UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Laurea Specialistica in Ingegneria Informatica

Sistema ubiquo applicato all'assistenza a persone con problemi di memoria

Laureando : **Alberto Caccin**

Relatore: Prof. **Carlo Ferrari**

Padova, 19 aprile 2010

a.a. 2009-2010

...to be a rock and not to roll...

Abstract

Il modello di computazione Ubiquitous Computing sarà di basilare importanza per ogni sistema futuro nell'ambito dell'IT. Alcuni settori soprattutto industriali e pubblici stanno già andando in questa direzione. L'aumento della capacità di interagire tra dispositivi diversi e tra reti diverse, per creare una rete diffusa, in cui l'utente è immerso in essa, è l'obiettivo principale di tale modello. In questa tesi tale paradigma è stato calato nel contesto dell'assistenza alle persone, soprattutto anziane, definendo un sistema che rispondesse adeguatamente alle esigenze degli utenti con lievi problemi di memoria. Per "lievi problemi" si intende gli effetti causati dal naturale decadimento delle facoltà mnemoniche, e non dovute a patologie o traumi. Dall'analisi degli ultimi lavori sul tema, si è giunti a definire un'architettura di sistema distribuita aperta, peer-to-peer, in completa controtendenza delle soluzioni attuali, che puntano ad un modello centralizzato. Si sono definiti più servizi, che rispondono ad esigenze diverse dell'utente : dall'assunzione dei medicinali ad un'agenda degli impegni, da un sistema trova oggetti smarriti al controllo remoto della casa. Utilizzando la tecnologia Java per sistemi distribuiti (Jini) si è implementata un'infrastruttura (che abbiamo chiamato Reminder*) che si basa sulle funzioni del sistema descritto. Tale infrastruttura analizza tutte le operazioni principali dei servizi precedentemente definiti : dalla gestione con il database, all'interazione tra le entità di Jini, fino all'interfacciamento con i dispositivi esterni.*

Indice

Elenco delle figure

Introduzione

La tesi tratta della progettazione di un sistema ubiquo applicato all'assistenza di persone con problemi di memoria. Quindi affronta due tematiche attuali : la computazione ubiqua e l'assistenza a persone anziane.

Ubiquitous Computing, Pervasive Computing, Calm Technology, Internet of Things. Numerosi nomi sono usati per descrivere sinteticamente il nuovo paradigma di computazione. Nel titolo, e nel proseguo della trattazione della tesi, si utilizza il termine computazione ubiqua. In questo paradigma l'uomo è immerso in un ambiente "popolato" da una moltitudine di dispositivi che comunicano tra loro, in maniera autonoma, per fornire informazioni, servizi e quant'altro. La direzione verso la quale i primi ricercatori si sono mossi è stata creare un ambiente intelligente, da cui gli uomini potessero trarre più informazioni possibili, senza però esserne distratti. Informare senza disturbare. Vengono unite due grandi sfide : da una parte riuscire a far interagire una moltitudine di dispositivi per creare una rete diffusa, dalle enormi potenzialità, dall'altra modellare questa tecnologia affinché non sia di disturbo agli utenti, cioè che il fluire delle informazioni sia il più naturale possibile. L'aspetto fondamentale di tale modello è proprio la comunicazione, cioè il fatto di collegare apparecchi diversi (basti pensare all'applicazione nella domotica, dove ci sono elettrodomestici, lettori dvd, luci e sensori diversi), creando più reti. Oltre a questo, la creazione di dispositivi di dimensioni molto ridotte, hanno reso possibili servizi molto più personali e pervasivi (sistemi di localizzazione, identificazione di oggetti tramite RFID, etc).
La definizione di questo modello di computazione risale alla fine degli anni 80 del '900, anche se tracce di queste tematiche si possono intravedere nei romanzi di fantascienza di anni precedenti. Quest'ultimi hanno influenzato molto l'opinione pubblica, su questo tema, molto affascinante, ma spesso visto come oscuro e minaccioso. I grandi timori sono l'avanzare della tecnologia come minaccia, l'aumentare di dispositivi elettronici, che *pervadono* la nostra esistenza, in ogni aspetto (anche i più personali) con conseguente perdita di privacy e perdita di umanità. In realtà questa visione è "lievemente" distorta. L'ubiquitous computing è certamente una grossa e ghiotta opportunità di migliorare l'interazione tra i dispositivi che già ab-

biamo nelle nostre case, ma soprattutto di creare nuovi, e fino a qualche tempo fa impensabili, servizi. Si va dalle auto intelligenti (che possono prevenire incidenti, avvisare della presenza di code, evitare autonomamente gli ostacoli), servizi turistici e cittadini migliorati (con informazioni in tempo reale in ogni parte della città), assistenza sanitaria completa a casa, e così via. Ogni settore, ogni lato della vita può assistere al nascere di nuove opportunità.

Un settore in cui la ricerca è sempre molto attiva è quello medico-sanitario. In questo, il tema dell'assistenza agli anziani ricopre sicuramente un ruolo molto importante. Infatti numerose pubblicazioni, progetti, e programmi di finanziamento pubblici investono quello che viene chiamato ElderCare. I fattori che elevano l'assistenza a persone appartenenti alla cosiddetta terza età a uno dei temi più seguiti nell'ambito socio-sanitario sono molteplici. Per prima cosa il numero di anziani è in costante aumento in tutto il mondo. Si stima che nel 2008 il 7% della popolazione mondiale fosse over65, percentuale che raddoppiava, e in alcuni casi triplicava, se si analizzavano i Paesi più sviluppati. Oltre a ciò, con l'avanzare degli anni, la salute delle persone diventa più vacillante : la probabilità di insorgenza di patologie aumenta e c'è un naturale decadimento di alcune facoltà (tra cui vista, movimenti, memoria). Ciò causa la necessità di un'assistenza continua (casalinga, quando non ospedaliera).

A queste due tematiche (computazione ubiqua e assistenza alla cura delle persone) si aggiunge il bisogno di apertura dei sistemi progettati. Apertura sia nei riguardi di eventuali aggiunte future, e di personalizzazioni (o "customizzazioni") dell'utente, sia nei riguardi di integrazioni di terze parti. Il primo tipo di apertura è ormai presente in ogni software. Molti sistemi vengono progettati con un modello modulare, lasciando aperta la strada ad aggiunte di ulteriori funzionalità, da integrare all'interno del sistema esistente senza dover sostituirlo per intero. Questo vale anche per le personalizzazioni, in cui moduli diversi vengono installati a utenti diversi (a seconda dei bisogni e delle richieste). Il secondo tipo di apertura, e cioè quello rivolto verso l'integrazione da parte di parti terze, è meno sfruttato. Questa però dovrà essere la direzione dello sviluppo nell'ambito dell'Ubiquitous Computing, in quanto uno degli obiettivi è di far comunicare sia dispositivi diversi (che molto probabilmente saranno prodotti da aziende diverse) sia reti diverse (per esempio la rete casalinga con la rete della propria auto). Per questo bisogno di aprire i sistemi e le reti si è scelto di usare la tecnologia Jini, basata su Java, che risponde, con il suo modello distribuito peer-to-peer sottostante, in maniera adeguata a tali necessità.

Ora vediamo in sintesi di cosa trattano i capitoli seguenti.

Capitolo 1. Nel primo capitolo verrà fornita una panoramica sulla tematica affrontata dal sistema descritto in seguito : l'ubiquitous computing applicato all'assistenza agli anziani. Questo è un tema molto attuale, e, negli ultimi anni, sono state presentate numerose soluzioni per risolvere le problematiche associate.

Capitolo 2. In questo capitolo viene presentato il sistema ubiquo, citato nel titolo della tesi. Esso è composto da vari servizi, che rispondono alle varie esigenze dell'utente. Vengono confrontati bisogni dal lato dell'utente (lato committente, o lato user) e le risposte tecniche che dovranno rispondere a tali richieste (lato tecnico). Inoltre si presentano varie possibili architetture utilizzabili per implementare i servizi descritti, e si presenta il modello effettivamente scelto.

Capitolo 3. Il sistema presentato nel capitolo precedente viene implementato utilizzando la tecnologia JINI. Essa risulta un'estensione a Java, a cui va ad aggiungere alcune librerie, necessarie ad implementare un modello distribuito, costituito dall'interazione tra tre elementi principali : il service, il client e il LookupService.

Capitolo 4. Del sistema presentato nel secondo capitolo, viene scelto un servizio che verrà implementato attraverso la tecnologia Jini. Esso ben rappresenta le funzionalità fondamentali di ogni servizio del sistema ideato e i meccanismi principali tra le varie componenti di Jini.

Capitolo 5. In questo capitolo finale vengono riassunti i punti critici del sistema presentato, le future possibili aggiunte ed estensioni al progetto e gli scenari futuri e le conclusioni sul lavoro fin qui svolto.

Capitolo 1

Ubiquitous Computing

1.1 Overview

In origine fu il mainframe. Una singola macchina, costosa e ingombrante, che veniva usata da pochi addetti e disponibile solo in determinati luoghi (centri di ricerca, università, etc). Il paradigma era "uno per tutti", un solo calcolatore per molti utenti. Poi venne il PC. Fu una rivoluzione nel mondo dell'informatica. I computer si diffusero in maniera prorompente, "uno per persona" era il motto. Attualmente noi siamo circondati da numerosi dispositivi, come i portatili, i telefonini, i lettori dvd, mp3, navigatori e molti altri ancora. Ora il modello è "Tanti per uno". Nella maggior parte dei casi, però, questi "elaboratori di informazioni" sono stand alone, interagiscono in maniera limitata, e sempre sotto l'iniziativa dell'utente. La direzione intrapresa in questi anni, e definita "Ubiquitous Computing" da Mark Weiser [1] nel 1988, è di immergere l'uomo in un ambiente intelligente, dove i dispositivi interagiscano tra loro, in maniera autonoma. Dispositivi molte volte meno potenti di quelli "usuali", ma che, grazie alla loro interazione, e alla loro diffusione, forniscono all'utente informazioni e servizi pervasivi (Pervasive Computing è un altro nome per indicare l'Ubiquitous Computing). L'idea di Weiser infatti era: "...highest ideal is to make a computer so imbedded, so fitting, so natural, that we use it without even thinking about it".

Ci sono settori che già utilizzano questo modello di computazione, come la domotica, la gestione di magazzini e alcuni servizi di monitoraggio di pazienti in strutture ospedaliere. Questi settori utilizzano tecnologie molto diverse tra loro, ma esse sono

[1] Mark D. Weiser (1952-1999) Fu Chief Technologist del centro di ricerca PARC (Palo Alto Research Center Incorporated) della Xerox di Palo Alto in California, USA. E' considerato il padre dell'Ubiquitous Computing, termine che coniò nel 1988.

accomunate dall'utilizzo capillare di sensori e dispositivi che consentono la creazione di un ambiente "intelligente". Nel caso della domotica, oltre alla gestione remota degli apparecchi domestici (elettrodomestici, luci, persiane, etc), abbiamo anche servizi per il risparmio energetico o per il condizionamento automatico della temperatura. Nel caso dei magazzini, c'è un costante bisogno di localizzare le merci, e tracciare gli spostamenti che vengono effettuati, senza dover utilizzare risorse umane. Nell'ambito ospedaliero invece, il monitoraggio continuo di parametri vitali sia dei pazienti ricoverati, sia di pazienti che si curano a casa propria, consente il tempestivo intervento dei soccorsi.
Per tutte queste attività, e non solo, sono disponibili svariate soluzioni tecnologiche, sia a livello di protocolli, sia a livello hardware. Per quanto riguarda la parte fisica, sempre più piede sta prendendo l'utilizzo degli RFID, sia attivi che passivi, per poter localizzare gli oggetti o le persone, o sensori e attuatori che molto spesso vengono affiancati da un potenza computazionale non banale. Per quanto riguarda i protocolli di trasmissione, il più diffuso e usato resta WLAN (IEEE 802.11 b, g) per far comunicare dispositivi a media distanza, mentre per la breve distanza Bluetooth (IEEE 802.15.1), Infrarossi (IrDA) e Zigbee (IEEE 802.15.4) sono quelli più gettonati. I problemi fisici connessi alla computazione pervasiva risiedono nelle ridotte funzionalità dei dispositivi, dalla capacità delle batterie (necessarie al funzionamento della maggior parte dei dispositivi), alla disponibilità di memoria, fino alla loro grandezza.

