ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria e Architettura, campus di Cesena
Corso di Laurea Magistrale in Ingegneria Informatica

TECNOLOGIE PER LA MOBILITÀ VERSO UN MIDDLEWARE GENERAL-PURPOSE

Elaborata nel corso di: Sistemi Multi-Agente LM

Laureando
Stefano Montesi

Relatore
Andrea Omicini

Correlatore
Stefano Mariani

ANNO ACCADEMICO 2013–2014
SESSIONE I

Ringraziamenti

Con questo atto si conclude un percorso non semplice che ha segnato un periodo importante della mia vita, permettendomi di crescere anche professionalmente e di imparare a gestire i miei impegni accademici. Si prospetta ora un cambiamento radicale che metterà alla prova quelle qualità che mi hanno consentito di arrivare fin qui: desidero quindi ringraziare tutti quelli che hanno permesso questa valorizzazione e che spero mi abbiano preparato al meglio per affrontare le sfide che il mondo del lavoro mi riserverà.

Un tanto doveroso quanto sentito ringraziamento va al professor Omicini, che ha rappresentato durante questi 5 anni un punto di riferimento per la mia attività accademica e che ha accettato anche in questi ultimi due anni di seguirmi durante la tesi con la massima disponibilità nonostante alcuni problemi che hanno rallentato lo sviluppo di quest'ultima. Al suo fianco, e soprattutto al mio, ha svolto un egregio lavoro di supporto l'Ing. Stefano Mariani, che con la sua competenza e voglia di fare, ha reso il mio compito molto più semplice fornendomi diversi spunti di riflessione e seguendo passo passo il mio lavoro.

Per motivi diversi, non ho avuto con me in questi due anni i miei compagni della triennale a cui sono particolarmente legato, ma sono comunque grato a loro per il supporto che mi danno ogni giorno e per i bei momenti passati insieme. Al di fuori dell'ambiente universitario, la mia vita si basa principalmente su una grande famiglia di amici, quelli del Quindi, che mi hanno permesso di affrontare questo percorso serenamente non facendomi mai mancare nulla di ciò di cui avevo bisogno: anche a loro devo tanto della mia tranquillità e spensieratezza. Ci tengo a ringraziare in modo speciale una persona che è entrata nella mia vita in un momento un po' particolare, dandomi la forza di andare avanti e offrendomi una spalla enorme su cui appoggiarmi: a lei mi unisce inoltre una passione che mi ha permesso di vivere dei

momenti splendidi lasciando alle spalle diversi pensieri.

Dulcis in fundo, ultimi ma non certo per importanza, desidero ringraziare la mia famiglia che ha continuato ad appoggiarmi anche nei momenti in cui le cose non andavano esattamente come previsto, dimostrandomi quanto tiene a me e quanto il loro sentimento non dipenda certo da voti universitari. Anche se per colpa del mio carattere probabilmente non gliel'ho mai dimostrato direttamente, l'impegno profuso in ogni attività che svolgo è anche volto a valorizzare la fiducia che loro ripongono in me con la speranza di renderli ogni giorno sempre più orgogliosi.

Introduzione

La crescita esponenziale nella diffusione di smartphone e dispositivi con capacità computazionali che vanno dal semplice sensore a veri e propri processori complessi, ha reso l'ambiente un inconsapevole serbatoio di raccoglitori di informazioni disposti a distanze spesso molto ravvicinate. L'evoluzione tecnologica ha inoltre portato gli utenti verso esigenze sempre più complesse ed in questo senso si sta volgendo lo sguardo verso sistemi che siano in grado non solamente di reagire ai nostri comandi diretti, ma che sappiano prevedere quelle che possono essere le nostre necessità prima che esse vengano espresse. Per fare questo sono ovviamente richieste numerose informazioni, le quali devono essere pre-processate e attraverso una sorta di intelligenza artificiale devono produrre un risultato funzionale all'utente: stiamo quindi parlando di context-aware pervasive computing.

Gran parte delle tecnologie recentemente introdotte e diffusesi negli ultimi decenni hanno permesso di contestualizzare informaticamente quelli che sono i nostri comportamenti: in questo modo tutto ciò che facciamo, e quindi anche i nostri movimenti, può essere interpretato da sistemi complessi in grado di sfruttare tali informazioni a nostro vantaggio. Questi sistemi dinamici devono adattare il loro comportamento alle situazioni in cui incorrono, situazioni che semplicisticamente possono essere identificate tramite tre elementi: dove sei, con chi sei, quali risorse hai vicino.

La location awareness è spesso considerata un elemento complementare alla context awareness ma come abbiamo appena visto ne può anche rappresentare un elemento di composizione fondamentale: essa è senza dubbio una delle caratteristiche più importanti e più immediate se si pensa che le nostre abitudini dipendono soprattutto dal dove ci troviamo. Sono proprio la nostra posizione e i nostri movimenti gli oggetti di interesse di questo trattato e, considerando che gran parte di noi porta sempre con sè almeno uno smartphone, tali informazioni sono sempre più facilmente

reperibili.

La modellazione e il controllo di sistemi composti da molti dispositivi computazionali è un problema considerevole che sta crescendo sempre più con l'aumento vertiginoso del numero e della densità di tali device. Recentemente lo spatial computing è emerso come un approccio promettente nella modellazione e nel controllo di questi sistemi aggregati: l'idea che sta alla base di tutto si fonda sul fatto che quando la densità è molto alta, c'è un forte legame fra la struttura della rete di dispositivi e la geometria dello spazio sul quale essi sono distribuiti. I modelli di computazione tradizionali hanno spesso astratto dalla posizione fisica degli elementi (vedi internet) affidandosi maggiormente nelle implementazioni a fattori temporali come la sequenzialità; questa tendenza si sta clamorosamente invertendo sia per le necessità di parallelismo sia per l'importanza crescente della posizione con la diffusione di sistemi distribuiti dovuti alla crescita delle dimensioni dei sistemi stessi.

L'obiettivo della tesi è quindi inizialmente quello di esplorare i più avanzati dispositivi, sensori e processori per la computazione spaziale, quindi di correlarli con i modelli di spatial computing e infine di derivarne un'architettura concettuale di middleware distribuito che possa supportare le più avanzate applicazioni in mobilità.

Indice

Capitolo 1

STRUMENTI A DISPOSIZIONE PER LA COMPUTAZIONE SPAZIALE

Cominciamo il nostro percorso occupandoci di tutto ciò che è presente allo stato dell'arte per quanto riguarda gli strumenti inerenti allo spatial computing che sono a disposizione dei costruttori e degli sviluppatori, in modo da coprire non solo le componenti hardware ma soprattutto le possibilità software proposte dai maggiori sistemi operativi mobile in termini di interfacce offerte dal dispositivo. Analizzeremo quindi i più diffusi sistemi di localizzazione e solo in seguito affronteremo nel dettaglio i rispettivi SDK (Software Development Kit) di IOS e Android.

1.1 Metodi di localizzazione

Per prima cosa cerchiamo di capire quali sono le principali tecniche adottate per la localizzazione dei dispositvi mobili, una prima suddivisione può essere effettuata fra:

- Handset Based, sono i metodi di calcolo della posizione basati sull'hardware e firmware del dispositivo (algoritmi di calcolo specifici, con o senza assistenza della rete);

- Network Based, sono i metodi di calcolo della posizioni basati su software e hardware installato a livello di rete radiomobile (con eventuale contributo del

terminale).

