione del file `CareMate.zip` nell'ambiente MacOSX.*

4.4 Device

Come prototipo per dimostrare il buon funzionamento del sistema è stato scelto un portatile non Centrino privo di connessione WI-FI per accertare l'effettiva comunicazione via Bluetooth. È stata inoltre mantenuta l'intenzione di scrivere un'applicazione per il prototipo *ora* sviluppato, che sia però anche in grado *in futuro* di essere eseguita su un dispositivo più adatto alla realizzazione concreta di un sistema di assistenza ad anziani e disabili. In questo senso, la scelta di eseguirlo su un computer portatile non limita in alcun modo la sua effettiva funzionalità su qualsiasi altro dispositivo dotato di piattaforma J2ME (nella configurazione più compatta, ovvero CLDC con profilo MIDP) e di implementazione delle JABWT.

Date le numerosi parti che compongono la struttura del *Device*, l'analisi è stata suddivisa in quattro parti:

- Analisi delle classi chiamate dal midlet per eseguire il parsing del profilo del paziente, la memorizzazione permanente dei dati e la gestione di eventuali ritardi nell'esecuzione delle attività assegnate.
- Analisi delle classi chiamate dal midlet per realizzare la comunicazione Bluetooth e per aggiornare il database con le informazioni ricevute.
- Descrizione del codice per la realizzazione dell'interfaccia minima.
- Descrizione del comportamento complessivo del *Device* nell'interazione con gli altri elementi del sistema CareMate.

4.4.1 Parsing, memorizzazione permanente e *timer*

Il profilo del paziente è memorizzato nel file `caremate.xml` inserito all'interno dell'eseguibile `jar` caricato sul *Device*. Dato che in `caremate.xml` le informazioni sono formattate in `xml`, per poterle estrarre è necessario effettuare un *parsing* del file stesso. Questa operazione viene svolta dalla classe `RSSParser` in concomitanza con l'interfaccia `RSSListener` implementata da `CareMateMIDlet` (come mostrato nel *Class Diagram* di figura 4.41). `CareMateMIDlet` implementa quindi i due metodi di `RSSListener`: `exception`, che ha lo scopo di visualizzare a video un eventuale eccezione relativa al *parsing*, e `itemParsed`, che effettua l'elaborazione dei dati estratti dal file `caremate.xml`.

Dopo aver aperto `caremate.xml` utilizzando il metodo `getInputStream` della classe `IOUtils`, il metodo `parse` della classe `RSSParser` effettua la lettura dei dati contenuti in `caremate.xml` in modo semplice ed intuitivo. Il listato 4.5 mostra infatti la struttura regolare dell'estrazione dei valori realizzata con le librerie di *kXml* [49]. I metodi `getName`, `getType` e `getValue` della classe `ParseEvent` consento di identificare rispettivamente il nome, il tipo e il valore all'interno del *tag* correntemente analizzato, mentre il metodo `getText` permette di assegnare il nome dell'attività ad una variabile di tipo `String`. Il metodo `read` della classe `XmlParser` viene invece utilizzato per leggere il *tag* successivo. I valori così estratti vengono passati al metodo `itemParsed` dell'interfaccia `RSSListener` chiamato al termine dell'estrazione dei dati di ogni singola attività, ed implementato all'interno di `CareMateMIDlet`.

Sui dati personali del paziente non viene effettuato il *parsing*: queste informazioni infatti non devono essere modificate dal *Device*, e devono essere invece inviate all'*Embedded* e al *Nomad* ancora in formato `xml`. Per evitare quindi un'inutile conversione, vengono semplicemente lette dal file `caremate.xml` quando richieste.

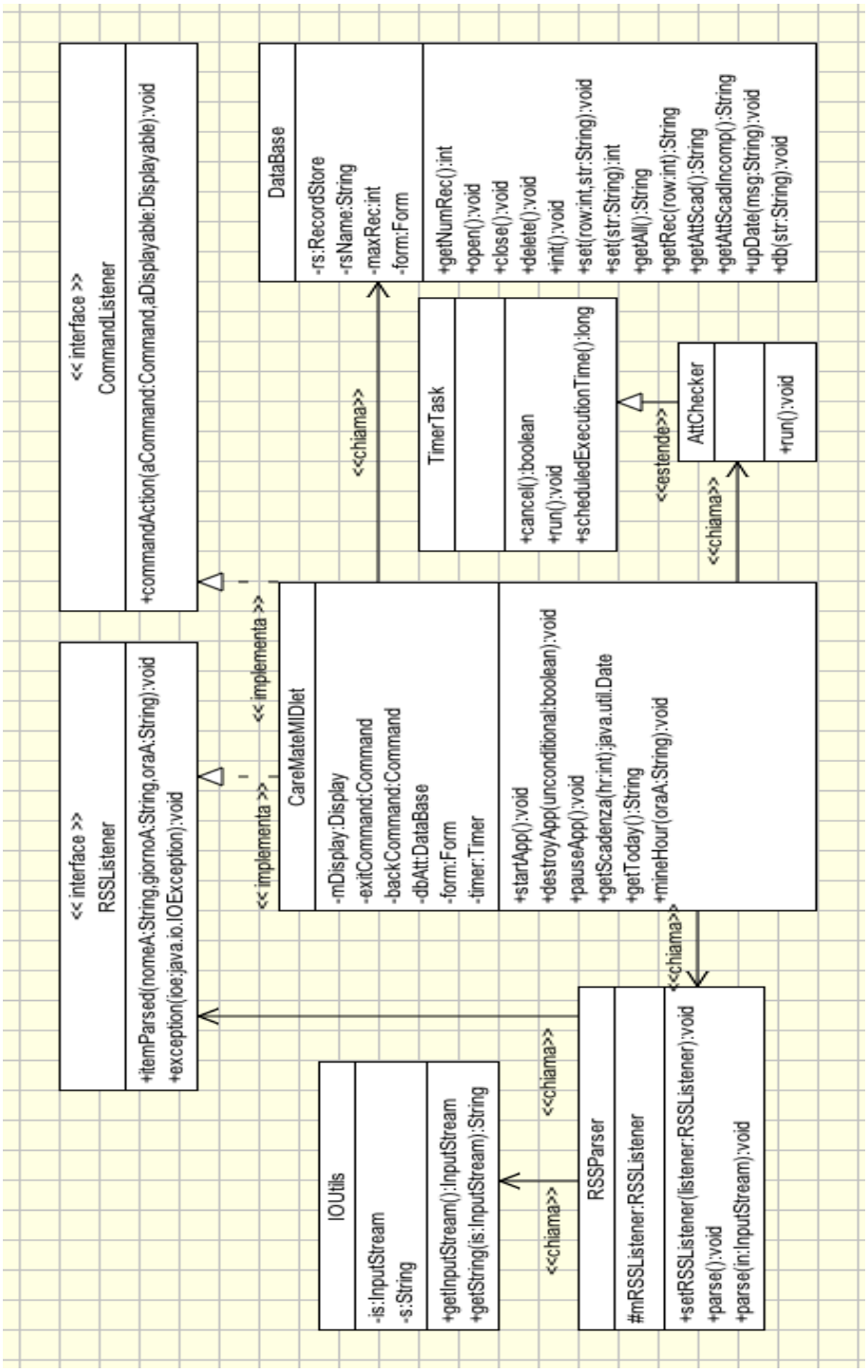

Figura 4.41: *UML Class Diagram del Device (senza la parte di comunicazione)*

```
ParseEvent pe = null;

if (( pe.getType() == Xml.START_TAG)
    && (pe.getName().equals(" attivita ") == true ))
{
   String name2 = pe.getName();
   row = pe.getValue("value");
   while (!(( pe.getType() == Xml.END_TAG)
         && (pe.getName().equals(name2) == true )))
   {
          pe = parser.read();
          if (pe.getType() == Xml.START_TAG && pe.getName().equals("nomeA"))
          {
              pe = parser.read();
              nomeA = pe.getText();
          }
          else
     if (pe.getType() == Xml.START_TAG && pe.getName().equals("giornoA"))
     {
              pe = parser.read();
              giornoA = pe.getText();
     }
     else
              if (pe.getType() == Xml.START_TAG && pe.getName().equals("oraA"))
              {
                       pe = parser.read();
                       oraA = pe.getText();
              }
   }
   mRSSListener.itemParsed(nomeA, giornoA, oraA);
   pe = parser.read();
}
```

Listato 4.5: *Parte del metodo* `parse` *della classe* `RSSParser` *per l'estrazione dei dati di un'attività.*

```
RSSParser parser = new RSSParser();
parser.setRSSListener(this);
parser.parse();
```

Listato 4.6: *Chiamata del parser* `RSSParser` *all'interno del metodo* `startApp` *di* `CareMateMIDlet`.

Nel midlet `CareMateMIDlet` , il *parser* viene chiamato all'interno del metodo `startApp` eseguito all'avvio del programma. In particolare il listato 4.6 evidenzia la creazione di un *parser* di tipo `RSSParser`, l'assegnazione del *listener* `RSSListener` e la chiamata al metodo `parse` che realizza effettivamente il *parsing*.

I dati estratti vengono utilizzati dall'applicazione per determinare eventuali anomalie nel comportamento del paziente. Per la memorizzazione sono stati utilizzati gli strumenti forniti dal pacchetto `javax.microedition.rms`, ed è stata creata la classe `DataBase` (mostrata nel *Class Diagram* di figura 4.41) che contiene tutti i metodi necessari per gestire il database `dbAtt` delle attività.

Date le ridotte capacità di memoria dei dispositivi considerati per lo sviluppo di un *Device* reale, è stato scelto di memorizzare in modo permanente solo i dati delle attività del giorno corrente. Il listato 4.7 mostra come questo avviene all'interno del metodo `itemParsed` implementato dalla classe `CareMateMIDlet`. Le variabili `nomeA`, `giornoA` e `oraA` corrispondono rispettivamente al nome, il giorno e l'ora dell'attività estratta nel *parsing*.

```
public void itemParsed(String nomeA, String giornoA, String oraA)
{

        String today = getToday();
        String att = "";
        if (giornoA.equals(today))
        {
                att = "\nincompiuta␣" + nomeA + "␣" + oraA;
                dbAtt.open();
                int row = dbAtt.set( att );
                dbAtt.close();

                // estae l'ora dalla stringa oraA
                int hour = mineHour(oraA);

                AttChecker checker = new AttChecker(dbAtt, row, nomeA, oraA);
                timer.schedule(checker, getScadenza(hour));
        }
}
```

Listato 4.7: *Metodo* `itemParsed` *di* `CareMateMIDlet`.

Prima di tutto viene confrontato `giornoA` con il giorno corrente (ottenuto con la chiamata al metodo `getToday` del midlet). Se i due valori coincidono, significa

che i dati dell'attività considerata devono essere salvati in modo permanente nel database `dbAtt` di tipo `DataBase` definito all'interno di `CareMateMIDlet`.