A conferma dell'importanza del tema, numerose conferenze vengono annualmente organizzate nel mondo. Le più importanti, tra queste, sono UBICOMP [1], MOBIQUITOUS [2], UBICOMM [3], UIC [4], ATC [5], PERVASIVE [6], a cui si affianca la rivista Pervasive Computing [7], edita da IEEE. Nell'ultimo biennio ha avuto grande risalto il tema della cosiddetta "*augmented reality*", in cui i sensori, in un ambiente pervasivo, riescono a dare all'utente informazioni sulla realtà circostante, che non sarebbero reperibili secondo i normali sensi umani. Altri sentieri molto battuti sono quelli delle "Vehicular Network", con interazioni auto-auto, e strada-auto, il risparmio di energia, con un monitoraggio capillare del consumo, l'uso del telefonino come soggetto principale della computazione ubiqua, e la cura della persone, "HealthCare", con dispositivi indossabili che controllano parametri fisici, o somministrano medicinali a intervalli regolari, a vantaggio della deospedalizzazione dei pazienti. Sempre di attualità, invece, resta la privacy. Infatti nei sistemi pervasivi vengono accumulate e gestite molte informazioni personali (preferenze, abitudini, funzioni vitali, localizzazione, etc) che necessitano di un buon livello di sicurezza. Questione molto delicata dato che spesso esse non vengono immagazzinate da una sola macchina, e i controlli devono essere estesi ad ogni dispositivo.

1.2 Elder Care : lo stato dell'arte

Tema attuale, dicevamo, è l'HealthCare, la cura della salute delle persone. La maggior parte dei lavori sono indirizzati verso l'assistenza a persone anziane. Essa affronta molti campi: le patologie, infatti, che affliggono principalmente gli anziani sono molteplici, e richiedono risposte molto diverse tra loro. Basti pensare al Morbo di Alzheimer, o al Morbo di Parkinson. Oltre agli aspetti patologici, l'ElderCare deve affrontare anche il declino fisico naturale delle persone. Ciò comporta, ad esempio, una maggiore debolezza del fisico con difficoltà motorie, riflessi meno pronti, memoria meno brillante, lievi perdite di vista e di udito.
La tecnologia viene in soccorso di tali problematiche, sia dalla parte fisica (hardware), sia dalla parte dei servizi (software). La parte hardware è molto presente nel trattamento di problemi fisici legati all'insorgenza di patologie (spesso invalidanti). Essa può concretizzarsi in macchine che aiutano il paziente nei movimenti di tutti i giorni, nell'assistenza ospedaliera (affianco al personale infermieristico) o in quella casalinga.
Il lato servizi, invece, punta più sull'assistenza casalinga dell'anziano, e mira a sopperire alle lacune fisiche che con il tempo si sono verificate. Viene data molta importanza ad un controllo agevole della casa, sia negli aspetti più critici, come il monitoraggio di perdite di gas, sia in quelli che possono rendere la vita delle persone meno pesante, come il controllo delle serrande e dei dispositivi domestici (luci, elettrodomestici, etc).

Abbiamo visto tutto questo dal lato delle richieste fisiche dell'anziano, ma esiste una componente molto importante, che viene spesso sottovalutata : l'aspetto psicologico.
Alcuni lavori [8] [9], vanno proprio in questa direzione, e dopo aver analizzato i bisogni fisiologici della "terza età", mettono in risalto questo aspetto, che dovranno affrontare necessariamente i progettisti. Molto importante è l'accettazione del sistema da parte dell'utente, sia per quanto riguarda l'interfaccia grafica, sia per quanto riguarda l'invasione nella sfera personale dei servizi offerti, nonché per il possibile rifiuto ad ogni possibile interferenza nella propria autonomia e autosufficienza. La cura di questi aspetti può portare al successo un sistema rispetto ad un altro, anche se quello scartato ha molte più funzionalità. Questo è vero per la maggior parte dei software e dei servizi, ma è di fondamentale importanza nell'ElderCare.
Una buona idea viene fornita dal progetto della Assistive Housing [10] in cui si fa affidamento alla TV, come interfaccia grafica ben conosciuta e non invadente. In questo lavoro viene prediletto l'uso di immagini, per comunicare in maniera più chiara possibile le funzioni offerte (possibilità di effettuare chiamate, riepilogo dei medicinali da assumere, possibilità di ascoltare contenuti audio-visivi).

Data la mole di problematiche esistenti, alcuni lavori si sono concentrati in punti ben specifici.
Nel progetto Autominder [11] si è implementato un calendario, molto flessibile, che avvisa in maniera intelligente (verifica se l'utente è impegnato in azioni più urgenti) scadenze e attività da compiere. In questo modo si riesce a fornire una discreta autonomia alle persone. In questo lavoro l'attenzione era focalizzata più sull'aspetto di AI (Intelligenza Artificiale), e quindi sulla relazione tra *constraint* tra gli impegni da ricordare e la definizione di regole, che sull'approntamento di un sistema.
Anche in [12] la segnalazione degli avvisi viene affiancata ad un riconoscimento delle azioni, usando sensori negli oggetti di uso comune. A differenza del lavoro precedentemente citato, questo si concentra sul metodo di riconoscimento delle azioni. Per fare ciò, propone un modello di rete bayesiana basato su catene di Markov.
L'effettiva realizzazione di un sistema viene presentata nel progetto COGKNOW [15], che affianca l'anziano non solo ricordandogli gli impegni della giornata, ma controllando gli accessi (porta di entrata chiusa o aperta) e avvisandolo di situazioni di pericolo (es. fughe di gas). Esso punta a favorire la sua indipendenza quotidiana, attraverso l'uso di un palmare (come dispositivo portatile) e di uno schermo touchscreen (come dispositivo fisso, installato in una stanza della casa). Il progetto presentato prevede un sistema centralizzato.

Le tematiche analizzate in questa tesi vanno nella direzione di quest'ultimo lavoro, ideando una serie di servizi che possano assistere persone (non necessariamente anziane) con lievi problemi di memoria, nella loro vita quotidiana, sia dentro casa, sia all'esterno.

Capitolo 2

Assistenza a persone con lievi problemi di memoria

Dall'esame del problema, si sono definiti alcuni servizi che possono incontrare le esigenze di una persona con lievi problemi di memoria. Con "lievi problemi di memoria" si intende la naturale e fisiologia perdita di memoria, e non legata a patologie o occorsa in seguito a traumi o interventi. In questo senso i servizi descritti in seguito possono adattarsi bene ad ogni tipologia di utente, fornendo un rinforzo alla memoria.

2.1 Requisiti del sistema

Vediamo ora di specificare questi servizi, sia per quanto riguarda il lato del committente (con alcuni scenari di esempio) sia per quanto riguarda i requisiti del lato tecnico (che dovranno rispondere alle esigenze dell'utente). I 4 servizi analizzati sono Remind Medicine, Object Finder, Controllo Remoto della Casa, Remind Impegni Giornata.

2.1.1 Lato committente

Remind Medicine

Lo scopo principale di questo servizio è di avvisare l'utente quando è giunto il momento di prendere i medicinali. Quindi si dovrà dotare il sistema della possibilità di modificare il piano giornaliero, o settimanale, di modificare la quantità e l'ora di assunzione delle medicine. Tali modifiche non solo sono apportate all'interno del

sistema casalingo, ma è necessario valutare la possibilità che l'aggiornamento possa essere effettuato anche dall'esterno, da strutture autorizzate (medici di condotta o ospedali).

SCENARIO : *Utente si reca dal medico di famiglia, e gli viene prescritto un nuovo medicinale. Il medico può aggiornare direttamente il sistema da casa, collegandosi in remoto con il servizio appropriato, oppure aggiornando il plan sul telefonino, che successivamente al ritorno a casa sincronizzerà i dati con il Remind.*

Questa opzione è molto importante dato che la persona può dimenticarsi di modificare il suo piano, oppure non ha le conoscenze tecniche per apportare modifiche complesse. Inoltre offre al medico il quadro completo sui medicinali assunti dal paziente.
Un problema delicato è assunto dagli avvisi, sia per la modalità, sia per il momento in cui eseguirli. Per quanto riguarda la modalità, si deve affiancare a degli avvisi sonori, anche avvisi visivi (TV, schermi, cellulari). Inoltre la conferma dell'avviso deve essere effettuata in maniera molto semplice. Il sistema dovrebbe scegliere i momenti più opportuni (o almeno non i più inopportuni) per avvisare l'utente (es. non quando si sta lavando). Se l'utente si trova fuori casa, gli avvisi dovrebbero raggiungerlo attraverso telefonino. Una funzione extra risiede nella memorizzazione di informazioni sui farmaci assunti (il cosiddetto "bugiardino") per una loro lettura più agevole, e per un controllo (perlomeno a livello base) su controindicazioni tra farmaci attualmente assunti.

Object Finder

L'obiettivo di tale servizio è rintracciare gli oggetti smarriti, all'interno della casa. Gli oggetti da rintracciare solitamente sono in numero modesto, e sono i più usati, ad esempio le chiavi di casa, gli occhiali, il telefonino. Uno schermo indicherà la stanza in cui si trova l'oggetto, e all'interno di questa l'oggetto da trovare verrà indicato o con un segnale luminoso o un segnale acustico.

Remind Impegni Giornata

Sistema offre la possibilità di pianificare la giornata inserendo promemoria, eventi, o altre attività in un calendario (che può essere integrato con il Remind Medicine). Tale remind dovrà essere dotato di connessione GPS per assistere l'utente anche in ambiente outdoor, avvisandolo di luoghi di interesse o di attività pianificate, quando si è in prossimità di essi. La mappa di tali luoghi è personalizzata dall'utente (o da familiari, accedendo da remoto) e deve essere di facile modifica e aggiornamento.

Controllo Remoto Casa

Tale servizio è simile a quelli proposti ora nel mercato. Esso dovrà monitorare le entrate della casa, avvisando quando esse sono state lasciate aperte, e, se possibile, chiuderle se sono a letto. Dovrà controllare se rubinetti del gas sono stati lasciati aperti (senza fiamma accesa dei fornelli), e abbinarli a sensori di rilevazione di fuoriuscita del gas. Anche i rubinetti dell'acqua (lavandini, doccia, etc) dovranno essere monitorati, per evitare sprechi, o altro. Il sistema dovrà controllare le luci, e gli elettrodomestici, per un controllo automatico dello spegnimento, e dovrà rendere disponibile lo stato della casa sullo schermo o sul telefonino. Le impostazioni dovranno essere modificabili da parte dell'utente in maniera accessibile.

Per quanto riguarda il sistema nel suo complesso, esso dovrà prevedere il possibile inserimento di ulteriori servizi, caratteristici delle stanze (es. cucina) o resi disponibili da strumenti della casa (es. stampante), che dovranno interagire con gli altri già esistenti.

2.1.2 Lato tecnico

Il sistema si comporrà di schermi LCD touchscreen, situati in posti chiave della casa (in numero variabile a seconda della disponibilità finanziaria), e un telefonino (necessario soprattutto per le attività outdoor dell'utente), per la visualizzazione degli avvisi, e per la modifica delle varie impostazioni.