Oltre alle considerazioni sul consumo energetico, nell'effettuare una scelta di questo tipo, si deve tener conto sì della qualità del servizio (QoS) che si vuole offrire (precisione della misura, tempo necessario ad ottenerla), ma anche del carico di messaggi necessari, della complessità dei dispositivi, del costo degli apparati di rete, della complessità architetturale e non ultima dell'affidabilità della soluzione scelta.

1.1.1 GPS based

Il GPS, o Global Positioning System (sistema globale di posizionamento), è attualmente basato sul funzionamento di 31 satelliti orbitanti intorno alla terra a circa 18-20 km di altezza che inviano costantemente segnali su tutta la superficie terrestre, questi segnali vengono ricevuti da semplici antenne che li utilizzano per triangolare la propria posizione: si utilizza questo termine in quanto tecnicamente, dopo una prima fase di allineamento generale, sarebbero sufficienti 3 satelliti che inviano segnali in modo triangolare, più un satellite supplementare che invia informazioni sull'altitudine. Un dispositivo elettronico di tipo ricevitore GPS, fino a quel momento privo di qualsiasi informazione di posizionamento, inizia a ricevere segnali dai satelliti nello spazio: appena almeno 8 di questi satelliti riescono a restituire al dispositivo informazioni relative alla posizione specifica, esso vi rimane costantemente in contatto e da quel momento sono sufficienti solo quelli della triangolazione, ovvero 3 satelliti che mantengano il segnale costante. Al termine della fase di allineamento, la posizione ottenuta viene confrontata in tempo reale con software di cartografia, in modo da stabilire con estrema precisione (attualmente fra i 5 e i 10 metri) la propria posizione su una mappa specifica. I fattori che incidono sulla rapidità dell'allineamento a 8/9 satelliti in simultanea, sono numerosi:

- copertura dovuta ai palazzi circostanti;
- orario del giorno e relativo posizionamento dei satelliti (migliore al mattino);
- schermatura dei parabrezza o di tutto ciò che impedisce il normale flusso dei segnali;
- posizione geografica in cui ci si trova (peggiore segnale in città, migliore in isole o lungo coste);

- saturazione ionica (cielo coperto, nuvoloso, oppure cielo azzurro e pulito);
- velocità di movimento (posizione fissa o dinamica).

Per tutti questi motivi, soprattutto nei dispositivi mobile come gli smartphone, il GPS si è evoluto in A-GPS ovvero Assisted GPS: i dispositivi GPS stand-alone commerciali infatti, sono generalmente dotati di antenne di discreta ampiezza di ricezione e di speciali algoritmi per l'allineamento rapido ai satelliti, gli smartphone moderni utilizzano invece chipset personalizzati, con il sistema GPS spesso integrato insieme ad altri sistemi. Questo è spesso causa di una ricezione non ottimale rendendo così necessario uno stratagemma che migliori la situazione: c'è la possibilità di appoggiarsi alla rete dati e a quella telefonica per ottenere informazioni supplementari sul posizionamento, allo scopo di raggiungere un allineamento più rapido e mantenere un segnale più intenso durante il tragitto.

1.1.2 SIM based

Partendo dal presupposto che il dispositivo non disponga di hardware o firmware con caratteristiche specifiche per la localizzazione, nel caso in cui si tratti di un dispositivo telefonico in possesso di una SIM card di un qualsiasi operatore, possiamo comunque determinarne la posizione. Ciascun operatore conosce infatti la posizione dei suoi ripetitori e quindi delle celle ad essi associate, recuperando l'ID della cella da cui l'utente è servito si può quindi risalire alla posizione di quel determinato utente. Tra i lati negativi di questo metodo troviamo ovviamente l'impossibilità di utilizzo in caso di mancanza di SIM nel dispositivo in nostro possesso, la mancanza di celle del nostro operatore telefonico nella zona e la forte dipendenza della QoS dalla densità di celle presenti sul territorio in questione. In questo senso si rende molto utile il servizio offerto da opensignal.com, sito che ci consente di visualizzare una mappa mondiale della copertura di rete evidenziando le posizioni delle torri degli operatori.

Lo stesso team è inoltre sviluppatore di un'app disponibile sia per IOS che per Android in grado di identificare la torre a cui si è connessi visualizzandola sulla mappa, oltre che di analizzare l'intensità del segnale e misurare la velocità di connessione.

Fra i metodi SIM based dobbiamo inoltre considerare il cosiddetto "Timing Advance": esso è un parametro di controllo della rete GSM e rappresenta la misura

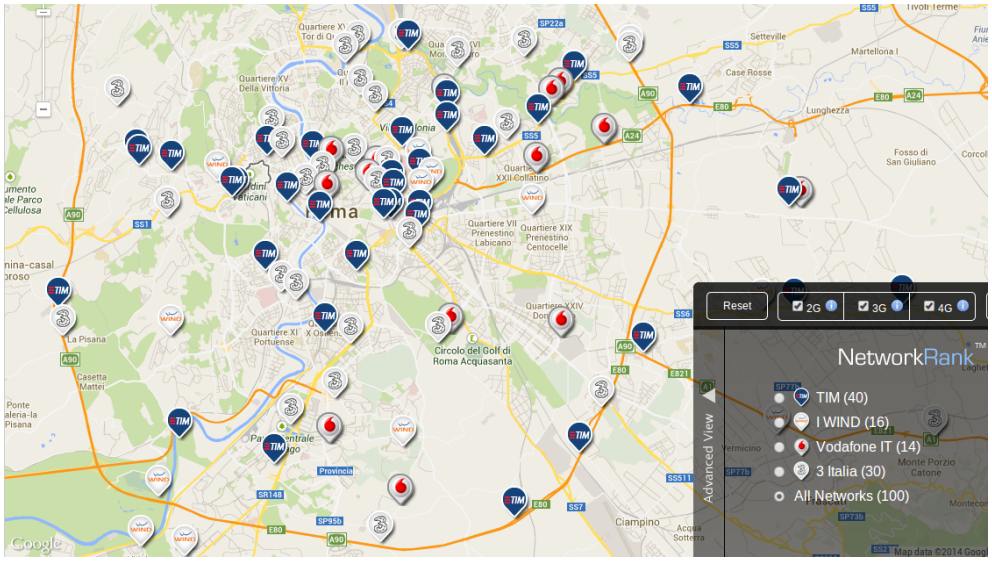

Figura 1.1: disposizione delle torri dei vari operatori nella zona di Roma

temporale di percorrenza fra la stazione radio base (Base Transceiver Station) e il terminale, in caso di rete UMTS si parla di Round Trip Time. Il valore di TA è di solito tra 0 e 63 e, considerando che le onde radio viaggiano a circa 300 metri per microsecondo, un passo TA rappresenta quindi un cambiamento nella distanza di andata e ritorno di circa 1100 metri: ciò significa che il valore TA cambia ogni 550 metri. Questo limite di 63 x 550 metri rappresenta la distanza massima di 35 km che un dispositivo può avere rispetto alla stazione base ed è il limite per quanto riguarda la distanza tra celle durante il loro collocamento sul territorio. Oltre all'ID della cella sono infatti proprio la potenza del segnale e il cosiddetto round trip time a consentirci una stima più precisa della posizione riducendo il raggio di localizzazione intorno alla torre.