Dato che il meccanismo di memorizzazione fornito dal pacchetto `rms` consente di salvare solo oggetti di tipo `String`, viene creata per ogni attività quotidiana una stringa del tipo:

incompiuta nomeAttività ora

È importante notare che tutte le attività vengono inizialmente salvate con stato "incompiuta", e non compare alcun riferimento al giorno, dato che vengono salvate solo quelle del giorno corrente. Alcuni esempi delle stringhe salvate nel database permanente, possono essere i seguenti:

incompiuta rehabilitation 17:00
incompiuta swimming 10:00
incompiuta gym 14:00

Come mostrato nel listato 4.7, al termine della memorizzazione dei dati nel database permanente, vengono attivati dei *timer* corrispondenti agli orari in cui le attività devono essere eseguite. Questa operazione corrisponde alla creazione di un oggetto `checker` della classe `AttChecker` mostrata nel *Class Diagram* di figura 4.41. In particolare, a partire dal valore di `oraA`, il timer viene settato mezz'ora dopo, per considerare gli spostamenti del paziente.

Quando uno di questi `timer` scade, l'attività corrispondente viene settata come "scaduta" nel database `dbAtt`, ripetendo un procedimento analogo a quello utilizzato per la memorizzazione iniziale mostrata nella prima parte del listato 4.7. Rifacendosi al caso illustrato poco sopra, se l'attività "swimming" non viene portata a termine, il database `dbAtt` verrà aggiornato nel seguente modo:

incompiuta rehabilitation 17:00
scaduta swimming 10:00
incompiuta gym 14:00

4.4.2 Comunicazione Bluetooth ed aggiornamento dati

Le classi sviluppate per realizzare la comunicazione con l'*Embedded* e il *Nomad* sono mostrate nel *Class Diagram* di figura 4.42. Data la stretta analogia tra il *client* addetto all'invio di informazioni verso l'*Embedded* e quello che svolge la stessa

azione in direzione del *Nomad*, per motivi di spazio nel *Class Diagram* è stato rappresentato solo `ClientToEmbedded`. Come già visto per `OBEXClient` descritto nella sezione 4.3.4, anche `ClientToEmbedded` e `ClientToNomad` implementano l'interfaccia `DiscoveryListener` per effettuare la ricerca di eventuali dispositivi Bluetooth adiacenti e dei loro servizi, e fanno uso della classe `Server` per la fase concreta di invio dei dati. Questi due *client* vengono chiamati dal *server* Bluetooth lanciato nel metodo `startApp` di `CareMateMIDlet`.

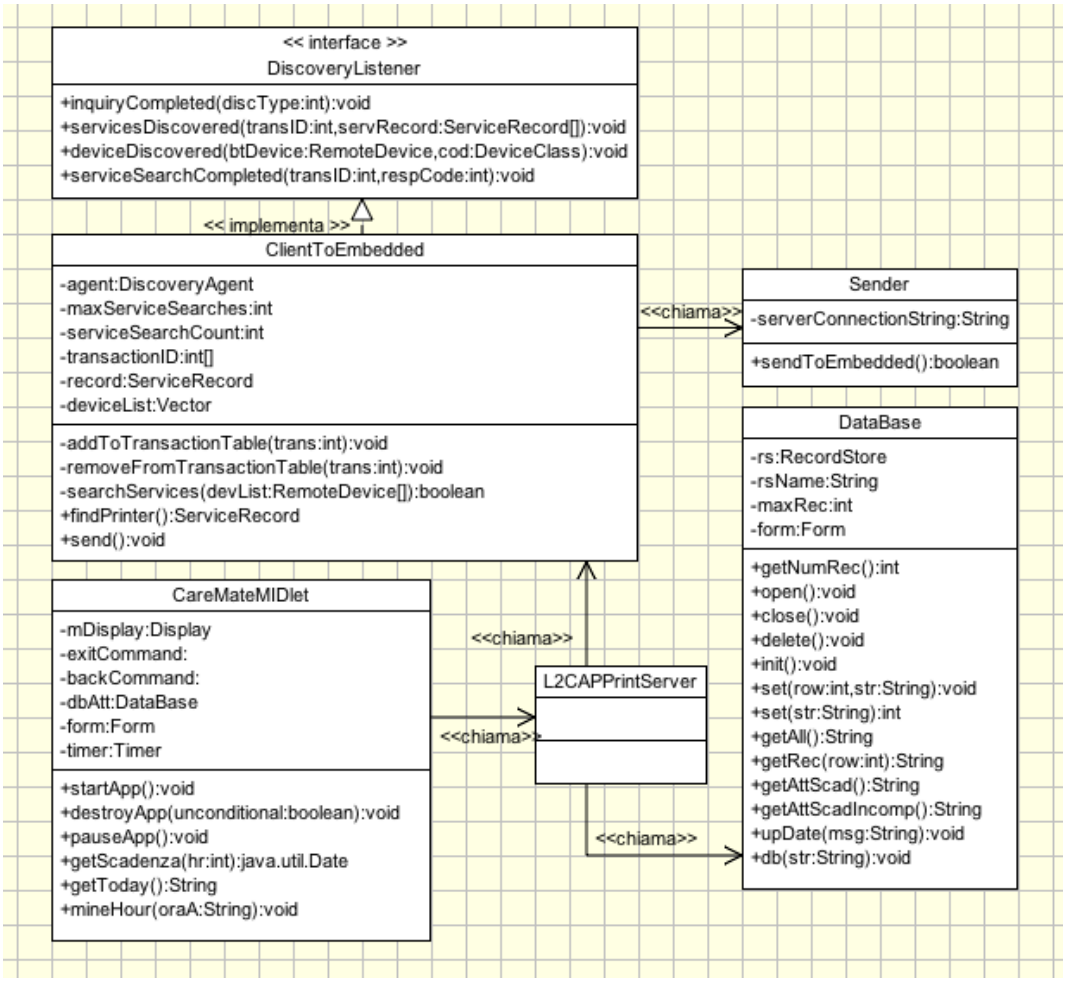

Figura 4.42: *UML Class Diagram della parte di comunicazione Bluetooth del Device (solo verso l'Embedded)*

Questo *server* supporta una comunicazione L2CAP e pubblica un servizio indicato dal codice `3333`. Dato che i *server* dell'*Embedded* e del *Nomad* forniscono lo stesso servizio (ovvero la semplice ricezione di un messaggio di testo), è stato necessario distinguerli rispettivamente con il codice `4444` e `5555`, in modo che

la comunicazione non subisca interferenze indesiderate. Per fare un esempio, se il *Device* vuole inviare un messaggio all'*Embedded* effettuerà una ricerca del servizio corrispondente al codice `4444`. L'*Embedded* a sua volta, per indirizzare la comunicazione verso il *Device* cercherà il servizio `3333`.

Il *server* rimane in attesa finché non viene contattato da un *client* che ne richiede il servizio. Dato che il servizio in questione corrisponde alla ricezione di un messaggio, dopo aver accettato la richiesta del *client* ed aver aperto la connessione, il *server* si appresta all'acquisizione del messaggio inviato. A ricezione avvenuta, viene controllato se il messaggio verifica una delle quattro seguenti condizioni e viene applicata l'azione corrispondente:

- *Il messaggio è* `embedded`: in questo caso vengono estratti i dati delle attività ancora da compiere (sia con stato "incompiuta" che "scaduta"), formattati in `xml` ed aggiunti a quelli del pazienti. Viene quindi creato un *client* di tipo `ClientToEmbedded` che ricerca il servizio identificato dal codice `4444` e se lo trova invia il messaggio `xml` precedentemente creato.

- *Il messaggio è* `nomad`: al verificarsi di questa condizione viene ripetuto il procedimento già visto nel caso precedente. Le uniche due differenze dipendono dal fatto che vengono estratte e formattate in `xml` solo le attività con stato "scaduta", e viene lanciato il *client* `ClientToNomad`, che ricerca il servizio corrispondente al codice `5555`.

- *Il messaggio contiene la parola* `error`: significa che l'*Embedded* ha inviato un messaggio di errore in risposta al profilo spedito dal *Device*. Viene allora visualizzato nel *display* del *Device* un corrispondente messaggio indicante la ragione dell'errore e l'attività relativa all'*Embedded* che l'ha inviato.

- *Il messaggio contiene la parola* `ok`: in quest'ultimo caso viene aggiornato ad "ok" lo stato dell'attività corrispondente nel database `dbAtt`.

Al termine di una delle operazioni associate a queste quattro condizioni, o nel caso in cui nessuna di esse sia verificata, il *server* torna in attesa di una richiesta da parte dell'*Embedded* o nel *Nomad*.

4.4.3 Interfaccia minima

Il *Device* dispone di un'interfaccia grafica rivolta all"utente (che in questo contesto è l'infermiere, non il paziente) per visualizzare eventuali anomalie. Dato che il sistema è finalizzato ad essere eseguito su un dispositivo al limite *wearable* (quindi senza alcuna interfaccia) è stata creata un'interfaccia minima, che in un futuro dispositivo reale potrebbe semplicemente essere sostituita da un *led* lampeggiante in caso di emergenza.

Come mostrato nel listato 4.8, la creazione dell'interfaccia è tutta contenuta all'interno del costruttore di `CareMateMIDlet`. I messaggi *Caremate* e *Sistema di comunicazione wireless per l'integrazione di robot di servizio in architetture domotiche* sono inseriti nel `Form` visualizzato nel `Display` del *Device*. Oltre a questo messaggio, viene inserito il comando *exit* che determina la chiusura dell'applicazione. Le azioni eseguite su questo comando sono gestite dal metodo `commandAction` definito nell'interfaccia `CommandListener` implementata direttamente da `CareMateMIDlet` (come mostrato nel *Class Diagram* di figura 4.41).

```
public  CareMateMIDlet()
{
        form = new Form("CareMate");
        form.append(new StringItem( null ,  "Sistema
                        di  comunicazione  wireless  per
                        l ' integrazione  di  robot  di  servizio  in
                         architetture  domotiche" ));

        exitCommand = new Command("Exit", Command.EXIT, 0);
        form.addCommand(exitCommand);
        form.setCommandListener(this );

        mDisplay = Display . getDisplay ( this );
        mDisplay. setCurrent (form);
}
```

Listato 4.8: *Creazione dell'interfaccia del Device*

L'interfaccia di base visualizzata sull'emulatore Nokia per la Serie 60 è mostrata nella figura 4.43. In questo caso sono presenti diversi messaggi visualizzati, corrispondenti alle specifiche del cellulare emulato: nella figura 4.43 a) è presente il nome dell'applicazione da scegliere, nella b) la richiesta di consenso alla connessione Bluetooth e nella c) viene definitivamente mostrata l'interfaccia che permarrà per tutta l'esecuzione dell'applicazione.