Remind Medicine

Per la definizione del plan giornaliero dovranno essere disponibili informazioni sul momento della somministrazione (mattina, pomeriggio, etc) e su eventuali precedenze (prima o dopo i pasti, prima o dopo aver assunto altri medicinali, o dopo aver confermato l'assunzione di determinati farmaci). Saranno disponibili opzioni come specificare se l'assunzione dovrà essere a giorni alterni, o una volta a settimana. Quindi è necessario ipotizzare il bisogno di una *vista settimanale*, e non soltanto *giornaliera*.
Per la visualizzazione degli avvisi e la modifica del calendario potranno venire utilizzati sia gli schermi sia il telefonino, perciò si deve porre attenzione al bisogno di sincronia dei dati.
Affianco a questi requisiti, si dovranno approntare due sistemi che consentiranno di fornire funzionalità aggiuntive: un *sistema di localizzazione* dell'utente, e un *sistema di riconoscimento delle azioni*. Il sistema di localizzazione potrà contare o su rilevatori a ultrasuoni, o su lettori RFID (ipotizzando il fatto che l'utente indossi,

o abbia oggetti che hanno, un'etichetta RFID). Questo sistema ci permetterà di far apparire gli avvisi, o altre funzionalità, sullo schermo LCD più vicino all'utente, e di attivare solo gli altoparlanti della stanza in cui esso si trova. Inoltre ci permette di disabilitare gli avvisi sugli schermi, se esso non è presente nella casa. Il sistema di riconoscimento delle azioni serve soprattutto per definire il momento opportuno per interrompere l'utente, inoltrandogli gli avvisi, e per definire un log delle azioni, che può venire utilizzato per pazienti deospedalizzati, che devono essere monitorati da strutture ospedaliere. Per riconoscere le azioni ci si può avvalere del sistema di localizzazione (per identificare la stanza), verificare l'uso di elettrodomestici (es. TV, fornelli, frigo) e l'uso di rubinetti d'acqua (es. in bagno), e usare dispositivi wearable (es. RFID reader per riconoscere oggetti usati, o accelerometri per stabilire l'attività motoria).

Object Finder

Gli oggetti che si vuole riuscire a rintracciare vengono etichettati con un tag RFID (hanno dimensioni compatibili con gli oggetti proposti come chiavi, occhiali, etc). Inoltre si dovranno installare degli RFID reader in ogni stanza. Il numero dipenderà dalla potenza dei lettori stessi. In questo modo si potrà localizzare l'oggetto, anche se esso è nascosto all'interno di cassetti o borse. Una luce indicherà il punto in cui i lettori hanno trovato l'oggetto. Questo servizio è stato ispirato dal paper [13].

Remind Impegni

Verrà definito un calendario, in cui sarà possibile visualizzare gli avvisi collegati al Remind Medicine, con varie opzioni (visita medica, visita ad un amico, negozio, etc) e la lista dei luoghi di interesse (visibili nella mappa personalizzata). Il sistema di localizzazione (GPS) verrà installato (o si userà quello già esistente) nel telefonino, e ci sarà una sincronizzazione dei dati tra sistema indoor e dispositivo mobile ad ogni cambiamento del calendario o della mappa. Quando l'utente è in prossimità del luogo di interesse, l'avviso può essere inoltrato tramite messaggio, o chiamata vocale. Si dovrà valutare se sarà possibile avere una interazione di rete con il computer di bordo dell'automobile, e far in modo che il sistema di casa possa rintracciare l'utente, attraverso il GPS del cellulare (es. mandando un messaggio contenente le coordinate al sistema casalingo).

Controllo Remoto Casa

Sensori (gas aperto, rubinetti, porta ingresso) e attuatori (chiusura porta, persiane avvolgibili, chiusura valvole) potranno essere collegati al sistema attraverso la tecnologia wired X10, così come anche le luci e gli apparecchi domestici (TV, stereo). Gli elettrodomestici di ultima generazione offrono la possibilità di connessioni a reti wireless, e connessione a internet (per il controllo remoto del dispositivo). Se non sono disponibili tali tecnologie, essi possono essere collegati come altri alla rete X10. La pianificazione e la gestione di tutti questi apparecchi saranno disponibili attraverso un controllo remoto, che potrà essere visualizzato tramite schermo o cellulare.

2.2 Architettura utilizzata

Diverse soluzioni sono state proposte per la progettazione di sistemi domotici. Primo fra tutti è stato l'utilizzo di un bus (Figura 2.1), che collegava tutti i dispositivi, solitamente via cavo (principale esempio è l'X10).

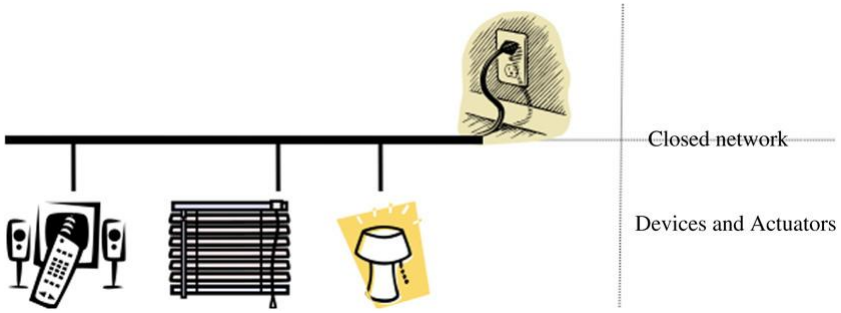

Figura 2.1: Scenario di un sistema ad un bus

Il bisogno di un controllo remoto della casa, ha portato all'adozione di un sistema centralizzato chiuso (tutti i componenti del sistema appartengono ad un unico costruttore), in cui un gateway (un controllore dei dispositivi) garantiva l'interazione con il service provider centrale (Figura 2.2).

Oggi la maggior parte dei sistemi progettati si basa sull'implementazione di una architettura centralizzata aperta (Figura 2.3), dove i vari dispositivi (anche di costruttori diversi) comunicano direttamente con il server, in cui vi sono potenza computazionale e memoria necessari per poter gestire i vari servizi. Tale architettura, concretizzando il paradigma client-server, offre buona eterogeneità e scalabilità. Questo è visibile nell'implementazione tramite web service, o tramite l'utilizzo di

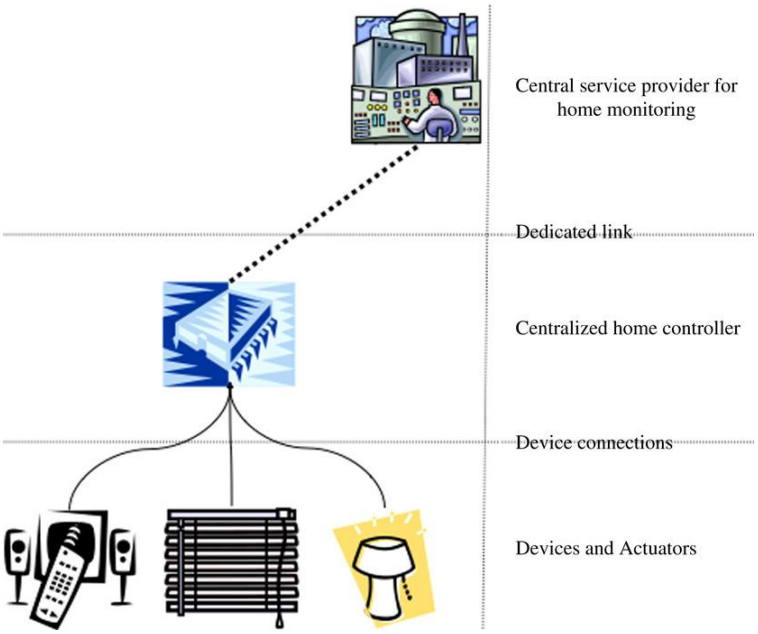

Figura 2.2: Scenario di un sistema centralizzato chiuso

framework adatti (es. OSGi). Il problema fondamentale però risiede sempre nel server centrale. Esso costituisce pur sempre un collo di bottiglia, e rappresenta un punto critico per la fault tolerance.

Utilizzando questa architettura, nel nostro sistema è possibile aumentare la tolleranza ai crash, affiancando alla macchina principale, altri dispositivi che possono agire da "surrogati", nel caso di crash del server. Un'idea è fornire al cellulare la possibilità di interfacciarsi direttamente ad alcuni dispositivi intermedi della casa (come il sistema di localizzazione dell'utente), riuscendo a garantire almeno alcuni servizi. Ovviamente tali dispositivi supplenti non avranno le stesse performance del server originale, e le funzionalità più complesse, o la gestione del database, saranno fortemente limitate.
In ogni caso, l'architettura centralizzata ha dei vantaggi non indifferenti, primo tra tutti la semplicità della progettazione. I servizi che verranno implementati risiederanno in un unico framework, con riferimenti condivisi. Inoltre l'attenzione sulla sicurezza dei dati, e degli accessi, viene focalizzata solo in un unico soggetto (server). L'unica "preoccupazione" risiede nel far comunicare i vari dispositivi (intermedi e non) con il sistema centrale.

Un'alternativa ancora poco applicata è l'architettura distribuita (Figura 2.4), detta anche P2P (peer-to-peer). In questo caso il sistema diviene più robusto rispetto

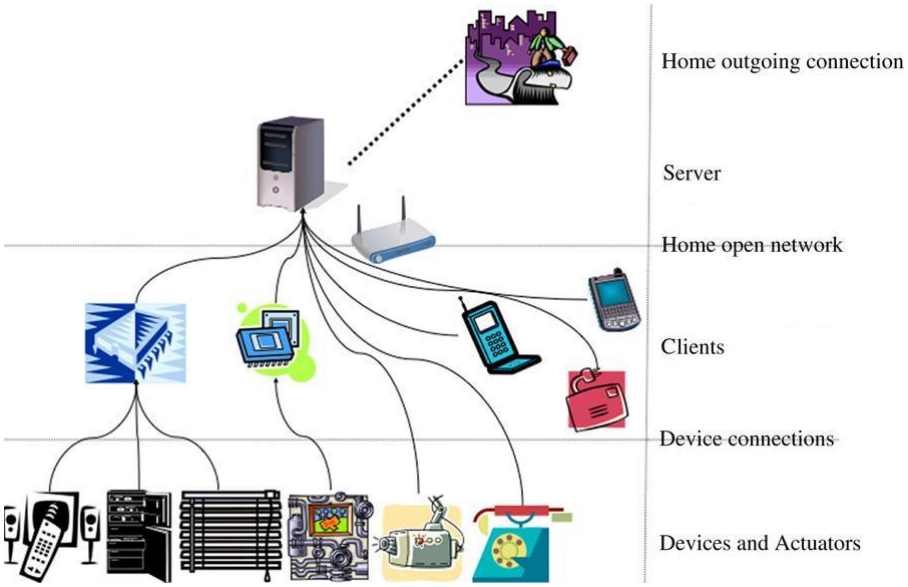

Figura 2.3: Scenario di un sistema centralizzato aperto

al fault tolerance, ma dall'altro lato diviene più complesso da progettare e gestire. I servizi diventano più efficienti, dato che possono dialogare direttamente tra loro. Un esempio può essere un allarme anti-incendio, dove i vari sensori (rilevatori di fumo, di gas, di temperatura) si scambiano informazioni, e lanciano l'allarme solo se la combinazione dei tre parametri è anormale, o pericolosa. L'eliminazione del server centrale comporta, ovviamente, l'uso di dispositivi intermedi più capaci, sia in termini di potenza di calcolo, sia in termini di memoria. Oltre all'aumento di fault tolerance, l'architettura distribuita offre la possibilità di integrare reti eterogenee, create anche in un secondo momento, come reti veicolari, reti di negozi, o comunque create da terze parti. Questa possibilità riesce ad elevare il sistema casalingo da isola stand alone a parte di un possibile sistema pervasivo più ampio. Questa visione di un scenario futuro (del prossimo futuro) ci ha portato a scegliere tale architettura per il progetto del sistema studiato.