1.1.3 Wi-Fi based

La grande diffusione del Wi-Fi che si è avuta nell'ultimo decennio ha portato anche al potenziamento delle possibilità di localizzazione: ogni access point può infatti essere associato a delle informazioni di posizione e le tecniche utilizzate nei meccanismi SIM based possono essere quindi estese anche in questo campo. A tal proposito si rende indispensabile il lavoro che aziende come Google, Microsoft e Apple realizzano quasi a nostra insaputa: esse infatti utilizzano periodicamente i nostri dispositivi

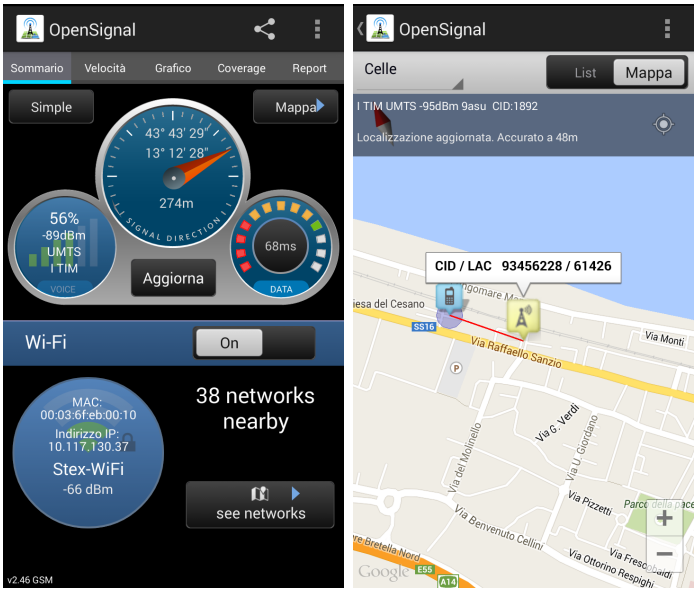

Figura 1.2: screenshot dell'applicazione openSignal su dispositivo Android

Figura 1.3: zone individuabili tramite cell ID e timing advance

per verificare la posizione tramite GPS, cell ID e Wi-Fi, ricevendo in cambio il MAC e l'SSID dell'access point a cui siamo connessi. In questo modo possono mantenere una sorta di enorme database in cui ogni access point viene associato ad una posizione ben precisa e quindi a importanti informazioni da poter riutilizzare quando a necessitare della posizione sarà un altro dispositivo che riceve il segnale della rete wi-fi in questione anche senza avere le credenziali necessarie a connettersi.

Lo scarso rendimento dei metodi GPS-based in ambienti chiusi e la crescente popolarità del Wi-Fi hanno inoltre incoraggiato le imprese a progettare nuovi metodi che sfruttano il wi-fi per il posizionamento interno.

```
IEEE 802.11abg  ESSID:"Stex-WiFi"
Mode:Managed  Frequency:2.422 GHz  Access Point: 00:03:6F:EB:00:10
Bit Rate=54 Mb/s   Tx-Power=15 dBm
Retry  long limit:7   RTS thr:off   Fragment thr:off
Power Management:off
Link Quality=68/70  Signal level=-42 dBm
Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
Tx excessive retries:1  Invalid misc:4645   Missed beacon:0
```

Figura 1.4: risultato del comando iwconfig su piattaforma Linux

Indoor Positioning

Come alcuni dei sistemi di posizionamento "esterni", quelli interni si basano su una rete di dispositivi definiti àncore che rappresentano dei nodi con una posizione nota: al momento non è stato definito uno standard in materia di IPS ed essi non rientrano negli standard dei real time locating systems, questo fa sì che si trovino sul mercato sistemi di questo tipo fra loro anche molto differenti. Questi dispositivi localizzano attivamente dei tag o offrono un contesto ambientale da poter percepire sfruttando tecnologie ottiche, radio e persino acustiche, ma in ogni caso si tratta comunque di tecnologie wireless.

- Radio frequency identification (RFID): è una tecnologia per l'identificazione e/o memorizzazione di dati che si basa sulle capacità di particolari dispositivi elettronici. Essi sono in grado di memorizzare dati e rispondere ad interrogazioni a distanza da parte di appositi apparati rappresentati da lettori a radiofrequenza: tra i punti favorevoli va annoverato il fatto che i tag passivi sono molto economici, mentre va considerato il limite del mancato supporto alle metriche.

- Ultrawide band (UWB): è una tecnica di trasmissione sviluppata per trasmettere e ricevere segnali mediante l'utilizzo di impulsi di energia a radiofrequenza di durata temporale estremamente ridotta e quindi con occupazione spettrale molto ampia. Il vantaggio di tale tecnica sta nel fatto che la brevità dell'impulso rende l'UWB poco sensibile alle interferenze dovute alla riflessione dell'onda stessa. La larghezza della banda fa sì che la densità spettrale di potenza sia quindi molto bassa; questa caratteristica permette alla comunicazione di non interferire con le applicazioni già esistenti e consente la realizzazione di dispositivi con un consumo energetico ridotto.

- Infrarossi (IR): si utilizza la radiazione elettromagnetica con banda di frequenza dello spettro elettromagnetico inferiore a quella della luce visibile, ma maggiore di quella delle onde radio.
- Visible light communication (VLC): anche in questo caso si trasmettono e ricevono segnali ma si tratta di luce visibile emessa da lampade fluorescenti.
- Ultrasuoni: le onde scambiate sono di natura acustica e si muovono molto lentamente favorendo un alto grado di precisione. In seguito vedremo nel dettaglio come può essere utilizzato a questo scopo anche il Bluetooth.

1.2 APPLE (IOS)

Come ben noto, la Apple Inc. è un'azienda informatica statunitense che produce computer e dispositivi multimediali, ma anche i sistemi operativi che permettono il funzionamento di questi device. In particolare, essendo interessati a nozioni riguardanti la mobilità, ci concentreremo sul sistema operativo dedicato ai dispositivi mobili ovvero IOS.

1.2.1 Motion CoProcessor (M7)

Con l'uscita a metà settembre 2013 dell'ultimo device di fascia alta, l'azienda di Cupertino a tentato di modificare la filosofia che stava dietro alla maggior parte delle computazioni spaziali presenti nel dispositivo introducendo un nuovo componente da abbinare ai classici processori. Apple M7 è un microcontrollore general purpose (NXP LPC18A1) basato su un ARM Cortex M3 che sfrutta un software custom sviluppato da Apple per controllare i sensori montati sulla scheda madre, esso viene affiancato ad un processore standard per le elaborazioni classiche e pertanto viene definito motion coprocessor. Introdotto per la prima volta nel nuovo iPhone 5S, la sua funzione è quella di raccogliere i dati provenienti dai sensori integrati (accelerometro, giroscopio e bussola) sollevando da questo incarico l'unità centrale di elaborazione ovvero il processore principale; in questo modo è possibile raccogliere, elaborare e memorizzare tali dati anche se il device è in standby mentre le applicazioni potranno accedervi non appena esso verrò riattivato. Tale scelta è stata ovviamente effettuata allo scopo di ridurre il consumo di batteria, demandando a un'unità di elaborazione

a risparmio energetico operazioni continue di processamento, si suppone infatti che M7 sia in grado di abilitare numerose operazioni di monitoraggio in background a livelli di energia veramente ridotti. Tale soluzione può essere sfruttata soprattutto per quanto riguarda le app che monitorano l'attività fisica del proprietario del dispositivo, in modo da percepire quale tipo di movimento l'utente stia effettuando (guidare, camminare, correre, dormire), oppure nell'ambito dell'indoor tracking.
Nello specifico i sensori a disposizione nell'iPhone 5s sono:

- una bussola elettronica a 3 assi AK8963 la quale combina un sensore magnetico per rilevare X, Y, e Z, un circuito di pilotaggio del sensore, un circuito aritmetico e una catena di amplificazione del segnale;
- un giroscopo a 3 assi STMicroelectronics;
- un accelerometro a 3 assi Bosch Sensortech BMA220.