Come già messo in evidenza, l'emulatore non consente la comunicazione Bluetooth con dispositivi esterni reali (ovvero l'*Embedded* e il *Nomad*). Quindi il prototipo è stato testato utilizzando *Me4Se* che fornisce la semplice interfaccia mostrata nella figura 4.44. In particolare la figura 4.44 b) mostra la visualizzazione di un messaggio di errore corrispondente al tentativo di effettuare un'attività terapeutica diversa da quelle assegnate. Questa interfaccia è visualizzata quando viene ricevuto un messaggio di errore dall'*Embedded*, e rimane visibile sullo schermo in modo che gli operatori competenti possano prenderne visione.

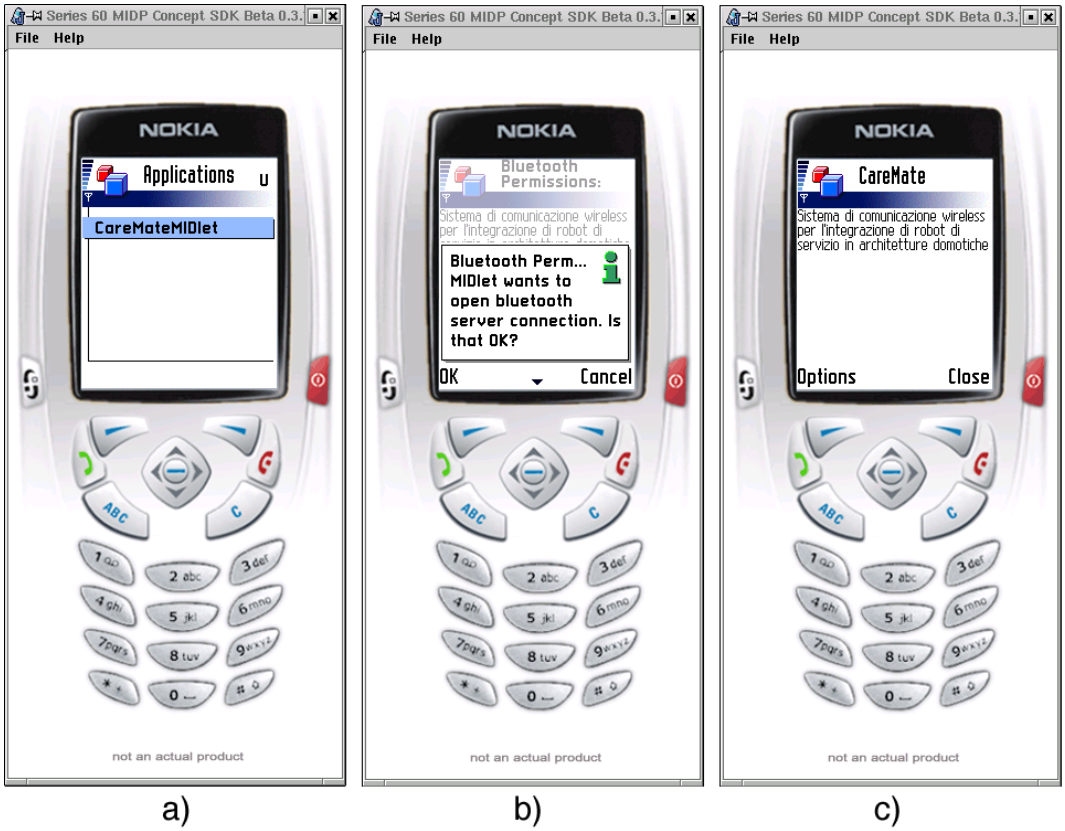

Figura 4.43: *Snapshot dell'emulatore Nokia Serie 60 a) Interfaccia iniziale del midlet. b) Richiesta di connesione via Bluetooth. c) Interfaccia di base del midlet in esecuzione*

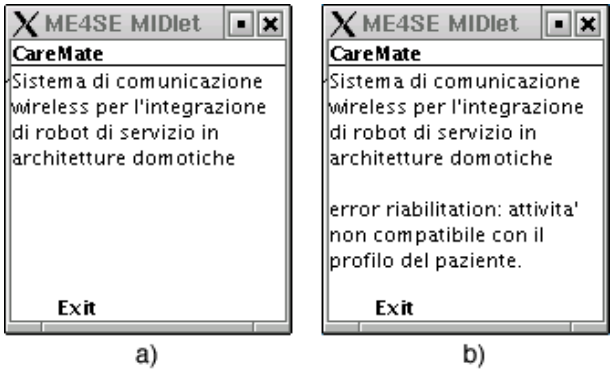

Figura 4.44: *Snapshot dell'interfaccia presentata da Me4Se a) Interfaccia iniziale del midlet. b) Visualizzazione di un errore nel comportamento del paziente.*

4.4.4 Comportamento complessivo del *Device*

Facendo riferimento al *Sequence Diagram* di figura 4.3 mostrato nella sezione 4.2 di questo capitolo, si può analizzare in dettaglio il comportamento del *Device* nelle diverse fasi di interazione con gli altri elementi del sistema CareMate.

Il *Device* viene creato alla ricezione del file inviato dal *DataBase*. Come si vede nella figura 4.45 appena il midlet viene eseguito inizializza il database dei profili ed effettua il *parsing* dei dati memorizzati nel file `caremate.xml`, relativi al profilo del paziente. Quando il *parsing* è terminato, viene chiamato il *server* Bluetooth, che da questo momento rimarrà in ascolto di eventuali richieste.

Nello scenario che ha generato l'output della figura 4.45, la prima richiesta viene ricevuta da parte del *Nomad*, che invia come riconoscimento la stringa `nomad`. Alla ricezione di questo messaggio, il *Device* accede al database `dbAtt` in cui sono memorizzate tutte le attività quotidiane ed estrae (se ve ne sono) quelle con stato "scaduta". Formatta quindi in `xml` e aggiunge queste informazioni a quelle relative ai dati personali del paziente, sempre in formato `xml`. Dopo aver creato il messaggio da inviare, si mette al ricerca del *Nomad*. Infatti, in un ambiente dinamico come quello qui considerato, il *Nomad* potrebbe non essere più raggiungibile, e questa condizione deve essere tenuta in considerazione nella progettazione del sistema.

Se il *Nomad* è ancora adiacente, il *Device* invia la risposta alla sua richiesta, identificata nella figura 4.45 con l'etichetta `answer`. Come si può notare, il messaggio di risposta nel caso esemplificato contiene il profilo del paziente "Carl Jones" a cui corrisponde nel giorno corrente una sola attività "scaduta", ovvero "rehabilitation" che doveva essere eseguita alle ore 10.

Dopo l'invio della risposta al *Nomad*, il *Device* torna in attesa di una richiesta, che questa seconda volta arriva da parte dell'*Embedded*. Il *Device* accede allora nuovamente al database delle attività quotidiane ed estrae sia quelle scadute che quelle incompiute. Nello scenario utilizzato, il messaggio di risposta inviato all'*Embedded* è equivalente a quello per il *Nomad* (il che significa che non sono presenti attività con stato "incompiuta").

Come mostrato nella figura 4.46, dopo l'invio della risposta per l'*Embedded*, il *Device* si mette in attesa di riceve un nuovo messaggio. Nel caso illustrato, riceve la stringa `ok` (dall'*Embedded*). Questo messaggio significa che l'attività che il paziente sta per compiere è compatibile con il suo profilo, e può quindi essere aggiornata allo stato "ok" (il che indica che è stata completata). Ricevuta questa conferma, il *Device* aggiorna lo stato dell'attività corrispondente e torna in attesa di una richiesta.

A questo punto, se il *Nomad* richiede informazioni sulle attività "scadute", il *Device* invierà la risposta mostrata nella figura 4.46, in cui non compare più alcuna attività (perché quella precedente è stata ora aggiornata). Lo stesso messaggio viene inviato anche all'*Embedded* in risposta alla sua richiesta. È però importante mettere

Figura 4.45: *Prima parte dell'output del Device (continua)*

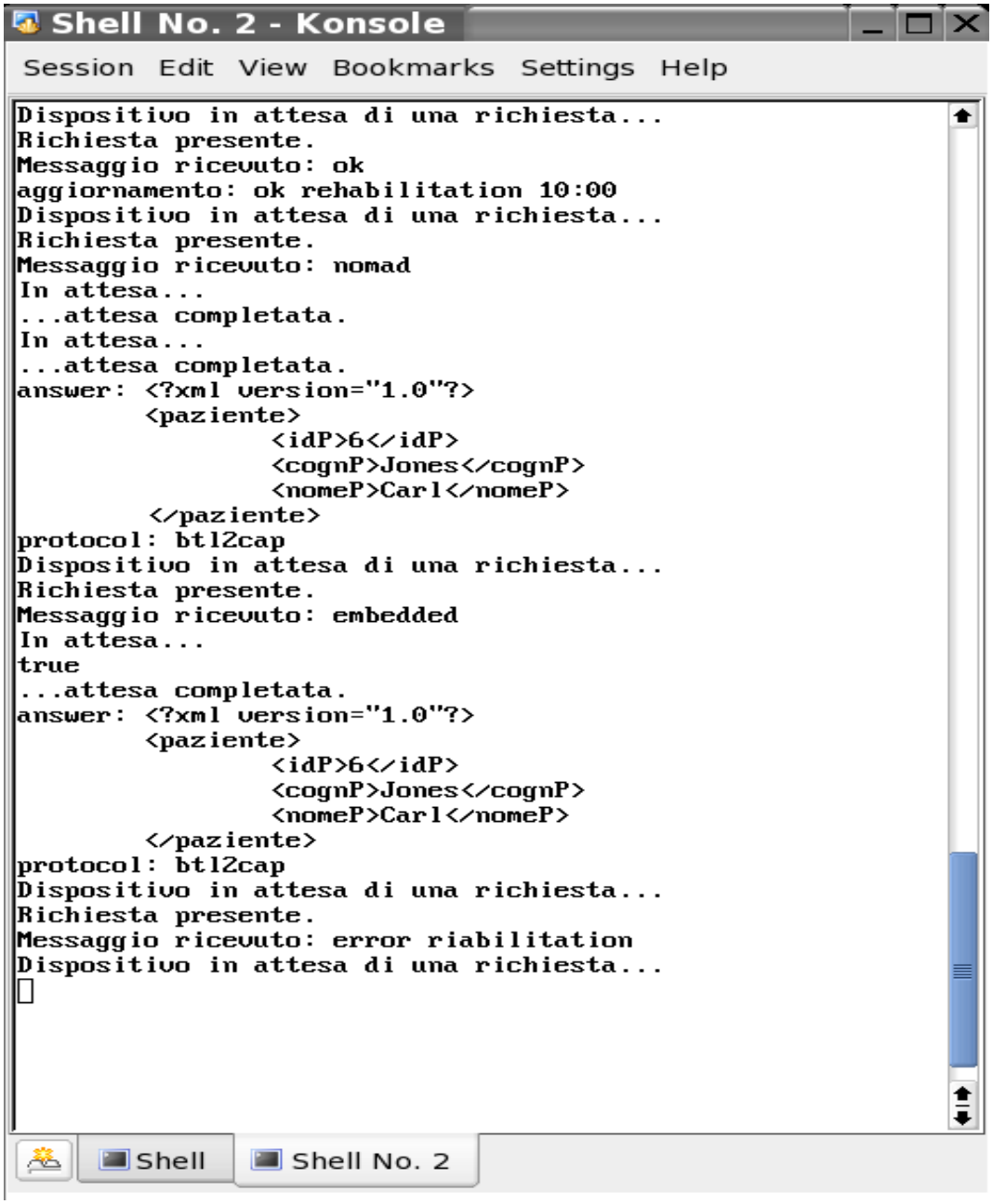

Figura 4.46: *Seconda parte dell'output del Device.*

in evidenza il diverso significato che questo stesso messaggio assume per il *Nomad* e per l'*Embedded*. Per il *Nomad* infatti significa che il profilo del paziente ora è normale e che non ci sono attività scadute: viene quindi interpretato positivamente. Per l'*Embedded* invece significa che il paziente sta tentando di effettuare una attività