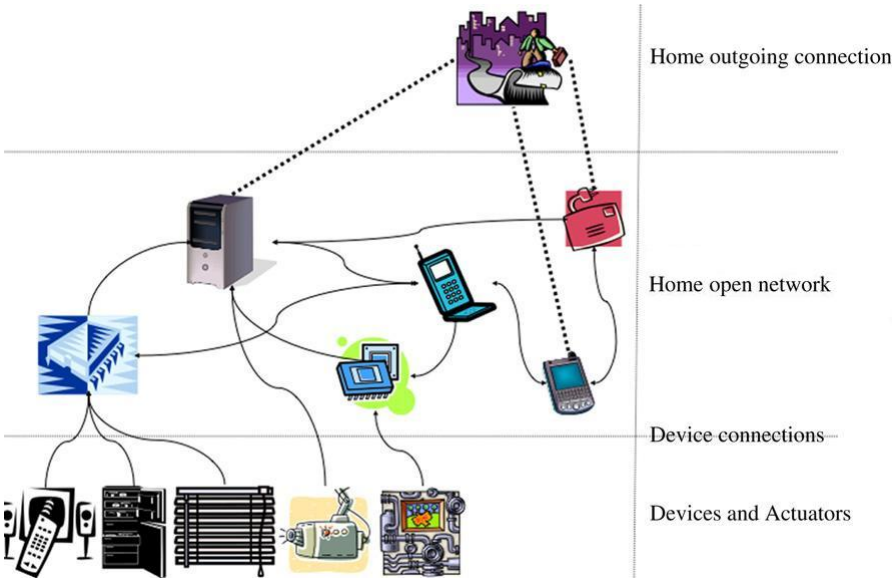

Figura 2.4: Scenario di un sistema distribuito

Capitolo 3

JINI

Per la realizzazione di questo progetto è stata scelta la tecnologia Jini, creata per costruire sistemi distribuiti con architettura service oriented. Essa estende il linguaggio di programmazione Java, per progettare reti dinamiche di servizi e client, attivi in macchine diverse.Originariamente sviluppato da Sun Microsystem, la responsabilità del suo sviluppo e continuo aggiornamento è stata trasferita ad Apache, prendendo il nome di "Rio Project".

3.1 Overview

Jini è stato pensato come tecnologia per sistemi distribuiti, e risponde molto bene alle esigenze dei sistemi ubiqui. Per prima cosa tale architettura è plug & play. I componenti del sistema possono infatti essere aggiunti o rimossi a run-time, e questo comporta un livello molto alto di dinamicità. E' robusta rispetto ai crash o alle disconnessioni. Se alcune risorse non sono più raggiungibili, Jini provvede ad eliminare i servizi non più attivi, e rileva automaticamente se risorse sono state aggiornate o aggiunte al sistema. Questo lo rende adatto a sistemi costruiti su reti wireless, affetti da una frequenza di fault molto alta (rispetto ad altri tipi di sistemi). Jini, inoltre, permette al sistema di accrescere in modo modulare e scalabile, fornendo la possibilità di collegare tra loro più Jini community (reti implementate utilizzando Jini). Dato che Jini si basa su Java, esso è indipendente dalla piattaforma su cui esso opera (Figura 3.1). L'unico requisito, infatti, è che in ogni dispositivo coinvolto sia attiva una Java Virtual Machine.

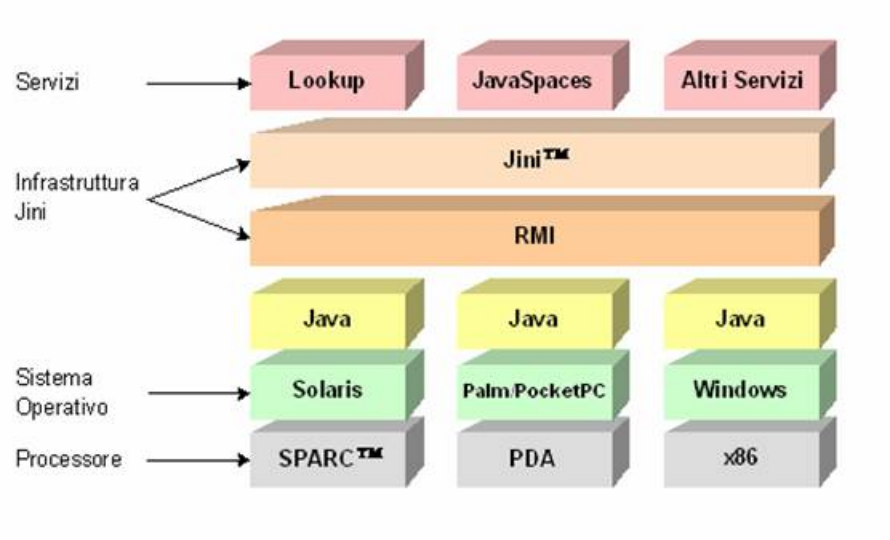

Figura 3.1: Schema dell'architettura della tecnologia JINI

3.2 Modello

Il modello di Jini si basa su tre attori principali: il service, il client e il Lookup Service (LUS).

Service

Il service è un'entità che implementa un qualche tipo di servizio (come un servizio di stampa, collegato ad una stampante), che può essere utilizzato da un client, o da un altro service. Il servizio produce un Service Object, che costituisce la sua parte mobile (e che verrà usato dal client), ed è formato da un Proxy e dalle Entry. Il Proxy è l'interfaccia che userà chi andrà ad usufruire del servizio, e a volte esso rappresenta il servizio stesso. Le Entry sono un insieme di attributi che servono per specificare il servizio, come nome, parametri particolari (es. stampante a colori o solo bianco e nero) o il fornitore degli attributi.

Lookup Service

Ogni servizio, per essere rintracciabile, deve registrarsi in un Lookup Service (LUS). Esso, quindi, risulta un registro centrale dove rivolgersi per avere informazioni o accesso ai servizi. Il LUS è strutturato e gestito come un servizio, ma, essendo

quello principale dell'architettura Jini, deve essere il primo della community. Per aumentare la robustezza e l'affidabilità del sistema, vi possono essere (ed è consigliato che ci siano) più LUS. I servizi, quando diventeranno attivi, si registreranno in tutti i LUS disponibili al momento (e anche in quelli che compariranno in un secondo momento).

Client

Un client è, in generale, un'applicazione che utilizza un servizio. In Jini tale client può essere a sua volta utilizzato come servizio da un secondo client (e così via). Non c'è infatti una rigida distinzione tra i due componenti. Tutti i servizi possono essere visti come client, se pensiamo che tutti hanno la necessità di utilizzare almeno un service (il LUS). I client cercano i servizi o conoscendo a priori l'interfaccia che utilizzeranno (e quindi i metodi che essi mettono a disposizione), o attraverso il matching delle Entry. Questo è il motivo per cui gli sviluppatori di Jini consigliano di utilizzare interfacce standard, lasciando completa libertà sulla loro implementazione. Questo comporta una maggiore integrazione tra servizi e client sviluppati da soggetti diversi. In ogni caso tale problema può venire superato tramite un'interazione con l'utente, che può scegliere e comprendere interfacce non conosciute a priori dal programma client.

3.3 Interazioni Fondamentali

Service, Client e Lookup Service interagiscono tra loro tramite alcune operazioni fondamentali, ognuna delle quali costituisce un protocollo di comunicazione: Discovery, Join e Lookup.

Discovery

Il processo di Discovery (Figura 3.2) viene utilizzato dai client e dai service per rintracciare l'esistenza di un LUS. Esso è responsabile dello spontaneo costituirsi delle Jini Community.

Non esiste un unico protocollo di discovery, ma diversi, a seconda delle esigenze:

- *Multicast Request Protocol* : è usato dai service e dai client per trovare i LUS. E' basato sul multicast UDP e viene utilizzato in reti LAN, in quanto esso (il multicast) ha una portata limitata alle intranet;

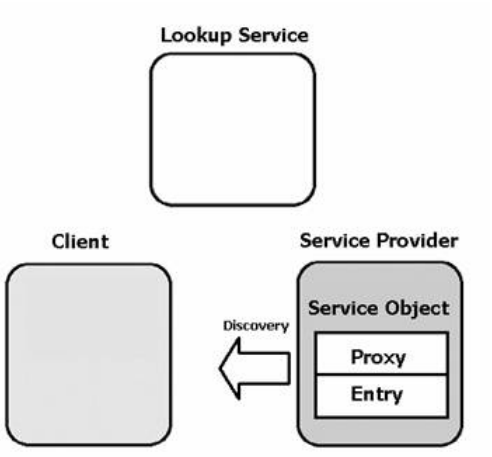

Figura 3.2: Operazione di Discovery

- *Multicast Announcement Protocol* : viene usato dai LUS per annunciare la loro presenza in una LAN. Anch'esso si basa su UDP;
- *Unicast Request Protocol* : viene utilizzato da un servizio Jini quando deve contattare un LUS a qualsiasi distanza, ma di cui conosce l'indirizzo. E' basato sul protocollo TCP, e viene usato quando ci si vuole registrare in un LUS situato all'esterno della propria LAN.

Il risultato finale di tale processo è un riferimento ai LUS che può essere utilizzato dai service e dai client per determinare quali servizi sono disponibili nella community.

Join

Quando un service si registra in un Lookup Service, utilizza un processo chiamato Join (Figura 3.3). In questa fase il LUS registrerà il nome del servizio, immagazzinando il Service Object.

Come veniva descritto nel paragrafo dedicato alla definizione di Service, tale Service Object è formato da un Proxy e da Entry. Le Entry sono generalmente delle stringhe che contengono informazioni riguardo al servizio, ma in realtà un attributo può essere costituito da un qualsiasi oggetto Java serializzabile. Il Proxy rappresenta la parte più consistente, in quanto ci permette di interagire con il servizio. Questo ci permette di usare un servizio o un dispositivo senza l'installazione di driver o software. I particolari dell'interazione tra service e proxy sono completamente a

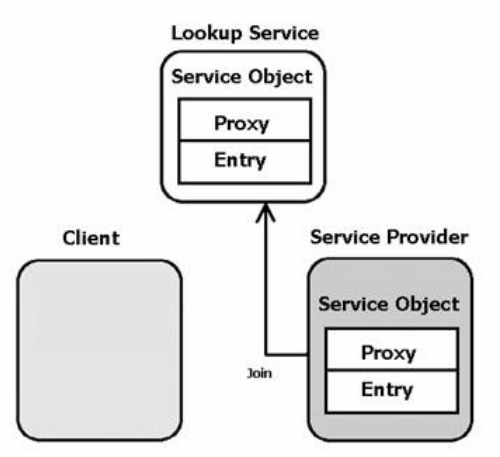

Figura 3.3: Operazione di Join

discrezione del creatore del servizio. Tra le implementazioni più comuni abbiamo le seguenti:

- l'oggetto proxy è il servizio. Esso contiene il servizio stesso e implementa in sé tutte le funzionalità del servizio. Quando ciò avviene non si ha la necessità di comunicare con il service. Un esempio può essere un traduttore di lingua, dove il proxy contiene tutte le coppie parola/traduzione;
- il proxy è un'interfaccia del servizio remoto. Qui l'elaborazione avviene tutta nel lato server, mentre il proxy serve solo per raccogliere le invocazioni e comunicare con il servizio. La comunicazione può avvenire tramite socket, o tramite JRMP, JERI o altri protocolli.