Combinando tutte le informazioni a disposizione, il chip è in grado di conoscere tutto ciò che riguarda il movimento (6 gradi di libertà), ottenendo anche velocità, accelerazione e spostamento: in questo modo l'utilizzo del GPS è richiesto solo occasionalmente per correggere la posizione dopo diverso tempo.

Possibili utilizzi

Un dispositivo in grado di comprendere lo stato attuale oltre alla posizione è un passo fondamentale verso device ancor più personali: nell'utilizzo delle mappe ad esempio, il contributo di M7 può rivelarsi utile cambiando indicazioni quando si passa dalla guida alla camminata, oppure nell'inibire le richieste di connessione a reti wi-fi durante un viaggio in automobile, in alternativa si può sfruttare l'inattività del dispositivo per ridurre la frequenza di aggiornamenti dalla rete. Il momento del passaggio da guida a camminata può inoltre essere utilizzato per registrare il punto in cui si parcheggia l'auto e quindi sfruttato in seguito quando si ha la necessità di ritrovare il proprio mezzo. Sembra inoltre che Apple abbia intenzione di utilizzare M7 per migliorare le mappe includendo caratteristiche relative a indoor mapping e trasporti pubblici, cosa che si rivelerebbe alquanto sensata viste le acquisizioni di compagnie come WiFiSLAM e Hopstop. Il chip M7 è reso accessibile attraverso l'utilizzo di

Core Location e Core Motion API che analizzeremo in seguito: esse sono presenti a partire da iOS 7.

1.2.2 IBeacon

Al WWDC (Worldwide Developers Conference) di giugno 2013 Apple aveva silenziosamente annunciato iBeacon come una delle caratteristiche più importanti di iOS 7 senza però fornire nessun dettaglio a riguardo, la presentazione a settembre dell'ultimo sistema operativo made in Cupertino è stata dunque la prima occasione per parlare concretamente di questa tecnologia. IBeacon rappresenta un profilo per le comunicazioni a bassa energia dedicato a trasmettitori di informazioni contestuali: in pratica esso è semplicemente un profilo proprietario che incapsula lo standard Bluetooth Low Energy già esistente (anche detto Smart o Bluetooth 4.0), il quale veniva già supportato fin dall'iPhone 4S soprattutto per consentire la comunicazione con accessori legati al fitness. Se paragonato all'NFC, i vantaggi consistono principalmente nel raggio d'azione (stimato indicativamente in 10 metri) e nella facilità di accoppiamento con i device, allo stesso tempo però vanno considerati come svantaggi i costi decisamente superiori e la necessità di energia: i tag NFC richiedono infatti una distanza massima per l'utilizzo di circa 10 cm (equiparabile al contatto) ma allo stesso tempo non devono essere alimentati. La maggiore raggiungibilità di iBeacon però rischia di trasformarsi in un punto debole quando si parla di un ambito cruciale come quello dei pagamenti: i dati che transitano via NFC non sono infatti intercettabili (e quindi violabili) da nessuno che sia al di fuori di quei pochi centimetri nei quali avviene la transazione. Questa novità è stata presentata da Apple come l'apertura di una porta verso il cosiddetto "internet of things" e verso l'indoor mapping, consentendo così la creazione di app basate sulla micro-localizzazione personalizzata: il segnale GPS viene infatti schermato da oggetti solidi come montagne o edifici e sono proprio queste le situazioni che iBeacon intende risolvere. Collegandosi col proprio device all'iBeacon più vicino potremo ottenere infatti la posizione GPS codificata in fase di installazione o utilizzare la potenza del segnale per captare il tipo di movimento che stiamo effettuando.

1.2.3 Core Location API

Per prima cosa cerchiamo di capire quale possibilità IOS ci offre in termini di rilevazione della posizione corrente e della direzione intrapresa: ovviamente il framework in questione utilizza tutto l'hardware a disposizione per configurare e programmare la generazione di eventi specifici. Possiamo quindi identificare 3 servizi principali:

- il servizio per i cambi di posizione rilevanti offre un modo di ottenere la posizione che non prevede un consumo energetico significante e consente di essere notificati in caso di cambiamenti;
- il servizio di localizzazione standard offre un metodo molto più configurabile in cui si possono scegliere diverse opzioni a seconda delle necessità specifiche;
- il servizio per il controllo delle regioni monitora i movimenti dell'utente in entrata e in uscita rispetto a delle aree geografiche precedentemente definite.

CLBeacon, CL BeaconRegion, CLRegion, CLCircularRegion
Queste 3 classi identificano e servono a definire gli elementi su cui poi il servizio di controllo delle regioni dovrà andare a lavorare valutando la posizione e i movimenti del dispositivo. Nel primo caso si parla concretamente dell'oggetto beacon che può essere incontrato scandendo una regione, esso non deve essere instanziato dal programmatore ma viene restituito dal location manager. Per le altre tre classi si tratta invece di due tipologie di regioni che il programmatore deve definire a partire da un beacon o da una posizione e da un raggio.

CLGeocoder, CLPlaceMark
La classe CLGeocoder offre dei servizi di conversione fra coordinate geografiche specifiche e la loro rappresentazione user-friendly, stiamo quindi parlando di elementi che l'utente può identificare in maniera molto più diretta rispetto a complessi numeri non sfruttabili con immediatezza. Sono quindi presenti metodi di forward geocoding che a partire da indirizzi o punti di riferimento restituiscono coordinate di latitudine e longitudine, al contrario il reverse geocoding consentirà di ottenere informazioni su strade, città, stati e paesi partendo da una posizione. Questo genere di servizio ha ovviamente bisogno della rete per funzionare e in entrambi i casi restitutisce uno o più oggetti di tipo CLPlaceMark, classe che viene quindi utilizzata per definire i dati relativi ad una posizione geografica.

CLHeading

Ciascun oggetto appartenente a questa classe contiene dati relativi alla direzione generati da un CLLocationManager: essi consistono nei valori del nord geografico e del nord magnetico, oltre che in una serie di dati grezzi per il vettore tridimensionale utilizzato nel computare tali valori.

Definisce la precisione della rilevazione: valore negativo errore di rilevazione

```
@property(readonly, nonatomic) CLLocationDirection headingAccuracy
```

Definiscono la direzione rispetto al nord magnetico e geografico

```
@property(readonly, nonatomic) CLLocationDirection magneticHeading,
    trueHeading
```

Indicano le componenti sui 3 assi dei dati geomagnetici misurati in microtesla.

```
@property(readonly, nonatomic) CLHeadingComponentValue x, y, z
```

Indica l'istante in cui la misura è stata raccolta

```
@property(readonly, nonatomic) NSDate *timestamp
```

CLLocation

Ciascun oggetto appartenente a questa classe contiene dati relativi alla posizione generati da un CLLocationManager: essi consistono nei valori delle coordinate geografiche accompagnati da indici di precisione dei dati.