a cui non è abilitato, altrimenti i dati dell'attività in questione comparirebbero nel suo profilo. In questo caso viene quindi interpretata come una situazione anomala, e per questo motivo l'*Embedded* invia al *Device* un messaggio di errore. Come visto nella descrizione dell'interfaccia del *Device*, questo messaggio viene utilizzato per visualizzare sullo schermo un messaggio di *allerta* per gli operatori competenti.

Come mostrato nell'ultima riga dell'output (figura 4.46), al termine di ogni comunicazione, il *Device* si rimette sempre in condizione di ascolto, per identificare un'eventuale richiesta di invio di informazioni.

Lo scenario così descritto copre ogni caso possibile e mette in evidenza tutti i diversi comportamenti del *Device*.

4.5 Nomad

La struttura delle classi che compongono il *Nomad* è mostrata nel *Class Diagram* di figura 4.47, in cui è immediato osservare che sia il *server* che il *client* implementano l'interfaccia `DiscoveryListener`.

Il *Nomad* deve infatti continuamente identificare la presenza di possibili *Device* a cui richiedere il profilo dei pazienti a cui sono associati. Per far questo viene eseguito il *server* che attraverso l'oggetto `agent` di tipo `DiscoveryAgent` effettua la ricerca di dispositivi Bluetooth adiacenti. Lo *State Diagram* di figura 4.48 mostra le fasi successive attraversate dal *server*.

Nel caso in cui il *server* identifichi la presenza di uno o più dispositivi Bluetooth, passa alla ricerca del servizio con codice `3333`, corrispondente a quello pubblicato dai *server* di tutti i *Device*. Se il servizio non viene trovato, torna allo stato iniziale e comincia una nuova ricerca. Nel caso in cui invece il dispositivo fornisca il servizio richiesto, il *server* procede alla verifica del protocollo utilizzato.

Tutte le comunicazione tra *Device*, *Nomad* ed *Embedded* avvengono infatti tramite `l2cap`, ovvero il protocollo a più basso livello per l'invio e la ricezione di messaggi di testo. La scelta di questo protocollo piuttosto che `rfcomm`, è stata dettata dalla volontà di minimizzare, anche sotto questo aspetto, la struttura delle applicazioni realizzate (in particolare dalla parte del *Device*).

Se anche la verifica che protocollo `l2cap` va a buon fine, si passa all'invio del messaggio `nomad`. Per far questo, viene creato un *client* che gestisce l'invio utilizzando la classe di supporto `Sender`, in cui il messaggio viene effettivamente spedito. L'interazione ed i legami tra queste classi sono mostrati nel *Class Diagram* di figura 4.47.

Al termine dell'invio del messaggio `nomad`, il server si mette in attesa di ricevere una risposta dal *Device* identificato. Pubblica quindi il servizio con codice `5555` per poter essere contattato (e non essere confuso con gli *Embedded* che pubblicano i loro servizi con il codice `4444`). Rimane poi in attesa finché il *client* del *Device*

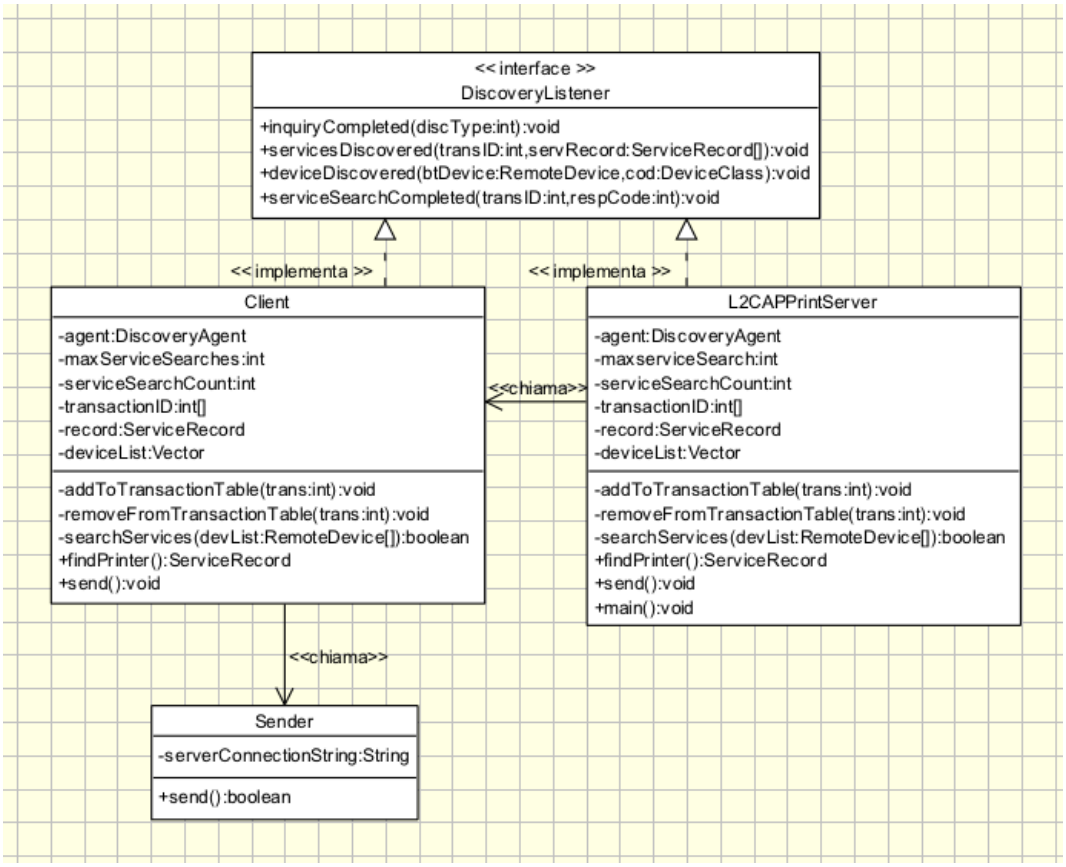

Figura 4.47: *UML Class Diagram rappresentante la struttura delle classi utilizzate per la comunicazione tra il Nomad e il Device*

non invia la risposta con il profilo (contenente i dati personali del paziente e delle eventuali attività scadute).

Alla ricezione dei dati, ne elabora il significato e utilizza il sintetizzatore sonoro per confermare al paziente la sua buona condotta, o indicargli le attività che non ha eseguito nei tempi stabiliti.

Terminata anche questa operazione, il *server* torna nello stato iniziale e comincia una nuova ricerca di dispositivi Bluetooth.

Per evidenziare l'interazione tra il *Nomad* ed il *Device* è utile considerare il *Sequence Diagram* di figura 4.49 e l'output scritto dal *Nomad* sul terminale (figura 4.50 e 4.51).

In particolare la figura 4.50 mostra l'output nel caso in cui il *Nomad* identifichi un paziente con un'attività non eseguita nell'orario prestabilito. Il profilo inviato dal *Device* contiene infatti l'indicazione all'attività "rehabilitation" da compiere alle 10, con stato "scaduta". Ricevendo un profilo di questo tipo, il compito del *Nomad* è

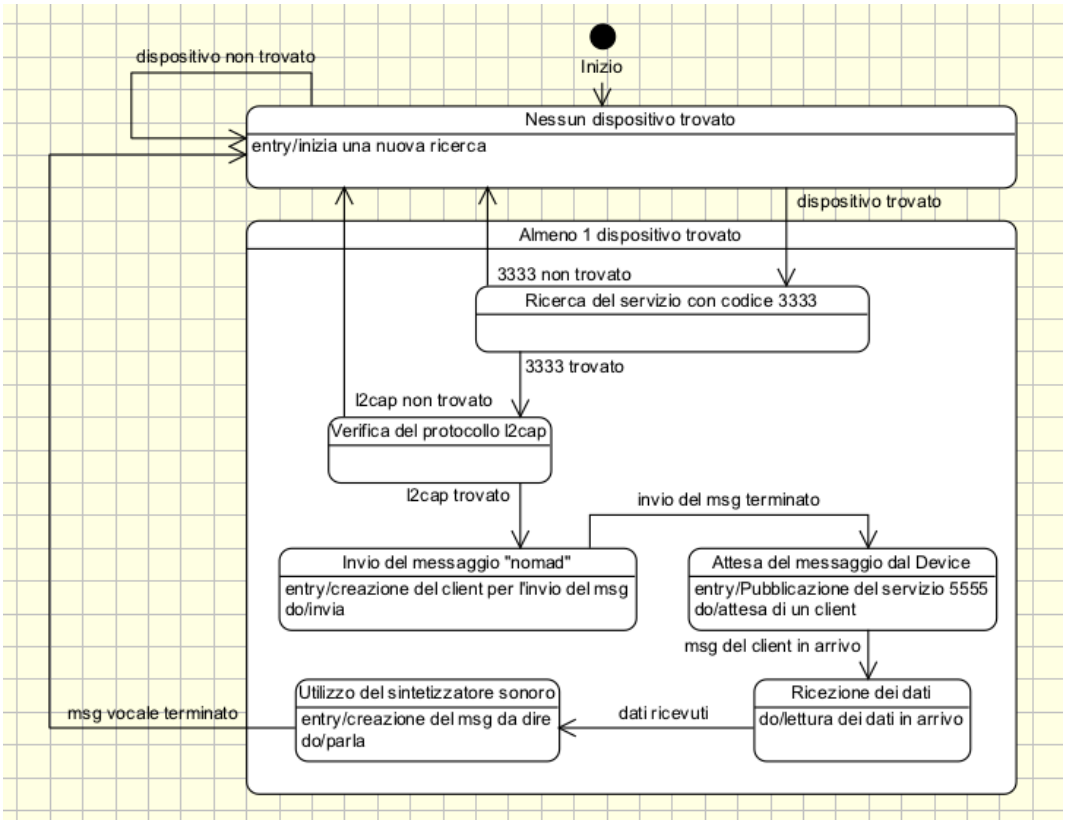

Figura 4.48: *UML State Diagram rappresentate il comportamento del server Bluetooth del Nomad*