Lookup

Lookup è il processo attraverso il quale il client interroga il LUS per trovare il servizio richiesto (Figura 3.4). Il LUS restituisce il proxy necessario per poterlo utilizzare.

La ricerca di servizi da parte del client può avvenire in tre modi diversi:

- tramite service ID. Esso è un identificativo che viene dato al servizio al momento della sua registrazione (Join). Esso è univoco e viene utilizzato ad ogni operazione di Join. Il client, se a conoscenza di tale ID, può richiederlo direttamente al LUS;

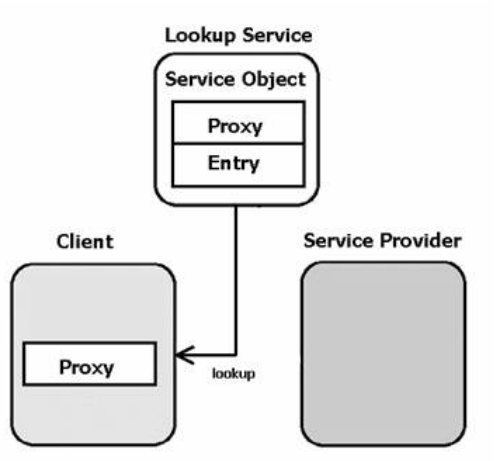

Figura 3.4: Operazione di Lookup

- ricerca tramite il tipo di proxy. Il client specifica una o più interfacce Java, e il LUS restituisce un insieme di proxy, istanze delle classi specificate;
- ricerca tramite entry. Il client specifica una o più entry, e il LUS ritorna tutte le istanze di proxy che matchano con almeno un attributo.

Una volta che il proxy è stato scaricato, il client lo usa come "front-end" per comunicare con il "back-end" del service (Figura 3.5), secondo l'implementazione utilizzata.

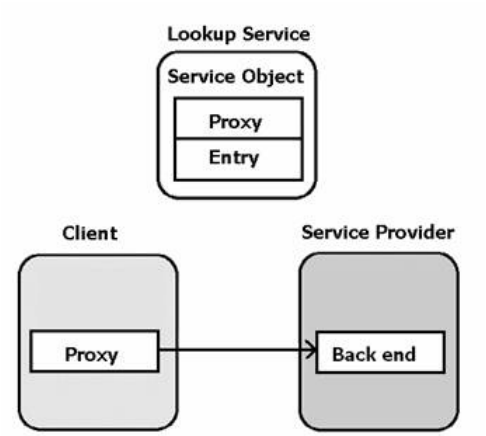

Figura 3.5: Comunicazione tra client e service

Capitolo 4

Servizio Reminder

Dopo aver discusso la composizione del sistema studiato, l'architettura sottostante e la tecnologia usata, si passa all'implementazione di un servizio. Tale servizio è chiamato Reminder, ed è l'emblema delle funzionalità richieste : dalla gestione del database, all'interazione tra componenti di Jini, fino ad arrivare alla possibilità futura di integrare parti che andranno ad arricchire l'offerta del sistema.

4.1 Scelta del servizio

Si è scelto di concentrarsi più sulle funzionalità del servizio offerto all'utente, piuttosto che inoltrarsi nelle capacità offerte dalla tecnologia Jini. Quest'ultima propone diversi approfondimenti nella gestione delle credenziali dell'utente, nella diversificazione dei protocolli di accesso remoto ai service, nella definizione del Lookup Service, e nella gestione dei leasing, solo per citarne alcuni. Questa seconda via sarebbe stata dispersiva, data la vastità degli argomenti trattabili. In quest'ottica va visto l'utilizzo di una versione già implementata del Lookup Service, contenuta nel pacchetto scaricabile di Jini, chiamata *reggie*. Oltre a questo si è deciso di concentrare l'implementazione su un singolo servizio, il Reminder di medicinali. Ad esso è stato accorpato anche il Reminder di Eventi, in quanto sfaccettature diverse della stessa funzionalità. Per questo come prima azione, sono state analizzate le varie tipologie di eventi che possono essere ospitate dal servizio, e che devono rispondere il più possibile alle esigenze degli utenti. In ogni caso l'ossatura dell'architettura, che ospita l'implementazione del servizio che vedremo in seguito, è la medesima per ogni servizio descritto nei capitoli precedenti. Essa infatti analizza la comunicazione tra service Jini, il collegamento e la gestione di database, e l'interfacciamento grafico con l'utente del servizio.

Tipologie di eventi

Il primo passo da compiere è la definizione del tipo di eventi che l'utente potrà inserire, e che il servizio dovrà gestire. La suddivisione poi avrà conseguenze anche nella tipologia di avvisi da presentare all'utente. Gli eventi inseribili possono essere della categoria : medicine, routine o appuntamenti.
Il primo tipo di eventi, rappresenta il tipico bisogno di ricordarsi di assumere una medicina ad un orario prestabilito, ma che può essere anche procrastinato di qualche minuto. Per questo motivo l'utente vedrà recapitato un solo avviso, all'ora prestabilita. Dato che la somministrazione dei medicinali, solitamente, segue una periodicità settimanale (ogni giorno, due volte alla settimana, etc...), l'utente dovrà specificare per quali giorni della settimana l'evento dovrà essere ripetuto.
La categoria *routine*, come la successiva, fa parte, concettualmente, del servizio Reminder Eventi. Un evento *routine* è un'azione che si svolge solitamente a cadenza settimanale, come l'andare in palestra. Anche in questo caso l'utente dovrà specificare in quale giorno della settimana tale evento verrà ripetuto. A differenza dei medicinali però, gli avvisi prodotti sono due. Il primo verrà recapitato all'utente un'ora prima della scadenza, e il secondo all'ora prestabilita. Si è scelto di far pervenire questa forma di "avviso di chiamata" un'ora prima perché è sembrato un ottimo compromesso tra il bisogno di avvertire per tempo l'utente (in modo che possa avere il tempo di prepararsi), e il fatto che troppo tempo tra il preavviso e l'effettivo orario di scadenza possa portare al dimenticarsi dell'evento.
L'ultimo tipo di eventi è l'*appuntamento*. Esso si differenzia dagli altri due, per il fatto di essere un'azione isolata, non ripetuta settimanalmente. Per questo l'utente deciderà il giorno in particolare in cui tale evento dovrà essere ricordato. Come per la "routine", anche questa tipologia crea due avvisi, uno alla mattina (inizio giornata) e uno un'ora prima della scadenza. Il primo avviso serve per ricordare all'utente, all'inizio della giornata, un appuntamento che non è solito fare. Il motivo del secondo avviso è il medesimo descritto nella categoria *routine*. L'avviso all'ora prestabilita è stato considerato superfluo, in quanto, solitamente, tali eventi si riferiscono a visite mediche o comunque ad appuntamenti fuori casa. Un esempio esplicativo di *appuntamento* è il seguente. L'utente inserisce l'evento "Visita chirurgica alla spalla in ospedale alle ore 16:00". Il sistema creerà sia un avviso di promemoria alle 9:00 (per ricordargli l'impegno della giornata), sia un avviso alle 15:00 (in modo che si prepari e che possa avere il tempo di arrivare in ospedale in tempo). Un avviso alle 16:00 sarebbe superfluo in quanto a quell'ora l'utente dovrà essere già nel luogo prestabilito, e quindi non necessita di promemoria.

4.2 Composizione del servizio

Il Reminder si compone di varie e diverse parti. Gli eventi e l'informazione relative agli avvisi da mandare sono contenuti in un database (db Reminder). Il core del servizio è composto da tre service Jini (Calendario, Eventi e Screen) che gestiscono l'accesso al database, la temporizzazione degli eventi, le modalità di invio degli avvisi e il controllo del monitor. A questi viene affiancata una interfaccia grafica che agevola il compito dell'utente nel visualizzare, aggiungere o eliminare gli eventi memorizzati nel database.

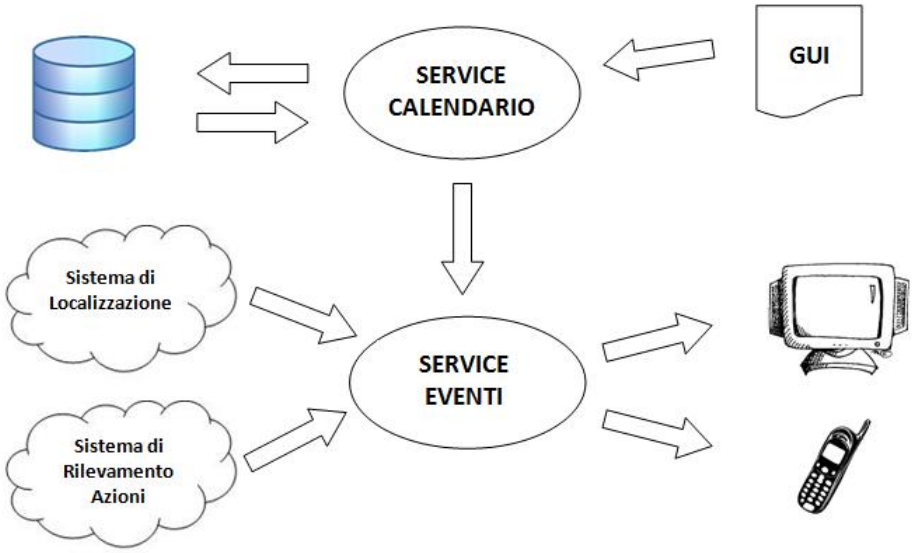

Figura 4.1: Servizio *Reminder*

4.2.1 Database *reminder*

Il database è diviso in tre tabelle : Agenda, Settimanale e Giornaliero. Questa divisione è stata creata per dividere le varie tipologie di eventi (e relativi avvisi). In realtà le varie tabelle non contengono gli eventi, bensì gli avvisi relativi all'evento stesso.
La tabella **Agenda** contiene tutti gli eventi del tipo Appuntamento, con data e ora. Ogni evento di questo tipo ha due record nella tabella (che corrispondono ai due avvisi). Ogni record ha questi campi : ID (int, auto incrementale), EVENTO (text), DATA (date), ORA (time) e TIPO (int).

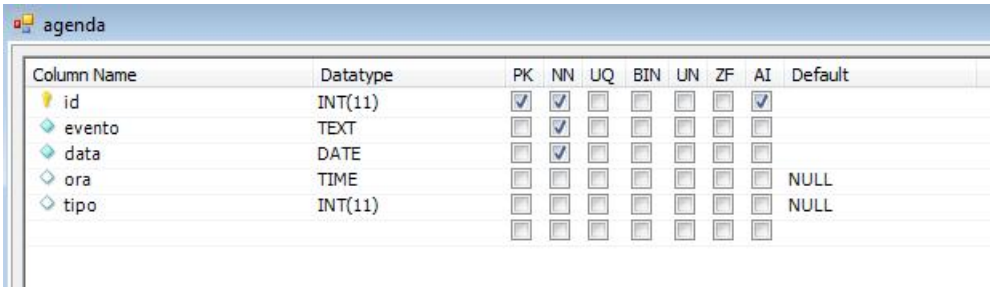

Figura 4.2: Database *Agenda*

La tabella **Settimanale** contiene tutti gli eventi del tipo Medicine o Routine. Ogni evento ha un numero di record parti agli avvisi da recapitare all'utente. I campi sono : ID (int, auto incrementale), EVENTO (text), GIORNO (int), ORA (time), TIPO (int). Il campo Giorno, contiene un numero (da 1 a 7) corrispondente ad un giorno della settimana (domenica = 1).