Indica l'altitudine misurata in metri a partire dal livello del mare

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationDistance
    altitude
```

Indica le coordinate di latitudine e longitudine

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationCoordinate2D
    coordinate
```

```
typedef struct {
       CLLocationDegrees latitude;
       CLLocationDegrees longitude;
} CLLocationCoordinate2D;
```

Indica la direzione di movimento misurata in gradi a partire dal nord e procedendo in senso orario

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationDirection
    course
```

Indica la velocità istantanea nella direzione corrente

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationSpeed speed
```

Indicano il raggio del cerchio il cui centro è identificato dalle coordinate latitudinali e longitudinali e la precisione relativa all'altitudine

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationAccuracy
    horizontalAccuracy, verticalAccuracy
```

Indica l'istante in cui la misura è stata raccolta

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) NSDate *timestamp
```

1.2.4 Core Motion API

Andiamo ora ad analizzare in dettaglio le Application Programming Interfaces che IOS 7 ci mette a disposizione per analizzare tutti i movimenti compiuti dal dispositivo: esse vengono racchiuse con la denominazione di "core motion API" e consentono di ricevere le informazioni provenienti dall'hardware presente sul device e di processarle secondo le nostre necessità.

CMLogItem

La classe CMLogItem è una classe base per le classi di Core Motion che gestiscono specifici tipi di eventi di movimento: gli oggetti di questa classe rappresentano dati

taggati a livello temporale che possono essere registrati in un file. CMLogItem definisce una proprietà di sola lettura timestamp che registra il tempo in cui una misura di movimento è stata presa.

Definisce quando la misura è stata raccolta in termini di secondi trascorsi dal boot del device

```
@property(readonly, nonatomic) NSTimeInterval timestamp
typedef double NSTimeInterval;
```

CMAccelerometerData

Un'istanza della classe CMAccelerometerData rappresenta un evento dell'accelerometro, ovvero la misura dell'accelerazione lungo i 3 assi spaziali in un particolare istante e include la proprietà timestamp della superclasse CMLogItem.

Rappresenta la misura dell'accelerazione lungo l'asse x, y, z in G

```
@property(readonly, nonatomic) CMAcceleration acceleration
typedef struct {
       double x;
       double y;
       double z;
} CMAcceleration;
```

CMAttitude

Un'istanza della classe CMAttitude rappresenta una misura dell'assetto del dispositivo in un particolare istante e si riferisce quindi all'orientamento di un corpo rispetto ad un dato sistema di riferimento. La classe CMAttitude offre tre diverse rappresentazioni matematiche dell'assetto: una matrice di rotazione, un quaternione, e gli angoli di Eulero (rollio, beccheggio e imbardata).

Rappresentazione matematica come angoli di Eulero

Come accennato in precedenza tale rappresentazione coinvolge 3 differenti proprietà:

Il rollio è una rotazione attorno ad un asse longitudinale che passa attraverso il dispositivo

```
@property(readonly, nonatomic) double roll
```

Il beccheggio è una rotazione attorno ad un asse laterale che passa attraverso il dispositivo

```
@property(readonly, nonatomic) double pitch
```

Un'imbardata è una rotazione attorno ad un asse che corre verticalmente attraverso il dispositivo

```
@property(readonly, nonatomic) double yaw
```

Rappresentazione matematica come matrice di rotazione

Una matrice di rotazione in algebra lineare descrive la rotazione di un corpo nello spazio euclideo a tre dimensioni.

Ciascun campo in questa struttura definisce un elemento della matrice di rotazione attraverso la sua posizione

```
@property(readonly, nonatomic) CMRotationMatrix rotationMatrix
typedef struct {
        double m11, m12, m13;
        double m21, m22, m23;
        double m31, m32, m33;
} CMRotationMatrix;
```

Rappresentazione matematica come quaternione

I quaternioni sono entità introdotte come estensioni dei numeri complessi

```
@property(readonly, nonatomic) CMQuaternion quaternion
typedef struct {
        double x, y, z, w;
} CMQuaternion
```

Un quaternione rappresenta la rotazione di theta radianti del vettore unitario x, y, z e le 4 componenti di un quaternione q sono definite come segue:
q.x = x * sin(theta / 2)
q.y = y * sin(theta / 2)
q.z = z * sin(theta / 2)
q.w = cos(theta / 2)

CMGyroData

Un'istanza della classe CMGyroData contiene una singola misura della velocità di rotazione del dispositivo e include la proprietà timestamp della superclasse CMLogItem.

Dati grezzi della velocità di rotazione del dispositivo attorno a tre assi estratti dal giroscopio

```
@property(readonly, nonatomic) CMRotationRate rotationRate
typedef struct {
       double x;
       double y;
       double z;
} CMRotationRate
```

CMMagnetometerData

Le istanze della classe CMMagnetometerData rappresentano misurazioni del campo magnetico realizzate dal magnetometro del dispositivo e includono la proprietà timestamp della superclasse CMLogItem.

Campo magnetico totale osservato dal dispositivo

```
@ property (readonly, nonatomic) CMMagneticField magneticField
typedef struct {
       double x;
       double y;
       double z;
} CMMagneticField;
```

CMDeviceMotion

Un'istanza di CMDeviceMotion incapsula misurazioni riguardanti l'assetto, la velocità di rotazione, e l'accelerazione di un dispositivo, inoltre include la proprietà timestamp della superclasse CMLogItem. Dal momento che il Core Motion è in grado di monitorare l'assetto di un dispositivo utilizzando sia il giroscopio che l'accelerometro, si può distinguere tra accelerazione utente e di gravità, la somma dei quali viene misurata dall'accelerometro. Un oggetto CMDeviceMotion fornisce le suddette misure nelle seguenti proprietà:

Il vettore di accelerazione di gravità espressa nel sistema di riferimento del dispositivo

```
@property(readonly, nonatomic) CMAcceleration gravity
```

L'accelerazione che l'utente sta impartendo al dispositivo

```
@property(readonly, nonatomic) CMAcceleration userAcceleration
```

Misura dell'assetto ovvero l'orientamento rispetto ad un dato sistema di riferimento

```
@property(readonly, nonatomic) CMAttitude *attitude
```

Valori della velocità di rotazione del dispositivo rilevata dal giroscopio e senza disturbi

```
@property(readonly, nonatomic) CMRotationRate rotationRate
typedef struct {
        double x;
        double y;
        double z;
} CMRotationRate
```

Campo magnetico totale nelle vicinanze del dispositivo senza i disturbi del dispositivo

```
@property(readonly, nonatomic) CMCalibratedMagneticField magneticField
typedef struct {
        CMMagneticField field;
        CMMagneticFieldCalibrationAccuracy accuracy;
} CMCalibratedMagneticField;
```

La struttura contiene i dati provenienti dal magnetometro a 3 assi

```
typedef struct {
        double x, y, z;
} CMMagneticField;
```

Indica la precisione della calibrazione della stima di un campo magnetico

```
typedef enum {
        CMMagneticFieldCalibrationAccuracyUncalibrated = -1,
        CMMagneticFieldCalibrationAccuracyLow,
        CMMagneticFieldCalibrationAccuracyMedium,
        CMMagneticFieldCalibrationAccuracyHigh
} CMMagneticFieldCalibrationAccuracy;
```

CMMotionActivityManager

La classe CMMotionActivityManager fornisce l'accesso ai dati di movimento memorizzati da un dispositivo. I dati di movimento riflettono se l'utente sta camminando, correndo, è in un veicolo, o è in sosta per un determinato periodo di tempo. Un app di navigazione potrebbe cercare cambiamenti nel tipo di movimento attuale e offrire soluzioni diverse per ciascuno di essi. Utilizzando questa classe, è possibile richiedere le notifiche quando cambia il tipo di movimento o si possono raccogliere gli ultimi dati relativi alle modifiche di movimento.