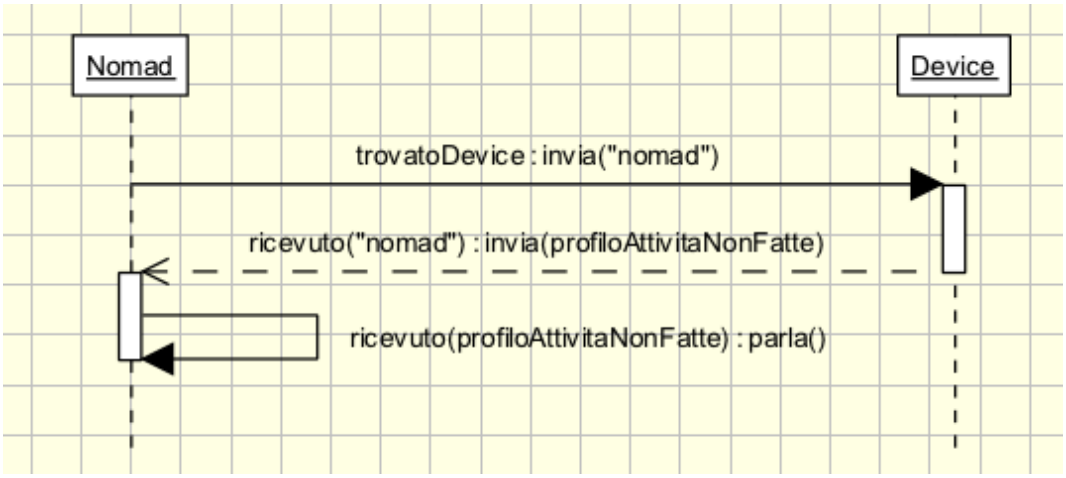

Figura 4.49: *UML Sequence Diagram della comunicazione tra il Nomad e il Device*

quello di avvertire il paziente dell'errato comportamento. Crea quindi un messaggio con il cognome del paziente ("Mr Jones", in questo caso) e dell'attività non svolta, ed utilizza il sintetizzatore vocale per comunicare al paziente il seguente messaggio:

> "Mr Jones, you have not done your rehabilitation session."
> ("Mr Jones, non hai effettuato la tua sessione di riabilitazione")

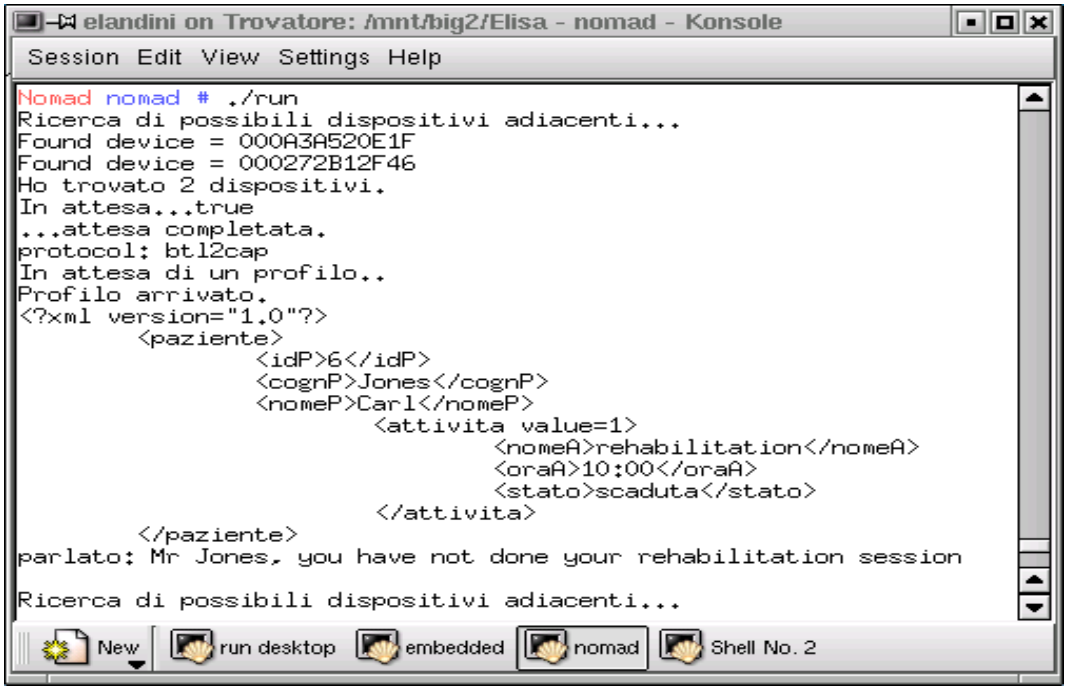

Figura 4.50: *Output scritto sul terminale dal Nomad nel caso di un'attività terapeutica non portata a termine.*

Nel caso in cui invece nel profilo ricevuto non siano presenti attività scadute, allora il *Nomad* lo informa del suo buon comportamento con il messaggio

> "Mr Jones, you are a good guy."
> ("Mr Jones, ti sei comportato bene.")

Dopo aver comunicato al paziente una delle due informazioni descritte, torna alla ricerca di nuovi *Device* da cui ricevere il profilo del pazienti a cui sono associati.

Infine, è importante evidenziare che in futuro il *Nomad* sarà interamente scritto in C++ per mantenere un buon grado di coerenza con l'architettura *real-time* su cui è stato sviluppato il sistema di controllo del robot mobile.

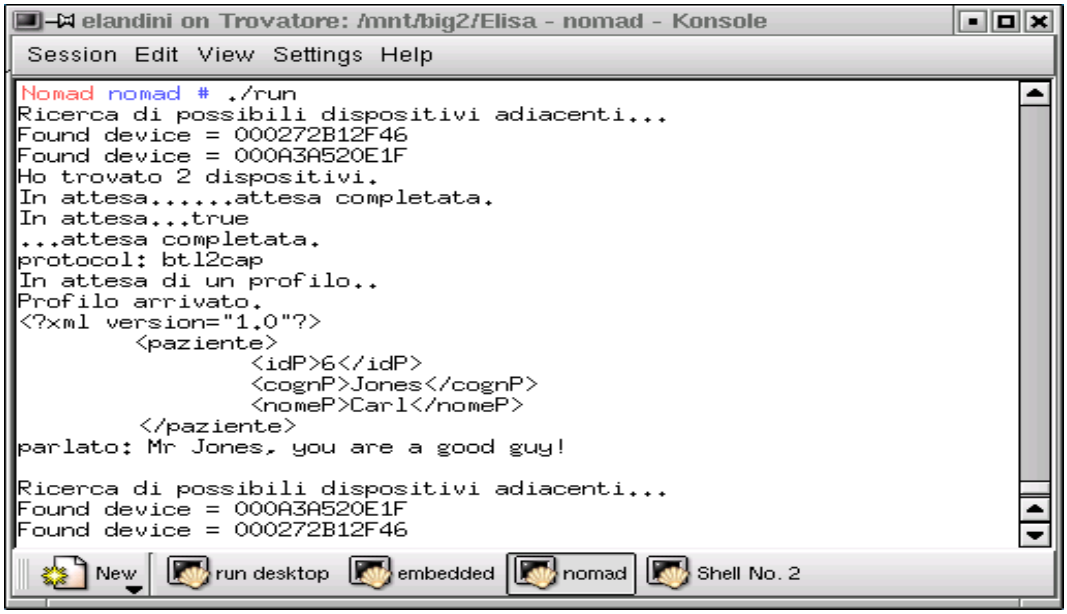

Figura 4.51: *Output scritto sul terminale dal Nomad nel caso in cui tutte le attività terapeutiche da fare siano state effettivamente fatte.*

4.6 Embedded

Gli *Embedded* sono associati a determinati ambienti ed all'attività che in questi ambienti viene svolta. Ad esempio può essere presente un *Embedded* all'ingresso della piscina ed uno all'entrata dell'ambulatorio. In questo sistema distribuito ogni *Embedded* è responsabile dell'aggiornamento del profilo del paziente che sta per effettuare l'attività a cui l'*Embedded* stesso è associato.

La struttura delle classi che compongono l'*Embedded* è uguale a quella del *Nomad* e quindi è mostrata nel *Class Diagram* di figura 4.47. Anche in questo caso infatti sia il *server* che il *client* implementano l'interfaccia `DiscoveryListener`.

Anche lo *State Diagram* rappresentante il comportamento del server Bluetooth dell'*Embedded* ha molte analogie con quello del *Nomad*. Entrambi infatti ricercano iterativamente possibili dispositivi Bluetooth adiacenti, e se ne trovano controllano la presenza del servizio con codice `3333` che contraddistingue i *Device*. Nel caso anche questa condizione sia verificata e il protocollo di comunicazione usato sia `l2cap`, allora l'*Embedded* invia al *Device* il messaggio `embedded` e si mette in attesa di ricevere la risposta dal *Device*. Per poter essere identificato, il *server* pubblica il servizio con codice `4444` e rimane in ascolto di una richiesta di accesso al servizio stesso.