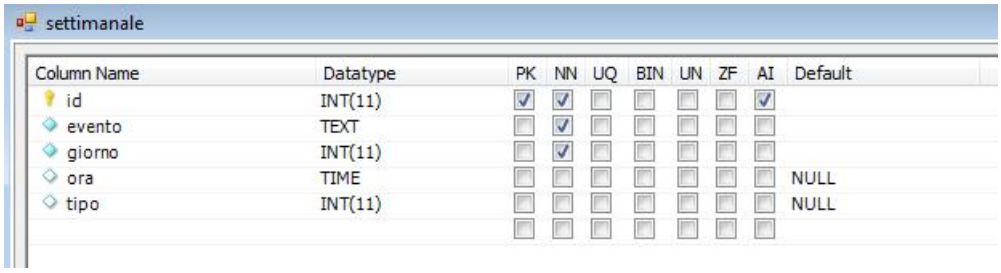

Figura 4.3: Database *Settimanale*

La tabella **Giornaliero** contiene tutti gli avvisi relativi al giorno attuale. Essa è composta da tutti gli eventi delle tabelle Agenda e Settimanale, relativi al giorno in questione, e viene riempita all'inizio di ogni giornata. I suoi campi sono : ID (int, auto incrementale), EVENTO (text), ORA (time), TIPO (int).

La chiave primaria di ogni tabella è il campo ID. ORA si riferisce all'orario in cui l'avviso contenuto nel campo EVENTO dovrà essere recapitato. TIPO funge da flag, per indicare la presenza di un ulteriore avviso che si riferisce allo stesso evento, e che avrà un ID consecutivo al suo. In questo modo la visualizzazione degli eventi risulta più pulita, e la loro eliminazione più agevole.

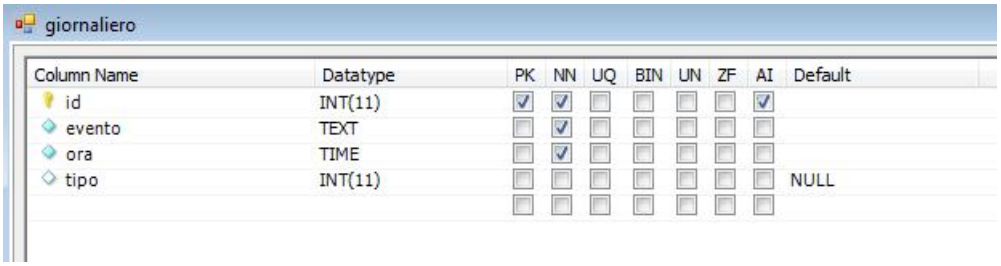

Figura 4.4: Database *Giornaliero*

Listing 4.1: Interfaccia *InterfaceC*

```
public interface InterfaceC extends Remote {
        public boolean aggiungiSettimana(String appunt, String ora, int
            giorno, int choose) throws RemoteException;
        public boolean aggiungiAgenda(String appunt, String ora, String
            data) throws RemoteException;
        //elimino record da DB
        public boolean eliminaSettimana(int id, boolean doppio) throws
            RemoteException;
        public boolean eliminaAgenda(int id, boolean doppio) throws
            RemoteException;
        //ritorno DB per visualizzazione
        public Vector viewDBGiorno() throws RemoteException;
        public Vector viewDBSettimana() throws RemoteException;
        public Vector viewDBAgenda() throws RemoteException;
}
```

4.2.2 Service *Calendario*

Il Service Calendario è il più articolato dei tre servizi implementati con la tecnologia Jini, ed è responsabile della gestione del database contenente gli eventi e i relativi avvisi. L'interfacciamento con il database MySQL utilizzato, si concretizza nell'implementazione dei metodi di connessione, di disconnessione e di inserimento di query (di aggiunta, eliminazione e selezione). Questi ultimi sono dichiarati come private e vengono usati dai metodi pubblici che rendono disponibile l'accesso e la modifica ai record da parte dell'utente. Il service implementa l'interfaccia *InterfaceC* (4.1), che rende accessibili da remoto i metodi sopracitati.

Nel metodo *aggiungiSettimana*, il parametro *choose* sta ad indicare se l'evento da aggiungere nella tabella Settimanale è di tipo Medicina (*choose* = 1) o di tipo Routine (*choose* = 2). Nei metodi di eliminazione dei record, il parametro booleano

Listing 4.2: Interfaccia *InterfaceE*

```
public interface InterfaceE extends Remote {
        public boolean aggiungiEvento(String nome, String ora) throws
            RemoteException;
        public boolean setOccupato(boolean occ) throws RemoteException;
        public boolean setInCasa(boolean casa) throws RemoteException;
}
```

doppio indica la presenza, o meno, di un ulteriore avviso riferito all'evento indicato dal parametro ID (se *doppio* = true, allora gli avvisi da eliminare saranno due, altrimenti solo uno). I metodi di visualizzazione delle varie tabelle del database restituiscono una struttura dati di tipo Vector. Ogni elemento di tale struttura è uno string array, e rappresenta un record. Il Vector restituito è ordinato secondo data/giorno della settimana e orario.
Un aspetto più interessante di questo service è la temporizzazione degli avvisi. Per fare questo si deve per prima cosa avere la concezione di "cambio di giorno". In tal senso va vista l'implementazione della classe privata *ThreadGiorno*. All'avvio del service, viene creato un thread di tale classe, che memorizza il giorno attuale e rimane in attesa, per il tempo che rimane alla fine della giornata. Successivamente viene popolata la tabella Giornaliero con gli avvisi relativi. Alla fine dell'attesa, viene creato un nuovo thread, si pulisce la tabella (utilizzando *truncate*), e ricomincia il ciclo. Ad ogni ciclo giornaliero, vengono presi tutti gli eventi relativi al giorno, e viene associato ad ogni avviso un thread (della classe *ThreadEvento*), che terrà conto dell'orario memorizzato in tabella, e resterà in attesa fino al raggiungimento dell'ora prevista. Alla scadenza del tempo atteso, il thread comunica al Service Eventi (attraverso l'apposito metodo remoto) che l'avviso può essere segnalato all'utente.

4.2.3 Service *Eventi*

Il Service Eventi gestisce la modalità e il momento per recapitare l'avviso all'utente. Nell'idea iniziale questo servizio doveva comunicare con due altri soggetti del sistema : il sistema di localizzazione dell'utente e il sistema di riconoscimento delle azioni. Essi non sono stati implementati, ma la loro possibile futura azione è stata prevista. L'interfaccia che viene implementata da questo service è InterfaceE (4.2).

Il metodo *setOccupato* imposta il valore della variabile *isOccupato* contenuta nel servizio. Se questa variabile booleana è impostata a *false*, allora gli avvisi possono essere recapitati all'utente senza alcun ritardo. In caso contrario, verranno inviati

Listing 4.3: Interfaccia *InterfaceScreen*

```
public interface InterfaceScreen extends Remote {
    public boolean avvisoEvento(String mess) throws RemoteException;
}
```

non appena il valore non verrà settato a *true*. Questo metodo risponde alla possibile presenza di un sistema di riconoscimento delle azioni, che dovrà decidere se l'utente è interrompibile, o meno, nelle sue azioni giornaliere. Un futuro passo può essere impostare soglie di interrompibilità, e associare ad ogni avviso, un grado di priorità. Se un avviso ha una priorità più alta della corrente soglia, allora esso verrà recapitato, altrimenti verrà procrastinato. Il metodo *setInCasa*, invece, risponde all'esigenza di far comparire l'avviso in un mezzo il più possibile vicino all'utente. Questo vuol dire che se la persona non è in casa, l'avviso dovrebbe essere recapitato sul telefonino, mentre se non è in casa, dovrebbe venir inviato ad un monitor il più vicino possibile a lui. Per ora le possibilità disponibili sono solo in casa, o fuori casa (variabile booleana *inCasa*). L'implementazione si è focalizzata solo sull'utilizzo di un monitor disponibile in casa.
Il service contiene una struttura dati di tipo Vector, che contiene tutti gli avvisi in coda (in ordine FIFO), che devono ancora essere comunicati all'utente. Un thread controlla se ci sono avvisi pendenti in lista, e se la variabile *isOccupato* è impostata a *false*. Nel caso questo avvenisse, il service inoltra questi avvisi allo schermo, in modo che l'utente possa vederli, in caso contrario, il thread, invocando un *wait()*, rimane in attesa di nuovi eventi in lista, o della disponibilità dell'utente (*isOccupato* = *false*).

4.2.4 Service *Screen*

Il Service Screen gestisce la visualizzazione degli avvisi, di un monitor. Esso implementa l'interfaccia *InterfaceScreen* (4.3), che pubblica un unico metodo : *avvisoEvento*(String mess). Questo metodo fa comparire a schermo una form grafica, in cui si comunica all'utente il messaggio contenuto nel parametro *mess*.

Screenshot di avvisi recapitati all'utente sono i seguenti (Screenshot 4.5 e 4.6).

4.2.5 Interfaccia grafica

Per la gestione del database, è stata implementata una interfaccia grafica, che agevolasse l'utente nell'aggiunta di eventi, nella visualizzazione dei farmaci da

Figura 4.5: Preavviso di un evento di routine

Figura 4.6: Avviso di un evento di routine

assumere, delle azioni di routine e degli appuntamenti, e nell'eliminazione degli stessi. Per questo si sono creati due frame, a cui competono l'inserimento di eventi e la visualizzazione e l'eliminazione. Nella figura 4.7 vediamo il menù principale, da cui si può accedere sia alla finestra di inserimento degli eventi di figura 4.8 (nell'esempio si può vedere l'inserimento di un evento di routine), sia alla finestra di visualizzazione, ed eliminazione, di figura 4.9 (dove si può vedere l'elenco settimanale degli avvisi relativi ai medicinali da assumere e degli appuntamenti di routine da compiere).