CMMotionActivity

La classe CMMotionActivity contiene i dati di un singolo evento di aggiornamento del movimento. Le proprietà di questa classe correlate al movimento non si escludono a vicenda: in altre parole, è possibile che più di una proprietà per volta contenga il valore SI ma allo stesso tempo è anche possibile che tutte le proprietà siano impostate su NO.

Un booleano che indica se il dispositivo è all'interno di un automobile

```
@property(readonly, nonatomic) BOOL automotive;
```

Un booleano che indica se il dispositivo è in possesso di una persona che sta correndo

```
@property(readonly, nonatomic) BOOL running;
```

Un booleano che indica se il dispositivo è fermo

```
@property(readonly, nonatomic) BOOL stationary;
```

Un booleano che indica se lo stato del dispositivo è sconosciuto

```
@property(readonly, nonatomic) BOOL unknown;
```

Un booleano che indica se il dispositivo è in possesso di una persona che sta camminando

```
@property(readonly, nonatomic) BOOL walking;
```

Indica il momento in cui si è verificata la variazione di movimento

```
@property(readonly, nonatomic) NSDate *startDate;
```

Indica la sicurezza nella valutazione del tipo di movimento.

```
@property(readonly, nonatomic) CMMotionActivityConfidence confidence;
typedef enum : NSInteger {
       CMMotionActivityConfidenceLow = 0,
       CMMotionActivityConfidenceMedium,
       CMMotionActivityConfidenceHigh
}CMMotionActivityConfidence;
```

CMMotionManager

Un oggetto CMMotionManager è il gateway per i servizi di movimento forniti da iOS. Questi servizi forniscono i dati accelerometrici, i dati di rotazione, i dati magnetometrici e altri dati di movimento, come l'assetto. Dopo la creazione di un'istanza di CMMotionManager, un app può utilizzarla per ricevere quattro tipi di movimento: dati grezzi dell'accelerometro, dati grezzi del giroscopio, dati grezzi del magnetometro e i device-motion trasformati (che comprendono accelerometro, rotazione, e le misurazioni di assetto) . I dati device-motion forniti sono stati processati da algoritmi di fusione dei sensori del core- motion in modo da ottenere un risultato in un certo senso raffinato. Un'applicazione può assumere due differenti approcci quando si ricevono i dati di movimento: gestendoli a specifici intervalli di aggiornamento

o campionando periodicamente i dati di movimento; con entrambi questi approcci, l'applicazione deve chiamare il metodo di arresto appropriato quando ha finito l'elaborazione di tutti i dati in oggetto.

CMStepCounter
La classe CMStepCounter fornisce accesso al numero di passi che l'utente ha effettuato con il proprio dispositivo. Informazioni riguardanti i passi sono raccolte su dispositivi che dispongono dell'appropriato hardware integrato e conservati in modo che sia possibile eseguire query per determinare la recente attività fisica dell'utente. È possibile utilizzare questa classe per raccogliere sia i dati di passo corrente sia eventuali dati storici.

1.3 GOOGLE (ANDROID) & co.

Il mercato in questione può essere definito senza grossi dubbi come un duopolio fra la già citata Apple Inc. e uno fra i più importanti colossi dell'informatica in generale: stiamo ovviamente parlando di Google Inc., azienda anch'essa statunitense nota principalmente per aver realizzato il miglior motore di ricerca al momento disponibile e per l'offerta di numerosi altri servizi online. A differenza di quanto detto in precedenza per Apple, Google non è produttrice di componenti hardware nè produce direttamente dispositivi (la gamma Nexus rappresenta una situazione abbastanza particolare), ma è la responsabile del progetto Android e quindi produce un sistema operativo utilizzabile su dispositivi mobili di produttori differenti.

1.3.1 Motion Processor per tutti i dispositivi

Sulla stessa scia intrapresa da Apple per l'M7, cominciano ad essere commercializzati degli "hub" o "fusion chip" che consentono di raccogliere e processare i dati provenienti da diversi sensori sollevando da questo compito il processore principale. Questi chip special purpose sono studiati appositamente per compiere questo genere di attività e quindi vengono ottimizzati soprattutto in termini di consumo: ad esempio il Sentral sensor fusion hub di PNI è in grado di gestire gli output di sensori differenti rendendo la percezione di un movimento su 9 assi super-precisa e pratica da implementare. Al momento tale circuito integrato è compatibile con tutti i dispositi-

vi che utilizzano Android e Windows 8 mobile, e dichiara di utilizzare meno del 10% dell'energia necessaria alle altre CPU per svolgere tale compito.

1.3.2 Bluetooth Low Energy

Il Bluetooth low energy (BLE) è una tecnologia di rete senza fili ideata per utilizzare le ultime applicazioni nei settori sanitari, del fitness, della sicurezza, e dell'intrattenimento domestico. Rispetto al classico Bluetooth, il BLE è destinato a fornire un consumo di energia notevolmente ridotto e a costo inferiore, pur mantenendo un range di comunicazione circa pari alla metà. Questo protocollo è stato fuso nello standard Bluetooth con la versione 4.0 ma non è retrocompatibile con le versioni precedenti: in definitiva le specifiche 4.0 consentono ai dispositivi di implementare la versione classica e quella LE sia in coppia sia singolarmente, tale implementazione distingue due classi di device:

- Bluetooth Smart Ready indica un dispositivo dual-mode, in genere un computer portatile o uno smartphone, il cui hardware è compatibile con entrambe le versioni;
- Bluetooth Smart indica un dispositivo solamente LE, tipicamente un sensore alimentato a batteria, che richiede uno Smart Ready per poter funzionare.

Il BLE utilizza la banda ISM a 2.4 GHz allo stesso modo di quello classico (questo permette ai dispositivi smart ready di utilizzare una singola antenna), ma diversamente usa un sistema di modulazione più semplice che consente di ottenere consumi minori e velocità di trasmissione maggiori. Il gruppo che si interessa di definire tali standard ha determinato diversi profili specifici a seconda del funzionamento richiesto in particolari contesti: i produttori dei dispositivi sono quindi tenuti ad implementare le specifiche appropriate per il funzionamento che vogliono garantire.

Supporto Android al BLE

Al Google IO di maggio 2013, evento dedicato agli sviluppatori, la casa di Mountain View ha annunciato il support al BLE a partire da Android 4.3 mostrando anche alcuni possibili utilizzi. Con tale versione dell'OS mobile, ogni dispositivo che monta un chip bluetooth dual mode diventa automaticamente Smart Ready e può così raccogliere dati dai vari accessori Smart. Per fare questo Google ha dovuto ricostruire il suo stack software del Bluetooth ma al momento supporta solo il cosiddetto "central role" ovvero il device Android può ricercare avvisi di disponibilità alla connessione da parte di dispositivi con "peripheral role" e stabilire tale connessione ma non viceversa.

1.3.3 Location

Nel corso del tempo, e più precisamente con l'avanzare delle versione di Android, Google ha raffinato gli strumenti messi a disposizione dei programmatori fornendo attraverso quelli che vengono definiti Play Services un framework di più alto livello, restano comunque allo stesso tempo ancora valide le operazioni base presenti fin dalle prima versione dell'OS di Mountain View. Le attività standard riguardanti la posizione vengono fornite dal package android.location, il quale garantisce l'accesso ai servizi di localizzazione offerti dal dispositivo. Richiamando la getSystemService(Context.LOCATION_SERVICE) otterremo un'handle a un'istanza del LocationManager, un system service su cui possono essere eseguite le seguenti operazioni:

- richiesta della lista di tutti i LocationProvider delle ultime posizioni dell'utente;
- attivare e disattivare la registrazione per aggiornamenti periodici della posizione corrente attraverso un LocationProvider;
- attivare e disattivare l'azionamento di un Intent da lanciare in caso il dispositivo entri all'interno di una zona geografica definita da coordinate spaziali.