Quando il *client* del *Device* lo contatta e gli invia il profilo del paziente (con l'indicazione di tutte le attività ancora da compiere), l'*Embedded* controlla che queste

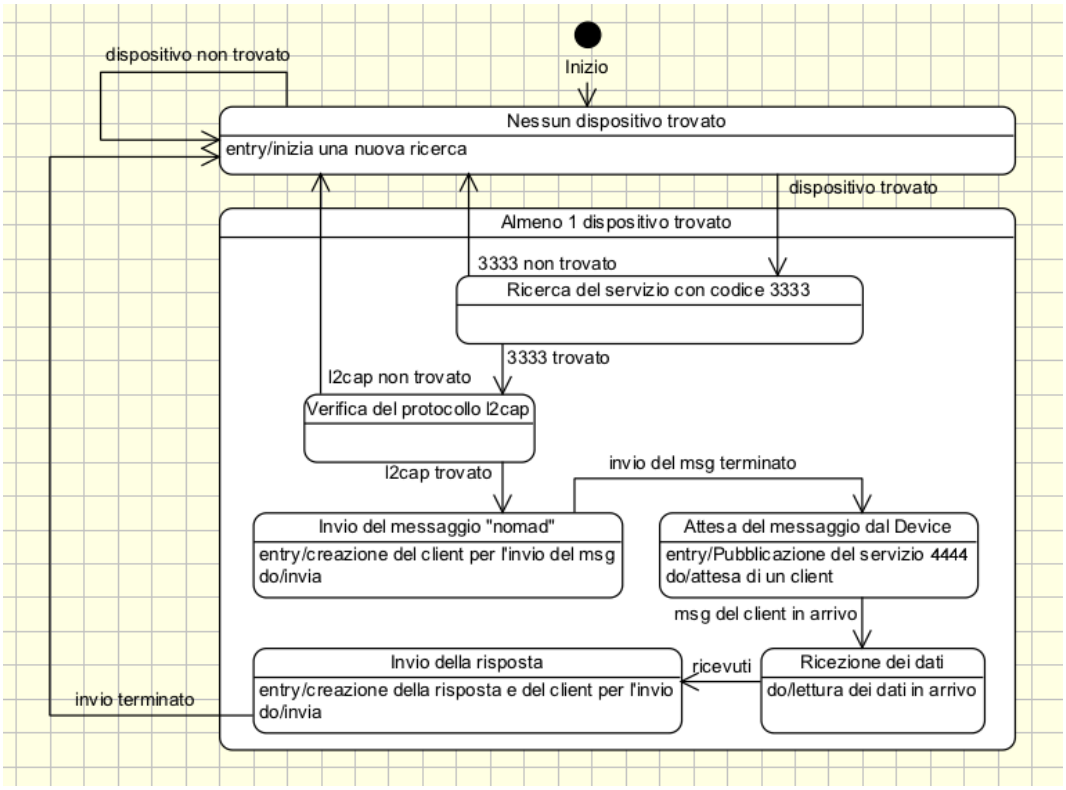

Figura 4.52: *UML State Diagram rappresentate il comportamento del server Bluetooth dell'Embedded*

informazioni siano compatibili con l'attività che il paziente sta per svolgere. Nel caso in cui l'attività in questione non sia contenuta tra quelle assegnate al paziente (o sia già stata completata), l'*Embedded* invia un messaggio di errore. Se invece non ci sono incompatibilità, viene inviato al *Device* un messaggio di conferma, utilizzato poi per aggiornare il database delle attività quotidiane.

L'output scritto sul terminale dall'*Embedded* in un caso esemplificativo (figura 4.54 e 4.55) può essere utile nell'analisi del *Sequence Diagram* di figura 4.53, in cui viene messa in evidenza l'interazione con il *Device*.

Infatti, dopo aver identificato un dispositivo Bluetooth ed aver verificato che è un *Device* (perché presenta il servizio con codice `3333`), l'*Embedded* invia il messaggio `embedded` e riceve in risposta il profilo del paziente. Nella figura 4.54 questo profilo contiene i dati del paziente e dell'attività "rehabilitation" ancora da compiere. Dato che nello scenario rappresentato questa è l'attività giusta associata al luogo in cui il paziente si trova, *Embedded* è abilitato ad aggiornare il profilo del paziente, inviando al *Device* un messaggio di conferma.

La figura 4.55 potrebbe invece corrispondere al caso in cui lo stesso paziente

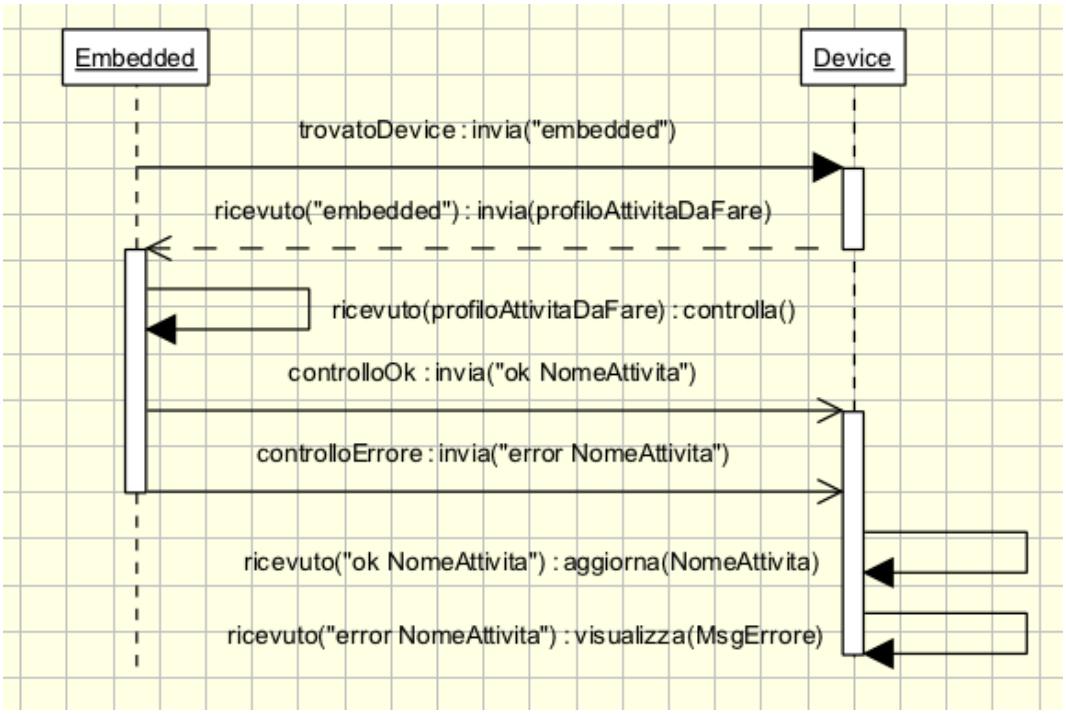

Figura 4.53: *UML Sequence Diagram rappresentante la comunicazione tra il dispositivo Embedded e il Device*

tenti di ripetere nell'arco della giornata un'attività già completata. Infatti, il profilo inviato dal *Device* mostra che non sono presenti attività ancora da compiere, e quindi sicuramente al paziente non è consentito effettuare l'attività associata a quel luogo (e a quell'*Embedded*, di conseguenza). In questo caso quindi verrà inviato al *Device* un messaggio di errore, per indicare che l'*Embedded* non è abilitato ad aggiornare il profilo del paziente, e per indicare una possibile situazione di emergenza agli operatori competenti.

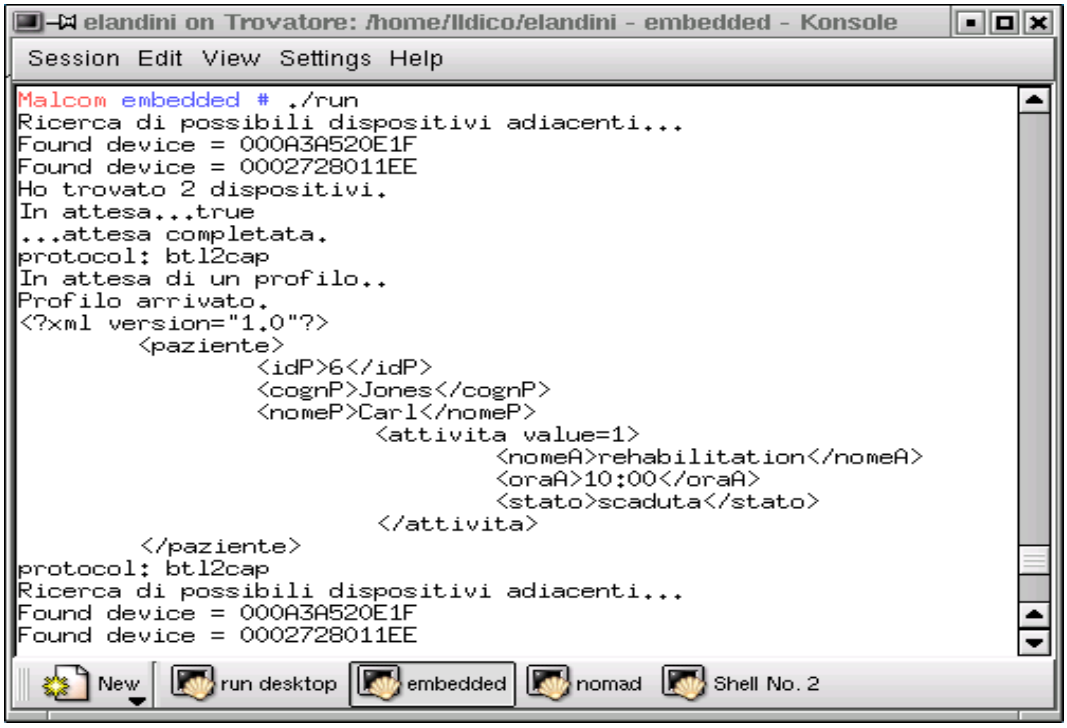

Figura 4.54: *Output dell'Embedded nella fase di aggiornamento*

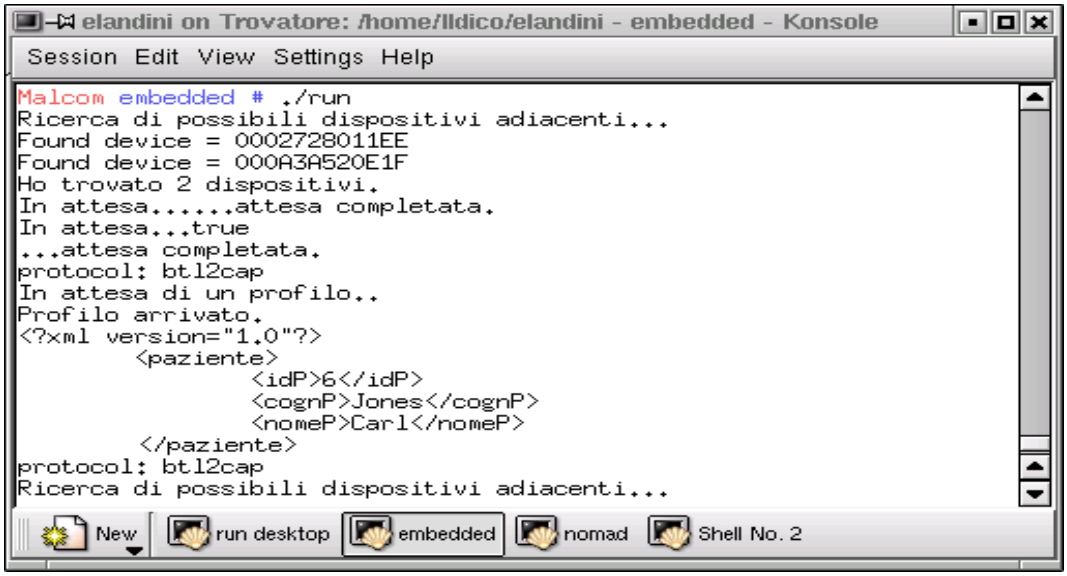

Figura 4.55: *Output scritto dell'Embedded in presenza di un'incompatibilità nel profilo del paziente*

Capitolo 5

Conclusioni

5.1 Conclusioni

In questo lavoro di tesi si è considerato il problema della realizzazioni di sistemi di assistenza ad anziani e disabili in grado di fornire un valido supporto agli operatori del settore ed alle famiglie dei malati, consentendo ai pazienti di acquistare una qualità della vita migliore.