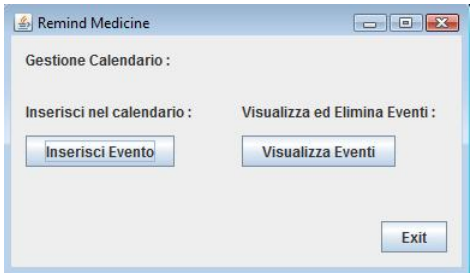

Figura 4.7: Finestra iniziale della GUI

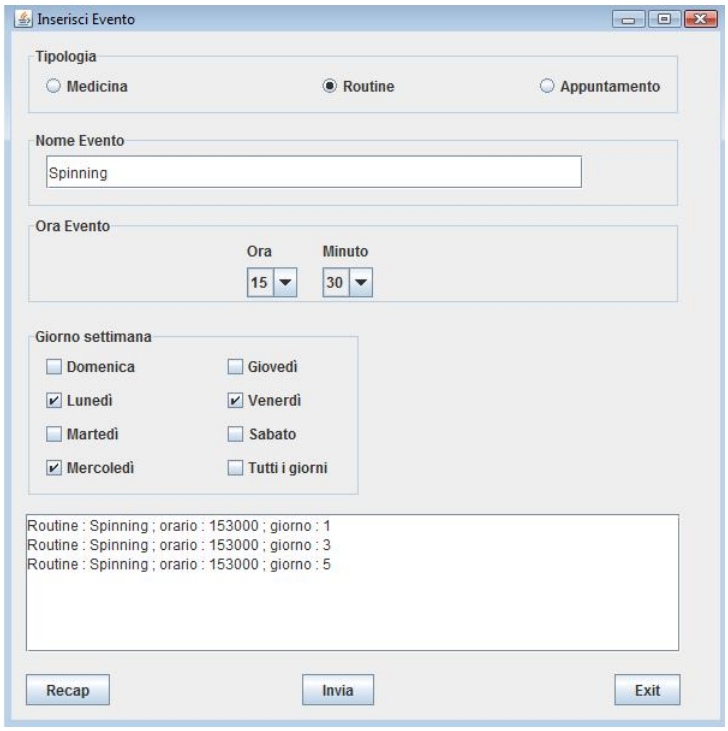

Figura 4.8: Inserimento eventi

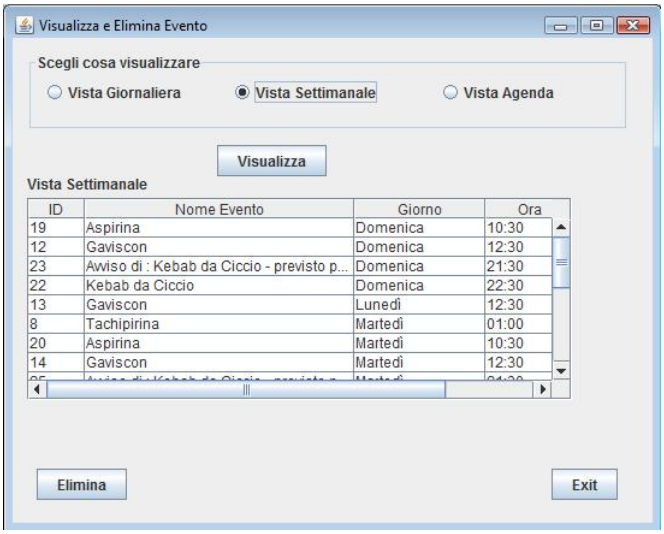

Figura 4.9: Visualizzazione ed eliminazione eventi

Capitolo 5

Conclusioni

L'obiettivo della tesi trattata è di calare il paradigma della computazione ubiqua nell'ambito dell'assistenza alle persone. Unire i temi molto attuali dell'Ubiquitous Computing e dell'HealthCare. Per fare ciò è stata utilizzata la tecnologia Jini, che fornisce gli strumenti per implementare l'architettura di un sistema ubiquo, e al contempo dà garanzie in fatto di tolleranza ai crash e ai fault, e di apertura verso integrazioni di sistemi futuri.

Si è visto che la scelta di un architettura distribuita, e in particolare peer-to-peer, è stata e sarà vincente sia per il qui presentato sistema, sia per il futuro dell'intera categoria di computazione pervasiva. E' impensabile che nel futuro possano esistere sistemi completamente centralizzati. Tale architettura può andare bene se pensata in un unico ambiente (es. domotica), ma è completamente inadeguata per una visione globale e futuribile. Infatti l'analisi e la definizione dei servizi ha messo in luce il bisogno di integrazione con sistemi e reti di terze parti. Uno degli scenari più rappresentativi del modello ubiquo, è la disponibilità dei dispositivi in ogni luogo (ubiqua appunto), che devono relazionarsi tra loro in maniera fluida. E' auspicabile quindi che i sistemi progettati siano il più aperti e il più dinamici possibile, per non far pesare all'utente il cambio di contesto (es. passaggio tra una rete domestica e una veicolare).

Da queste esigenze è stato scelto di usare una tecnologia che fa parte del mondo Java (e negli ultimi tempi passata sotto l'egida di Apache), notoriamente un ambiente che fa della portabilità, della integrazione con terze parti e della openness i suoi punti di forza. Il modello sottostante si è rivelato relativamente semplice nella sua struttura, ma non per questo è povero di funzionalità. Al contrario Jini pone attenzione ai problemi propri dei sistemi distribuiti, quali la fault tolerance, la necessità di protezione dei dati, e la sicurezza dell'autenticazione degli utenti.

Il servizio implementato (Reminder) ha coperto i punti critici di ogni servizio esposto del sistema : dalla gestione con un database, alla comunicazione tra servizi server e servizi client. L'attenzione all'ossatura del servizio, a discapito della definizione di altre componenti, non ha fatto trascurare il possibile sviluppo del sistema. Infatti sono stati definiti i meccanismi di interazione con servizi non ancora sviluppati, ma necessari per un aumento delle funzionalità del sistema. La possibilità di integrazione del sistema di riconoscimento delle azioni e del sistema di localizzazione dell'utente infatti si può notare nella definizione del Servizio Eventi. Questi due sistemi possono essere definiti a sé (vari esempi si possono trovare nella letteratura) e poi facilmente accorpati al sistema in esame. Stesso discorso per le parti che concernono elementi hardware, come la gestione degli RFID Reader, o il collegamento tra dispositivi casalinghi (elettrodomestici, luci, etc).

Appendice A

Programmando in Jini

A.1 Installazione

Jini è un'insieme di librerie che estendono la tecnologia Java, e quindi ha come requisito fondamentale l'installazione della sua Virtual Machine (nella versione JDK). Non ha particolari problemi sulla versione utilizzata (infatti Jini è stato pensato anche per cellulari, e per dispositivi con capacità di memoria e computazione di molto inferiori ad un computer desktop). Da Java eredita anche l'indipendenza dal sistema operativo e dall'architettura sottostante, quindi l'installazione della VM è l'unico prerequisito all'installazione.
E' possibile scaricare Jini sia attraverso un pacchetto autoestraente, sia attraverso un archivio semplicemente da scompattare, dal sito ufficiale di Jini www.jini.org/Category:Introduction_to_Jini (o www.jini.org/Category:Getting_Started).
Dopo aver installato la VM di Java e aver installato/scompattato le librerie di jini (non è discriminante in quale cartella lo si è fatto), Jini è pronto per essere utilizzato. Di seguito vedremo in dettaglio i servizi (web server e lookup service) da rendere attivi prima di poter caricare i nostri, e le impostazioni da settare (configurazione, codebase e policy). Si anticipa solo che all'attivazione di ogni servizio, dovrà essere specificato l'indirizzo del file di configurazione e del file di policy (non è essenziale che essi siano in una cartella prestabilita, anche se è consigliato per mantenere ordine tra i file).

A.2 HTTP Server e Reggie

Http Server

Il primo requisito che viene richiesto da Jini è un semplice Web Server. In Jini infatti i servizi devono poter scaricare pezzi di codice dei servizi che vogliono usare. La vera trasmissione del codice avviene tramite il protocollo HTTP, ma è sufficiente anche un semplice server (che viene anche messo a disposizione dalle librerie di Jini). L'importante però è che in ogni macchina in cui sono attivi dei servizi che forniscono codice scaricabile, sia attivo anche un HTTP server.
Di default l'HTTP Server, reso disponibile da Jini, opera sulla porta 8080. Se in tale porta è già attivo un altro Web Server, basta cambiare il numero di porta direttamente da consolle, all'avvio del server, passando l'opzione `-port`. Oltre al numero di porta, all'avvio bisogna specificare la cartella, o le cartelle, in cui il codice scaricabile si può trovare. Esse vengono specificate nell'opzione `-dir`. In queste directory dovranno comparire i proxy dei servizi; si veda come esempio la cartella *lib-dl* all'interno della home di Jini (solitamente C:\jini). L'opzione `-verbose` serve solo per seguire passo passo le operazioni richieste ed eseguite dal server. Il comando finale sarà quindi nella forma :

```
%jinihome%\java -jar lib\classserver.jar -port 8080 -dir dir1;dir2;
                                -verbose
```

Reggie

Il punto centrale dell'architettura Jini è rappresentato dal Lookup Service. Esso "pubblica" i servizi, e "comunica" ai client l'esistenza di service attivi. Jini propone una implementazione base di questo servizio centrale, chiamata *reggie*. Essa si trova nella cartella *scripts* situata in

%jinihome%\source\src\com\sun\jini\example\hello .

Esistono più varianti di tale servizio, a seconda delle esigenze di sicurezza richieste dall'occasione. Le differenze tra le versioni sono spiegate all'interno del file `index.html` della stessa directory *hello*. Il LUS scelto per questo progetto è *jrmp-reggie*. E' possibile avviare tale servizio attraverso il file `.bat` omonimo che si trova nella stessa cartella.

Listing A.1: Esempio file di configurazione (per il lato server)

```
/* Configuration source file for jrmp server */
import net.jini.discovery.LookupDiscovery;
import net.jini.jrmp.JrmpExporter;

//inizio impostazioni relative alla parte com.sun.jini.asService

com.sun.jini.asService {

    //implementazione scelta per l'oggetto Exporter
    exporter = new JrmpExporter();

    //definizione del gruppo a cui apparterrà il service
    private groups = new String[] { "nonsecure.hello.example.jini.sun.com" };

    discoveryManager = new LookupDiscovery(groups, this);

}
//fine com.sun.jini.asService

//eventuale inizio relativo ad una seconda parte
//lo stesso file di configurazione può essere usato per più service
...
```

A.3 Avvio di un service

All'avvio di un service è necessario specificare alcuni parametri, quali le impostazioni sulla sicurezza (File di policy), le configurazioni iniziali del service (File di configurazione) e l'url in cui trovare il codice scaricabile (Codebase).

File di Configurazione

La maggior parte dei servizi JINI ha parametri configurabili, che è meglio determinare nella fase di deployment (a runtime quindi), piuttosto che impostarli nel listato dell'applicazione. Tipicamente questo include che gruppi di LUS usare, la durata dei lease, i codebase, etc. A questi sono stati aggiunti la selezione dell'implementazione di RMI usata, i vincoli per effettuare chiamate remote, che protocolli di trasporto o che soggetti di autenticazione usare, etc. Con la versione 2.x viene introdotto un nuovo modello, il *config model*, in cui tutti questi parametri vengono impostati a runtime, prendendo i valori inseriti in un file di configurazione (vedi A.1). In questo modo si possono dividere le figure del developer (chi scrive il programma, e che decide che parametri possono essere impostati successivamente) e del deployer (chi usa il programma e lo personalizza in base alle proprie esigenze).

Codebase

All'avvio di ogni service, si deve specificare l'URL in cui il codice scaricabile è stato salvato. Questo luogo viene chiamato codebase, e ospita i proxy, che verranno scaricati dai servizi client, per poter interagire da remoto con i servizi server. La composizione di questi proxy è variabile, a seconda del protocollo di interazione usato (JRMP, JERI, etc) e del tipo di protezione che si vuole dare al proprio servizio (autenticazione, rispetto di parametri specifici, etc). Nel nostro caso, i proxy sono composti dalle interfacce implementate e da uno stub della classe del servizio. Questo stub è creato tramite l'applicazione rmic di Java.

Policy

Oltre al file di configurazione e all'url del codebase, all'avvio del service si deve passare anche un file, dove vengono specificate le operazioni che può effettuare il servizio. In fase di development di solito viene lasciata completa autonomia al programma (AllPermission), mentre in fase di deployment le autorizzazioni vengono ridotte, per motivi di sicurezza del sistema.

A.4 Classi significative delle librerie Jini

Ora analizziamo come vengono utilizzate la classi messe a disposizione dalle librerie di Jini.