In questo modo abbiamo la possibilità di ottenere le informazioni che desideriamo, ma allo stesso tempo si devono considerare anche numerose problematiche riconducibili a fattori differenti:

- GPS, Wi-Fi e id della cella di rete a cui si è connessi sono tutte sorgenti valide ma con caratteristiche di accuratezza, affidabilità e consumo energetico ben differenti;

- si presume valida la possibilità che l'utente e di conseguenza il dispositivo si muovano e quindi debbano essere effettuate valutazioni sugli intervalli di campionamento;

- non va scartata l'eventualità che la precisione di una sorgente i cui dati risalgono a 10 secondi fa sia in ogni caso maggiore rispetto a quella di dati più recenti ma derivanti da una sorgente diversa.

Come anticipato, Google offre inoltre delle API legate ai Location Services in grado di garantire la possibilità di scegliere la tipologia di Location Provider oltre a gestire il consumo energetico e ad individuare l'attività corrente dell'utente; tutto questo può essere racchiuso in tre caratteristiche molto importanti:

- Il fused location provider permette di stabilire necessità di alto livello come l'accuratezza richiesta e il consumo energetico (es. high accuracy o low power), di ottenere accesso alla posizione migliore e più recente, di minimizzare il consumo energetico sfruttando la soluzione più efficiente per necessità specifiche e di soddisfare una vasta gamma di esigenze che vanno da utilizzi in foreground con alta precisione richiesta a usi in background dove invece si richiedono aggiornamenti periodici con consumi poco rilevanti;

- Il geofencing consente la configurazione di confini geometrici che delimitino aree circostanti a luoghi specifici permettendo la generazione di notifiche o il lancio di attività in base ai movimenti che l'utente compie rispetto a queste aree. Anche in questo caso si limita al minimo il consumo energetico e si effettuano correzioni sulla posizione in base alla vicinanza del geofence e all'attività corrente.

- Stiamo quindi parlando dell'Activity recognition, un passo fondamentale e necessario soprattutto con lo sviluppo di applicazioni sempre più contestuali, il quale consente di verificare appunto l'attività fisica corrente dell'utente: fermo, a piedi, in bicicletta,in macchina. In questo modo si completa la "location"

con la "movement awareness" permettendo di variare ad esempio la frequenza degli aggiornamenti richiesti in base alla tipologia di movimento che si sta effettuando.

1.3.4 Sensors API

Come anticipato in precedenza, quello che andremo ad analizzare nell'ambiente Android deve poi essere verificato nei vari dispositivi appartenenti a OEM differenti: non è infatti scontato che ogni dispositivo disponga dello stesso equipaggiamento hardware che viene in linea teorica utilizzato dalle API Android.
La maggior parte dei dispositivi è dotata di sensori di movimento che misurano l'orientamento e le diverse condizioni ambientali. Questi sensori sono in grado di fornire dati grezzi con elevata precisione e accuratezza, e sono utili se si desidera monitorare il movimento tridimensionale, il posizionamento o i cambiamenti del contesto ambientale in prossimità di un dispositivo. La piattaforma Android supporta tre grandi categorie di sensori:

- sensori di movimento: questi sensori misurano le forze di accelerazione e le forze di rotazione lungo tre assi. Questa categoria comprende gli accelerometri, i sensori di gravità, i giroscopi e i sensori dei vettore di rotazione;

- sensori ambientali: questi sensori misurano diversi parametri ambientali, come la temperatura ambientale e la pressione dell'aria, l'illuminazione, e l'umidità. Questa categoria comprende i barometri, i fotometri, e i termometri;

- sensori di posizione: questi sensori misurano la posizione fisica di un dispositivo. Tale categoria comprende i sensori di orientamento e i magnetometri.

I sensori hardware disponibili sul device vengono sfruttati per acquisire dati grezzi mediante il framework dei sensori di Android: esso fornisce diverse classi e interfacce che consentono di eseguire una grande varietà di compiti correlati ai sensori.

- SensorManager: è possibile utilizzare questa classe per creare un'istanza del service di percezione. Questa classe fornisce diversi metodi per l'accesso e l'enumerazione dei sensori, la registrazione e l'annullamento della registrazione di ascoltarori di eventi sensoriali, e l'acquisizione di informazioni di orientamento. Essa fornisce inoltre diverse costanti che vengono utilizzate per

sgnalare la precisione del sensore e la velocità di acquisizione dati impostata, oltre che per calibrare i sensori.

- Sensor: si può utilizzare questa classe per creare un'istanza di un sensore specifico. Questa classe fornisce diversi metodi che consentono di determinare le capacità di un sensore.
- SensorEvent: il sistema utilizza questa classe per creare un oggetto che fornisce informazioni riguardanti un evento sensoriale. Tale oggetto include i dati grezzi del sensore, il tipo di sensore che ha generato l'evento, l'accuratezza dei dati, e la marca temporale per l'evento.
- SensorEventListener: è possibile utilizzare questa interfaccia per creare due metodi di callback che ricevono le notifiche (eventi del sensore) quando i valori del sensore cambiano o quando cambia la precisione del sensore.