Il primo passo per il raggiungimento di questo scopo è stato lo studio delle tecniche applicate nella realizzazione di vari sistemi di automazione e sorveglianza dell'ambiente atti a monitorare i comportamenti degli individui al fine di individuare ed evitare pericolose situazioni di emergenza.

È stato quindi definito uno scenario dettagliato che permettesse di valutare le prestazioni del sistemi per quel che riguarda le problematiche comuni a questi progetti, quali la mobilità degli individui e delle informazioni, la distribuzione dell'ambiente e l'interoperabilità tra dispositivi eterogenei. I *benchmark* comuni così identificati sono stati formalizzati come requisiti del sistema. Lo scenario è stato basato sulle attività quotidiane comunemente svolte in strutture assistenziali, quali possono essere *day-hospital* per anziani o ospedali geriatrici a lunga degenza.

Questo progetto di tesi, inserendosi nel più ampio progetto RoboCare, in corso presso il Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Parma, ha creato un sistema di comunicazione wireless ad hoc per l'integrazione dei robot di servizio già operanti nell'ambiente, ed ha sviluppato un'architettura domotica basilare per il completamento di compiti di varia complessità in cui sia richiesta l'interazione di dispositivi diversi. È stato quindi fornito un valore aggiunto agli strumenti già inseriti nel sistema, e ne è stata ridisegnata la cornice di azione.

La sperimentazione ha messo in evidenza i benefici apportati da un approccio di questo tipo. Prima di tutto è stata dimostrata l'efficacia delle tecnologie di integrazione utilizzate, l'unione delle quali si è dimostrata indispensabile per il pieno

raggiungimento degli obiettivi prefissati. L'utilizzo dello standard Bluetooth per le comunicazioni wireless tra i dispositivi ha infatti permesso di soddisfare i requisiti di trasparenza e di mobilità in un ambiente fortemente distribuito. Inoltre, la sinergia con la piattaforma J2ME, ha consentito la coesistenza e l'interazione di dispositivi diversi, rispondendo al requisito di interoperabilità, ulteriormente rafforzata dall'uso di `xml` per la comunicazione dei messaggi tra i vari elementi del sistema. La scelta di J2ME ha posto infine le basi per una minimizzazione delle richieste computazionali e di memoria dei possibili apparecchi utilizzati.

In secondo luogo la sperimentazione ha evidenziato la possibilità di realizzare concretamente infrastrutture tecnologiche al servizio della persona in grado di ricreare quelle condizioni necessarie per consentire ad anziani e disabili un maggior grado di autonomia. La monitorizzazione delle attività quotidiane dei malati riduce infatti il tempo richiesto ad assistenti sanitari e familiari, la cui presenza viene automaticamente richiesta in caso di emergenza.

Infine, i servizi forniti da questa infrastruttura hanno dimostrato di rispondere in tempi brevi e a basso costo alle richieste degli utenti ed ai cambiamenti dell'ambiente. Questo è possibile perché i numerosi dispositivi *embedded* possono elaborare le informazioni ricevute da fonti diverse per ottenere una visione completa ed aggiornata del sistema, adeguandovisi di conseguenza. Inoltre, le informazioni possono essere trasferite dal sistema distribuito a quello centrale, raggiungendo gli addetti responsabili per informarli di possibili comportamenti anomali nell'ambiente. Il sistema centrale è infatti dotato di un database e di un'intuitiva interfaccia grafica che consente la visualizzazione dello stato attuale e degli eventi previsti nel corso della giornata.

5.2 Sviluppi futuri

Per quanto riguarda l'osservazione costante dei comportamenti di anziani e disabili, i possibili sviluppi del progetto riguardano l'apprendimento delle loro abitudini quotidiane, al fine di adattate le azioni dei robot di servizio e dell'ambiente alle necessità dei pazienti. La prospettiva futura è infatti quella di passare da un'automazione generale dei servizi forniti, ad una personalizzazione in grado di adattare il sistema e l'ambiente alle esigenze dei suoi abitanti. Questa visione è attuabile con la realizzazione di agenti software in grado di utilizzare i dati estratti da sensori locali e remoti per personalizzare l'ambiente a seconda delle peculiarità di ogni individuo e prevenirne così le richieste.

Un ulteriore sviluppo consiste nella creazione di un sistema di reti eterogenee da integrare a Bluetooth per la copertura totale degli spazi del sistema. Lo standard IEEE 802.11g potrebbe ad esempio essere utilizzato per la comunicazione immediata dei dati aggiornati dei pazienti dai robot mobili e i dispositivi *embedded* verso

la postazione centrale, nel caso in cui il Bluetooth non sia in grado di coprire la distanza tra questi due elementi del sistema. Una rete via cavo dedicata e a banda larga potrebbe poi estendere la comunicazione tra un padiglione dell'ospedale e l'altro. In questo nuovo scenario diventa quindi essenziale l'uso di uno o più *gateway* (la cui posizione deve essere attentamente studiata a seconda delle caratteristiche del sistema) che consenta un'interfaccia comune ai diversi standard di comunicazione utilizzati.

Un altro aspetto fondamentale da tenere in considerazione in un futuro ampliamento del sistema è quello della sicurezza, intesa sia come incolumità degli abitanti del sistema, che come sicurezza dei dati trasmessi. In questa seconda accezione, potrebbe essere utile fare uso del *gateway* sopra citato per la realizzazione di un sistema di controllo del flusso di dati e l'introduzione di meccanismi di sicurezza quali la criptazione dei dati trasmessi.

Infine, data la stretta analogia tra l'hardware e il software utilizzati in sistemi di questo tipo, sarebbe auspicabile lo sviluppo di un dispositivo *wearable*, in grado di supportare lo standard Bluetooth e le API Java per il Bluetooth (JABWT), in modo da testare il sistema dal punto di vista concreto della trasparenza. Le possibilità a disposizione di uno strumento di questo tipo possono poi essere ulteriormente ampliate da vari sensori (ad esempio di temperature o pressione) utilizzati per monitorare lo stato di salute del paziente e comunicare immediatamente eventuali anomalie.

5.3 Valutazioni etiche

Le riflessioni e le discussioni che hanno accompagnato la realizzazione di questa tesi hanno messo in evidenza la necessità di precisare le valutazioni etiche alla base di un progetto di assistenza di questo tipo. L'utilizzo di robot di servizio e di sistemi di sorveglianza automatizzata delle attività quotidiane di anziani e disabili generano infatti le critiche di coloro che intravedono due possibili minacce:

- La limitazione delle libertà personali, messe in discussione dalla continua monitorizzazione di *tutte* le attività dell'individuo.
- L'abbandono di anziani e disabili all'interno di sistemi di assistenza automatizzati, in cui le famiglie e gli assistenti sanitari vengano sostituiti da robot di servizio altamente specializzati.

Per quel che riguarda la limitazione delle libertà individuali, è necessario tener conto delle specifiche esigenze delle persone disabili a cui questi sistemi sono indirizzati. La possibilità di riappropriarsi di un maggior controllo dell'ambiente in cui si vive, coniugato alla sicurezza garantita da un sistema di monitorizzazione capace

di identificare ed evitare situazioni di emergenza, ha per un malato la priorità rispetto al timore di vivere in un ambiente domotizzato. Anzi, si può affermare che in questo contesto si impone un nuovo concetto di libertà: la libertà di poter scegliere di migliore la qualità della propria vita, anche in presenza di un grave deficit.

In relazione alla seconda critica, è importante evidenziare come questi sistemi siano realizzati per supportare familiari ed operatori, non per sostituirne la figura. Le nuove tecnologie vengono infatti introdotte per migliorare l'assistenza quotidiana, operando soprattutto sui punti deboli tipici (ed inevitabili) della natura umana. Un sistema domotizzato può ad esempio mantenere un grado di attenzione costante su un individuo che nessun operatore, per quanto attento, possa mai raggiungere. Analogamente, la valutazione dell'elevato numero di variabili che possono portare alla previsione di un'imminente situazione di emergenza, non è alla portata di alcun essere umano.

Appendice A

A.1 Bluez

Bluez [42] è lo stack Bluetooth ufficiale per il sistema operativo Linux e fornisce supporto per i livelli ed i protocolli principali di questo standard. Presenta inoltre numerose caratteristiche, tra cui:

- È flessibile ed efficiente, ed è costruito su un'architettura modulare.
- Supporta più dispositivi Bluetooth.
- L'elaborazione dei dati viene gestita in modalità *multithread*.
- Può essere utilizzato sia con dispositivi USB che seriali (RS-232).

Per utilizzare lo stack Bluez (disponibile a partire dal kernel 2.4.4) è consigliabile disporre del kernel 2.6. Si evita così di dover utilizzare delle *patch* già contenute nelle versioni 2.6.x. Prima di tutto è necessario installare, lavorando sempre da *root* il supporto per il Bluetooth, comprendente i seguenti elementi:

- Bluetooth subsystem support;
- L2CAP protocol support;
- SCO links support;
- RFCOMM protocol;
- RFCOMM tty support;
- BNEP protocol support;
- Bluetooth device driver: sia il driver HCI USB, che quello HCI VHCI.

Non deve invece essere installato contemporaneamente il supporto USB Bluetooth, che impedirebbe il funzionamento dello stack Bluez. Comunque, nel caso in cui venga erroneamente scelta questa opzione, viene visualizzato un messaggio che informa dell'incompatibilità con lo stack Bluez.

A questo punto si devono caricare i seguenti moduli, utilizzando (sempre da terminale e come *root*) il comando `modprobe`:

- hci_usb;
- rfcomm;
- l2cap;
- bluetooth;

Per verificare l'avvenuto caricamento, il comando `lsmod` permette di visualizzare tutti i moduli correntemente caricati nel sistema. Se, dopo aver installato il supporto per il Bluetooth, non venisse consentito il caricamento dei moduli sopra citati, potrebbe significare che il *kernel* utilizzato non supporta gli strumenti necessari per la loro esecuzione. È quindi necessario scaricare dal sito `http://www.bluez.org/` i seguenti *tool*:

- bluez-sdp;
- bluez-utils;
- bluez-lib.

Questi strumenti vanno compilati eseguendo in sequenza da terminale i comandi

```
./configure
make
make install
```

Eseguendo nuovamente il `modprobe` dei moduli elencati, non dovrebbero verificarsi altri problemi.