Configuration

Nella parte iniziale del servizio, lo sviluppatore crea un oggetto in cui vengono memorizzate tutte le informazioni contenute nel file di configurazione (vedi A.2). Questo oggetto è creato da un'istanza dell'interfaccia *net.jini.config.Configuration*, ed è composto da più Entry (una per ogni elemento configurato).
net.jini.config.Configuration definisce un'interfaccia che fornisce oggetti necessari a configurare istanze come Exporter. Le entry di tale interfaccia sono definite da un *component* e un *name*. I metodi che reperiscono tali entry possono specificare un valore di default, nel caso l'entry cercata non compaia nel file. Il component identifica l'oggetto di cui si vuole configurare il comportamento, mentre il name specifica le diverse entry configurate per tale oggetto. Un file di configurazione può essere usato da più servizi (che possono implementare funzioni diverse), o da un solo servizio che crea più oggetti

Listing A.2: Acquisizione delle configurazioni iniziali (tratta dal Service Eventi)

```
//variabile in cui vengono memorizzate le impostazioni iniziali
protected final Configuration config;

//estratto del metodo main
public static void main(String[] args) throws Exception {
      servE = new ServiceE(args);
      ...
      }

//costruttore che inizializza la variabile config, che verrà interrogata in seguito
     tramite getEntry
protected ServiceE(String[] configOption) throws ConfigurationException {
      config = ConfigurationProvider.getInstance(configOption, (this.getClass()).
          getClassLoader());
      }
```

Exporter

Con la versione 2.0 e l'introduzione di JERI sono stati introdotti gli Exporter, responsabili della generazione di un riferimento remoto ad un oggetto remoto (il servizio). I dettagli del comportamento di export e di unexport (nel caso si volesse tornare nei propri passi), che includono sia i protocolli di comunicazione usati per le chiamate remote, sia semantiche aggiuntive per tali invocazioni, sono definiti a seconda della particolare implementazione scelta. Sono forniti vari exporter, tra i quali i più usati sono:

- JRMP, che richiede l'uso di RMIC per generare stub e skeleton;
- JERI, che genera un proxy dinamico come riferimento remote, risparmiando l'uso di RMIC;
- IIOP, che rende il service disponibile via RMI-IIOP;
- Activation, che avvolge (wrap) un altro exporter, per rendere l'oggetto remoto disponibile via activation.

La maggior parte di questi exporter accettano istanze Endpoint come parametri per i loro costruttori, che forniscono i mezzi per configurare differenti protocolli di trasmissione, tra cui:

- TCP
- HTTP
- HTTPS

Listing A.3: Uso Exporter (tratto dal Service Eventi)

```
//preparo l'oggetto da rendere disponibile ai servizi client
protected void initio() throws Exception {
      Exporter exporter = getExporter();
      InterfaceE serverProxy = (InterfaceE) exporter.export(this);
      ...
      }

//creo oggetto Exporter
protected Exporter getExporter() throws ConfigurationException, RemoteException {
      return (Exporter) config.getEntry("com.sun.jini.asService", "exporter",
          Exporter.class, new BasicJeriExporter(TcpServerEndpoint.getInstance(0),
          new BasicILFactory()));
      }
```

- SSL
- Kerberos

Data la flessibilità fornita da questo strumento, molti servizi specificano i propri exporter direttamente nel file di configurazione, in modo tale da poter cambiare in maniera molto semplice l'implementazione scelta (ovviamente se il servizio in questione supporta più tipi di implementazione).

Nel codice riportato in A.3, tratto dalla classe ServiceE del service Eventi, possiamo vedere il metodo *getExporter()* che acquisisce le impostazioni riferite all'Exporter, contenute nel file di configurazione iniziale. Se la voce corrispondente non viene trovata, viene utilizzato l'ultimo parametro del metodo *getEntry(...)* (in questo caso verrà utilizzato l'implementazione BasicJeriExporter).

ServiceID

Per ogni service pubblicato è necessario creare una chiave identificativa univoca (da 128 bit), per riuscire a distinguerlo tra tutti i vari service della rete (che possono implementare la stessa interfaccia). Per fare ciò Jini mette a disposizione la factory UuidFactory (per creare la chiave) e la classe ServiceID (per contenerla). Tale chiave verrà passata al LUS nella fase di Join.

Discovery e Join

La fase di Discovery è già stata presentata nel capitolo 3. Di seguito andiamo a vedere come si concretizza in fase di programmazione. L'operazione di ricerca del Lookup Service è comandata da un try-catch (A.5). Si cercherà infatti di trovare

Listing A.4: Creazione ID del service (tratto dal Service Eventi)

```
//creo un ID univoco per i LUS da assegnare a questo servizio
protected static ServiceID getServiceID() {
      Uuid uuid = UuidFactory.generate();
      return new ServiceID(uuid.getMostSignificantBits(), uuid.
          getLeastSignificantBits());
                }
```

Listing A.5: Fase di Discovery, lato server (tratto dal Service Eventi)

```
      ...
      DiscoveryManagement discoveryManager;
      try {
          discoveryManager = (DiscoveryManagement) config.getEntry("com.sun.jini.
              asService", "discoveryManager", DiscoveryManagement.class);
      }
      catch (NoSuchEntryException e) {
          //stringa vuota = gruppo di default
          discoveryManager = new LookupDiscovery(new String[] {""}, config);
      }
      ...
```

una Entry nel file di configurazione corrispondente alla fase di Discovery (che conterrà il nome del gruppo in cui cercare il LUS). Se la ricerca non avrà esito positivo (catch), allora si cercherà nel gruppo di default (stringa vuota), a cui appartengono tutti i Jini service, LUS compresi.

La fase di Join è molto semplice : il service si "pubblica" nei LUS definiti nell'oggetto *discoveryManager*, mette a disposizione i suoi metodi ai client, accessibili tramite l'oggetto Exporter (qui chiamato *serverProxy*) e si assegna un identificativo univoco per poter essere rintracciato agevolemente (tramite getServiceID()).

Listing A.6: Fase di Join (tratto dal Service Eventi)

```
     ...
     JoinManager joinManager = new JoinManager(serverProxy, null, getServiceID(),
         discoveryManager, null, config);
     ...
```

Listing A.7: Fase di Discovery, lato client (tratto dal Service Eventi)

```
...
ServiceDiscoveryManager serviceDiscovery;
try {
    serviceDiscovery = (ServiceDiscoveryManager) config.getEntry( "com.sun.
        jini.asClient" , "serviceDiscovery", ServiceDiscoveryManager.class);
    }
catch (NoSuchEntryException e) {
    //stringa vuota = gruppo di default
    serviceDiscovery = new ServiceDiscoveryManager(new LookupDiscovery(new
        String[] {""}, config),null, config);
    }
...
```

Listing A.8: Fase di Lookup (tratto dal Service Eventi)

```
                                ...
  Class[] classe = {InterfaceScreen.class};
  ServiceItem serviceItem = serviceDiscovery.lookup(new ServiceTemplate(null,
       classe, null), null, Long.MAX_VALUE);
  schermo = (InterfaceScreen) serviceItem.service;
                                ...
```

Discovery e Lookup

La fase di Discovery dal lato client poco si discosta dalla stessa fase fatta nell'ottica server. Le operazioni sono le stesse, unica differenza la classe. Mentre nel caso precedente veniva utilizzata la classe DiscoveryManagement, qui viene usata ServiceDiscovery.

Come dicevamo nel capitolo 3, nella fase di Lookup il servizio client deve conoscere l'interfaccia del servizio server che sta cercando. E' per questo che si crea un oggetto Class, che verrà passato come parametro nella ricerca (metodo *lookup(...)*). Oltre a questo parametro, da notare il valore *Long.MAX_VALUE*, per impostare il tempo di Leasing. Il tempo di Leasing è il periodo di tempo oltre il quale il riferimento al servizio utilizzato deve essere rinnovato. La ricerca può trovare uno o più oggetti (della classe ServiceItem) corrispondenti a uno o più servizi disponibili. In questo caso (A.8) viene preso il primo della lista (dato che non c'è la necessita di averne uno in particolare). L'item viene utilizzato per creare l'oggetto che rappresenta il collegamento al server, e da cui si invocano i metodi remoti.

Ringraziamenti

Dicono che le uniche due cose sicure al mondo siano solo la morte e le tasse. Nel mio caso, dato che devo ancora fare esperienza delle due (fortunatamente!), l'unica cosa certa della mia vita è stata, ed è la mia famiglia. Essa è stata sempre un "porto tranquillo" in cui rifugiarmi, in cui trovare sollievo e in cui trovare ispirazione. Oltre a questo devo ringraziare mamma, papà e mio fratello perché mi hanno fornito i più bei esempi di vita che io potessi desiderare, e devo a loro il fatto di essere come sono.

Voglio ringraziare il mio relatore Prof. Carlo Ferrari per la disponibilità e il supporto durante lo svolgimento di questo lavoro.

Ringrazio Francesco, Andrea e Edoardo (i Daunlò!) perché volontariamente o involontariamente mi hanno sempre supportato (e sopportato) in ogni momento della mia vita (personale, scolastica, sportiva, affettiva e associativa).
Ringrazio Barbara per essere stata la parente più meravigliosamente rompi** della storia :D.
Assieme a loro è doveroso citare il mio stupendo gruppo di amici : Michele, Enrico, Marzia, Daniele, Mattia, Federica, Alessandra, Matteo, Andrea ZZ, Alberto, Diego, Giacomo, Valentina, Tommaso, Monica.

Ringrazio tutti i parenti a cominciare dai nonni, passando per i miei zii e arrivando ai miei cugini, in particolare Anna. Un ricordo speciale va a Nonna Antonia.

E arriviamo agli amici più recenti, ma non per questo meno importanti (anzi!) Ciccio (Forza Palermo!), Filippo (detto Leonard), Peppe (il database sportivo umano), Marco (detto Leonard), Andrea (detto Zoncavep), Fabio (il tirapacchi), Marco L, IL Costa, Lovo (detto Clichè), Mauro.

Ringrazio l'Associazione Il Barco per avermi dato tanto da fare in questi anni (ma di contro tanta soddisfazione), nelle persone di Massimo, Mattia, Luca, Gilberto, Stefano, Marianna, Alessia, Jessica, Carlotta, Monica, Clarissa, Melissa. Doveroso citare la squadra di volley, a cominciare dal mitico coach Roberto, per arrivare a Ermanno e Alice, passando per Giovanni e Matteo, e gli ASI Boys, Alberto, Renato e Marco.

Bibliografia

[1] ACM International Conference on Ubiquitous Computing, www.ubicomp.org

[2] International ICST Conference on Mobile and Ubiquitous Systems, www.mobiquitous.org

[3] International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, www.iaria.org/conferences.html

[4] International Conference on Ubiquitous Intelligence and Computing, www.itee.uq.edu.au/~uic09

[5] International Conference on Autonomic and Trusted Computing, www.itee.uq.edu.au/~atc09

[6] International Conference on Pervasive Computing, www.pervasive2008.org

[7] IEEE Pervasive Computing, www.computer.org/pervasive

[8] S. Duval, C. Hoareau, H. Hashizume, *Age in Ubiquitous Computing : A Thin Thread*, Third 2008 International Conference on Convergence and Hybrid Information Technology, Busan, Korea, Novembre 11-13, 2008

[9] M. Alwan, J. Nobel, *State of Technology in Aging Services According to Field Experts and Thought Leaders*, Center for Aging Services Technologies (CAST), Washington, USA, Febbraio 2008

[10] M. Ghorbel, F. Arab, M. Mokhtari, *Assistive Housing : Case Study in a Residence for Elderly People*, 2nd International Conference on Pervasive Computing Technologies for Healthcare 2008, Tampere, Finlandia, Gennaio 30 - Febbraio 01 2008

[11] M. E. Pollack, L. Brown, D. Colbry, C. E. McCarthy, C. Orosz, B. Peintner, S. Ramakrishnan, I. Tsamardinos, *Autominder : an Intelligent Cognitive Orthotic System for People with Memory Impairment*, USA, 2003

[12] V. Osmani, D. Zhang, S. Balasubramaniam, *Human Activity Recognition Supporting Context-Appropriate Reminders for Elderly*, Pervasive Health 2009, London, Gran Bretagna, Aprile 1-3 2009

[13] T. Nakada, H. Kanai, S. Kunifuji, *A Support System for Finding Lost Object using Spotlight*, Mobile HCI 2005, Salzburg, Austria, Settembre 19-22 2005

[14] M. Aiello, S. Dustdar, *Are our homes ready for services? A domotic infrastructure based on the Web service stack*, Pervasive and Mobile Computing 4, 2008, pg 506-525

[15] COGKNOW, Helping people with mild dementia to navigate their day, www.cogknow.eu

[16] W. K. Edwards, *Core Jini* 2nd Edition, Prentice Hall, USA, 2001