Connettendo un dispositivo Bluetooth (ad esempio un adattatore USB), si può verificare se questo è attivo digitando il comando `hciconfig`. Un possibile output corrispondente ad un dispositivo funzionante è il seguente:

```
hci0:   Type: USB
BD Address: 00:02:72:B1:2F:46 ACL MTU: 192:8  SCO MTU: 64:8
UP RUNNING PSCAN ISCAN
RX bytes:99 acl:0 sco:0 events:13 errors:0
TX bytes:296 acl:0 sco:0 commands:12 errors:0
```

Nel caso in cui questo output non venga visualizzato, significa che il dispositivo deve ancora essere attivato. Per farlo, sempre da *root*, si deve digitare

```
hciconfig hci0 up
```

I servizi che utilizzano *HCI* e *SDP* richiedono anche l'attivazione dei due *demoni* corrispondenti, ovvero `hcid` e `sdpd`. Se ad esempio, si vuole testare la comunicazione tra un *client* ed un *server* e si riscontrano dei problemi nell'identificazione dei servizi pubblicati dal *server*, potrebbe essere utile controllare la presenza di `sdpd` tra i processi attivati. Nel caso sia assente, può essere attivato digitando il comando

```
sdpd
```

Un'altra utile osservazione è quella relativa al *pairing*, ovvero alla capacità di effettuare un'associazione con un altro dispositivo utilizzando una chiave numerica. La chiave stabilita va inserita nel file

```
/etc/bluetooth/pin
```

e deve essere digitata identica dal dispositivo che vuole effettuare il *pairing* al momento della richiesta.

Per fare un esempio, per fissare la chiave

```
1234
```

si deve scrivere nel file `pin`

```
echo "PIN:1234"
```

Se si vuole modificare il file utilizzato per il *pairing*, o in generale la configurazione del dispositivo Bluetooth utilizzato, si devono cambiare le informazioni scritte nel file

```
/etc/bluetooth/hcid.conf
```

corrispondente alla configurazione del *demone* `hcid`. Quella utilizzata per questo progetto di tesi è mostrata nel listato A.1.

```
# HCI daemon configuration file.
#
# $Id: hcid.conf,v 1.3 2002/07/18 18:12:46 maxk Exp $
#

# HCId options
options {
        # Automatically initialize new devices
        autoinit yes;

        # Security Manager mode
        #     none - Security manager disabled
        #     auto - Use local PIN for incoming connections
        #     user - Always ask user for a PIN
        #
        security user;

        # Pairing mode
        #     none   - Pairing disabled
        #     multi - Allow pairing with already paired devices
        #     once   - Pair once and deny successive attempts
        pairing multi;

        # PIN helper
        pin_helper /etc/bluetooth/pin;
}

# Default settings for HCI devices
device {
        # Local device name
        #     %d - device id
        #     %h - host name
        name "Malcom␣(%d)";

        # Local device class
        class 0x100;

        # Default packet type
        #pkt_type DH1,DM1,HV1;

        # Inquiry and Page scan
        iscan enable; pscan enable;
```

```
        # Default link mode
        #   none   – no specific policy
        #   accept – always accept incoming connections
        #   master – become master on incoming connections,
        #            deny role switch on outgoing connections
        #
        #lm accept,master;
        #
        lm accept;

        # Default link policy
        #   none    – no specific policy
        #   rswitch – allow role switch
        #   hold    – allow hold mode
        #   sniff   – allow sniff mode
        #   park    – allow park mode
        #
        #lp hold,sniff;
        #
        lp hold,sniff,park;

        # Authentication and Encryption
        #auth enable;
        #encrypt enable;
}
```

Listato A.1: *Configurazione del file* `hcid.conf`

All'interno di Bluez sono inoltre presenti alcuni strumenti per testare il proprio dispositivo e visualizzare le caratteristiche di quelli adiacenti. Uno di questi è `hcitool`, che con il comando

```
hcitool scan
```

permette di visualizzare l'indirizzo dei dispositivi Bluetooth raggiungibili.

Bibliografia

[1] A. Lori, A. Golini, and B. Cantalini. Atlante dell'invecchiamento della popolazione, 1995.

[2] A. Bianchetti, C. Geroldi, and M. Trabucchi. *La malattia di Alzheimer in Italia: qualità e costi dell'assistenza*. Science Adv, 1995.

[3] B. Jönsson, L. Jönsson, and A. Wimo. I costi della demenza: una review. In *Demenza*, pages 365–393. CIC Edizioni Internazionali, 2004.

[4] J. Abascal and A. Civit. Mobile Communication for Older People: New Opportunities for Autonomous Life. *Workshop on Universal Accessibility of Ubiquitous Computing: Providing for the Elderly*, 2002.

[5] M. Berggren. Wireless communication in telemedicine using Bluetooth and IEEE 802.11b. *Uppsala Master's Thesis in Computing Science*, 2001.

[6] D. Bennett. The net comes home. In *NewScientist*, number 2382, 2003.

[7] Intelligent Inhabited Environments Group. `http://iieg.essex.ac.uk`.

[8] A. Reyes, A. Barba, V. Callaghan, and G. Clarke. The Integration of Wireless, Wired Access and Embedded Agents in Intelligent Buildings. *The fifth World Multi-Conference on Systemics, Cybernetics and Informatics*, 2001.

[9] A. Holmes, H. Duman, and A. Pounds-Cornish. The *iDorm*: Gateway to Heterogeneous Networking Environments.

[10] R. Brooks. A Robust Layered Control System for a Mobile Robot. In *IEEE Journal of Robotics and Automation*, 1986.

[11] R. Brooks. Intelligence Without Representation. In *Artificial Intelligence 47*, pages 139–159, 1987.

[12] R. Brooks. Intelligence Without Reason. In *Computer and Thought*, 1991.

[13] S. Sharples, V. Callaghan, and G. Clarke. A Multi-Agent Architecture For Intelligent Building Sensing and Control. *International Sensor Review Journal*, 1999.

[14] V. Callaghan, G. Clarke, M. Colley, and H. Hagras. A Soft-Computing DAI Architecture for Intelligent Buildings. *Studies in Fuzziness and Soft Computing Agents*, 2000.

[15] B. C. Williams and P. P. Nayak. Immobile Robots: AI in the New Millenium. *AI Magazine*, 1996.

[16] V. Callaghan, G. Clarke, M. Colley, and H. Hagras. Embedding Intelligence: Research Issues for Ubiquitous Computing. *The first Equator IRC Workshop on Ubiquitous Computing*, 2001.

[17] H. Duman, H. Hagras, and V. Callaghan. A Soft-Computing based Approach to Intelligent Association in Agent-Based Ambient-Intelligence Environments. *The fourth International Conference on Recent Advances in Soft Computing*, 2002.

[18] A. Pounds-Cornish and A. Holmes. The iDorm: a Practical Deployment of Grid Technology.

[19] H. Hagras, V. Callaghan, G. Clarke, M. Colley, A. Pounds-Cornish, A. Holmes, and H. Duman. Incremental Synchronous Learning for Embedded-Agents Operating in Obiquitous Computing Environments. *Frontieres in Artificial Intelligence and Application*, 2002.

[20] H. Hagras, V. Callaghan, M. Colley, G. Clarke, and H. Duman. Online Learning and Adaption for Intelligent Embedded Agents Operating in Domestic Environments. *Fusion of Soft Computing and Hard Computing for Autonomous Robotic Systems*, 2002.

[21] Università dell'Essex. `http://www.essex.ac.uk/`.

[22] SNAP. `http://www.imsys.se`.

[23] F. Cayci, V. Callaghan, and G. Clarke. A Distribuited Intelligent Building Agent Language (DIBAL). *The sixth International Conference on Information Systems Analysis and Synthesis*, 2000.

[24] P. Davidsson. Energy Savings and Value Added Services; Controlling Intelligent-Buildings Using a Multi-Agent System Approach. In *DA/DSM Europe DistribuTECH, PennWell*, 1998.

[25] M. Mozer. The Neural Network House: An Environment That Adapts To Its Inhabitants. In *Proc of American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pages 110–114, 1998.

[26] N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes. HIVE: Distribuited agents for networking thinks. In *ASA/MA*, 1999.

[27] R. Krikorian. Internet 0: Bringing IP to Leaf Node. *O?Reilly Emerging Technologies Conference*, 2003.

[28] Disappearing Computer. `http://www.disappearing-computer.net`.

[29] A. Kameas, S. Bellis, I. Mavrommati, K. Delaney, M. Colley, and A. Pounds-Cornish. An Architecture that Treats Everyday Objects as Communicating Tangible Components. In *The First IEEE International Conference on Pervasive Computing and Communications*, 2003.

[30] The eHealth Initiative. `http://www.ehealthinitiative.org/`.

[31] Integer House. `http://www.integerproject.co.uk`, `http://demo.echelon.com`.

[32] The Internet Home. `http://www.cisco.com/warp/public/3/uk/ihome`.

[33] Bluetags. `http://www.bluetags.com`.

[34] Infrared Data Association. `http://www.irda.org`.

[35] Java APIs for Bluetooth. `http://www.jcp.org/en/jsr/detail?id=82`.

[36] Time for a blew. `http://www.rococosoft.com/publications/index.html`.

[37] Rococosoft. `http://www.rococosoft.com`.

[38] Bluetooth Stack per J2ME di Atinav.
`http://www.atinav.com/bluetooth/bt_javaj2me.htm`.

[39] Progetto JavaBluetooth. `http://www.javabluetooth.org`.

[40] Impronto Developer Kit.
`http://www.rococosoft.com/products/impronto.html`.

[41] Distribuzione Gentoo. `http://www.gentoo.org/`.

[42] Stack Bluez. `http://www.bluez.org/`.

[43] Digicom. `http://www.digicom.it`.

[44] Nomad 200. `http://nomadic.sourceforge.net/production/n200/`.

[45] Me4Se. `http://kobjects.dyndns.org/kobjects/auto?self=$81d91ea1000000f5d073810%0`.

[46] Emulatore Nokia Serie 60.
`http://www.forum.nokia.com/main/1,6566,034-243,00.html`.

[47] MySql. `http://www.mysql.com/`.

[48] Tecnologia JDBC. `http://java.sun.com/products/jdbc/`.

[49] kXml Parser. `http://kxml.objectweb.org/`.

[50] Apache Ant. `http://ant.apache.org/`.

[51] Progetto KDEBluetooth. `http://kde-bluetooth.sourceforge.net/`.

[52] M. Colley, G. Clarke, V. Callaghan, and A. Pounds-Cornish. Intelligent Inhabited Environments: Cooperative Robotics and Buildings. *32nd International Symposium on Robotics*, 2001.

[53] V. Callaghan, G. Clarke, , A. Pounds-Cornish, and S. Sharples. Buildings as Intelligent Autonomous Systems: A Model for Integrating Personal and Building Agent. *Sixth International Conference on Intelligent Autonomous Systems, Venice*, 2000.

[54] V. Callaghan, M. Colley, G. Clarke, and H. Hagras. The Cognitive Disappearing of the Computer: Intelligent Artifacts and Embedded Agents. *Workshop WS4 on Cognitive Versus Physical Disappearance*, 2001.