Recupero dei sensori e loro funzionalità

Di seguito un esempio pratico di come gestire tali sensori.

```
private SensorManager mSensorManager;
mSensorManager = (SensorManager)
    getSystemService(Context.SENSOR_SERVICE);
List<Sensor> deviceSensors =
    mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

Se si desidera enumerare tutti i sensori di un certo tipo, basta sostituire la costante TYPE_ALL con le costanti che vedremo in seguito (tutte precedute da "TYPE_"). Inoltre, una volta ottenuto un oggetto sensor possiamo utilizzare su di esso il metodo getMinDelay() che ritorna il minimo intervallo di tempo in microsecondi a cui tale sensore può inviare dati: i sensori che ritornano zero alla chiamata di tale metodo sono definiti non di flusso in quanto invieranno dati solo in caso cambino i valori dei parametri che sta monitorando.

Sistema di riferimento

In generale, il sensor framework utilizza un sistema standard di coordinate a 3 assi per esprimere i valori dei dati. Per la maggior parte dei sensori, il sistema di coordinate viene definito rispetto allo schermo del dispositivo quando il dispositivo viene

tenuto nel suo orientamento predefinito. In questa condizione, l'asse X è orizzontale e punta verso destra, l'asse Y è verticale e punta verso l'alto, mentre l'asse Z punta verso l'esterno della faccia dello schermo. La cosa più importante di questo sistema di riferimento è che esso non cambia nel momento in cui il dispositivo viene ruotato e quindi gli assi rimangono fissi, ma allo stesso tempo bisogna fare attenzione al fatto che la maggior parte dei tablet ha come orientamento predefinito quello in landscape.

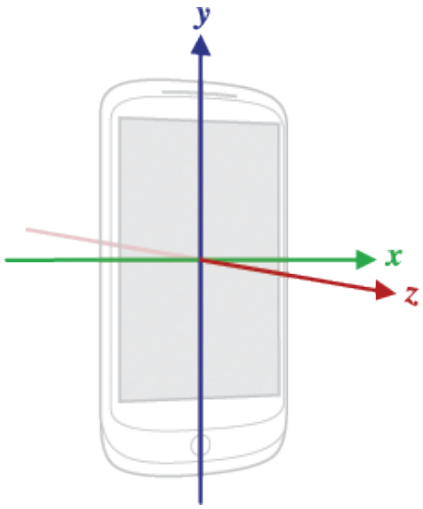

Figura 1.5: sistema di coordinate usato dalle API sensoriali

Andiamo ora ad analizzare nel dettaglio le tre classi di sensori introdotte in precedenza.

Sensori di movimento

I sensori di movimento sono utili per il monitoraggio dei movimenti del dispositivo, come ad esempio l'inclinazione, lo scuotimento e la rotazione. Il movimento è di solito un risultato dell'input diretto dell'utente, ma può anche essere un riflesso dovuto all'ambiente fisico in cui il dispositivo è posizionato. Nel primo caso, si sta monitorando il movimento relativo alla struttura di riferimento del dispositivo, nel secondo invece si considera il movimento relativo alla struttura di riferimento del mondo.
Tutti i sensori di movimento ritornano un array multidimensionale relativo ai valori del sensore per ogni SensorEvent generato:

- **TYPE_ACCELEROMETER** (Hardware)

Accelerazioni lungo gli assi x, y, z con inclusa la gravità (m/s^2)

- **TYPE_GRAVITY** (Software or Hardware)
  Accelerazioni di gravità lungo gli assi x, y , z (m/s^2)

- **TYPE_GYROSCOPE** (Hardware)
  Velocità di rotazione lungo gli assi x, y, z (rad/s)

- **TYPE_ROTATION_VECTOR** (Software or Hardware)
  Componenti lungo gli assi x, y, z e componente scalare del vettore di rotazione

- **TYPE_LINEAR_ACCELERATION** (Software or Hardware)
  Accelerazioni lungo gli assi x, y, z esclusa la gravità (m/s^2)

Sensori ambientali

La piattaforma Android supporta quattro sensori che consentono di monitorare le varie proprietà ambientali: è possibile utilizzare questi sensori per monitorare l'umidità relativa ambientale, l'illuminazione, la pressione ambiente, e la temperatura ambiente in prossimità di un dispositivo. Tutti i sensori sono basati sull'hardware e sono quindi disponibili solo se il produttore del dispositivo li ha inseriti nel circuito integrato.
Anche in questo caso viene restituito un array ma monodimensionale quindi tutti i valori si trovano nella posizione 0:

- **TYPE_TEMPERATURE** (Hardware)
  Temperatura del dispositivo (° C), deprecato

- **TYPE_AMBIENT_TEMPERATURE** (Hardware)
  Temperatura ambientale (° C)

- **TYPE_LIGHT** (Hardware)
  Illuminazione (lx)

- **TYPE_PRESSURE** (Hardware)
  Pressione ambientale (hPa o mbar)

- **TYPE_RELATIVE_HUMIDITY** (Hardware)
  Umidità relativa dell'ambiente (%)

Sensori di posizione

La piattaforma Android supporta due sensori che consentono di determinare la posizione del dispositivo: il sensore di campo magnetico terrestre e il sensore di orientamento; essa offre inoltre un sensore che consente di determinare quanto il display sia vicino a un oggetto (sensore di prossimità). Il sensore di orientamento è basato su software e trae i suoi dati dall'accelerometro e dal sensore di campo magnetico terrestre. In questo caso per ogni SensorEvent generato i sensori ritornano valori incapsualti in array (uno monodimensionale, gli altri multidimensionale):

- **TYPE_ORIENTATION** (Software), disponibile ma deprecato
  Angoli lungo gli assi x, y, z ovvero azimuth, beccheggio e rollio (°)
- **TYPE_MAGNETIC_FIELD** (Hardware)
  Forza del campo geomagnetico lungo gli assi x, y, z (μT)
- **TYPE_PROXIMITY** (Hardware)
  Distanza dall'oggetto (cm)

1.4 Altri dispositivi

Nello stesso raggruppamento dei classici smartphone citati in precedenza, possiamo includere anche le ultime novità sul mercato che contengono lo stesso tipo di sensori. Esempio pratico sono gli smartwatch, tanto in voga nell'ultimo semestre, i quali contengono accelerometro e giroscopio e i cui dati possono essere processati dal sistema operativo come visto in precedenza. Discorso a parte meritano i laptop, nei quali possiamo trovare degli accelerometri inseriti allo scopo di individuare una caduta in atto e reagire con delle azioni che preservino l'integrità del computer. Tali meccanismi sono presenti solo nei laptop di ultima generazione e solo per alcuni brand, ma in questo caso non è possibile accedere a tali dati se non per i sistemi di protezione stessi.

1.4.1 BLE e Wi-Fi tags in concreto, primi prodotti

Una startup statunitense, Estimote, ha già realizzato dei trasmettitori che sfruttino tale tecnologia inviando informazioni contestuali nel proprio raggio d'azione: fra gli

utilizzi consigliati, un video mostra direttamente l'esempio di un negozio dove il consumatore viene notificato sul proprio smartphone a proposito di offerte, informazioni e recensioni riguardanti i prodotti nelle vicinanze. Tra le caratteristiche riportate spicca la durata di tali trasmettitori, la cui efficacia viene dichiarata pari a due anni con una singola batteria da orologio, notiamo inoltre che essi trasmettono segnali Bluetooth 4.0 a una frequenza di 2.4 GHz. Come dicevamo in precedenza, i costi sono ancora uno svantaggio in quanto 3 trasmettitori sono preordinabili attualmente ad un prezzo di 99$ ma se consideriamo il raggio di azione coperto da essi (in linea teorica fino a 7500 metri quadri ciascuno) capiamo quanto essi possano essere sfruttati, sono infatti presenti anche un accelerometro e un sensore di temperatura che permettono quindi l'invio di informazioni non solamente collegate alla posizione.
Per utilizzare questi beacon viene fornito un sdk da sfruttare nello sviluppo di applicazioni mobili, le quali opereranno in background ed attiveranno azioni diverse a seconda della distanza dai beacon stessi; in linea teorica tutti gli smartphone che supportano il Bluetooth 4.0 sono in grado di interagire con questi prodotti (lista aggiornata qui).

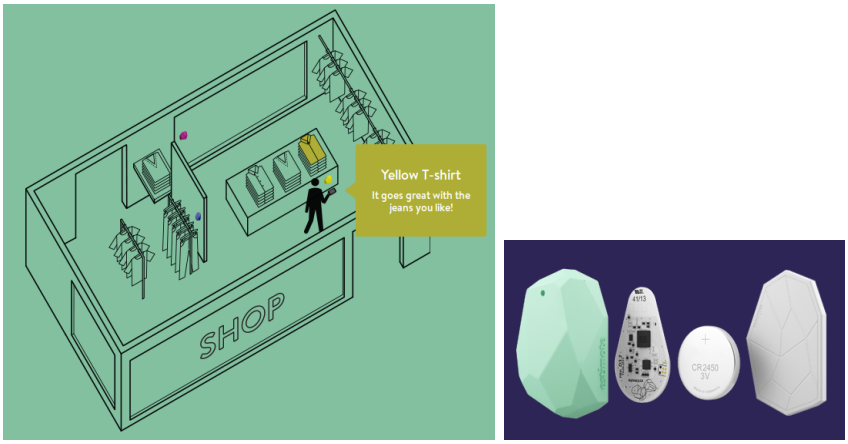

Figura 1.6: esempio di utilizzo e aspetto dei beacon

Un meccanismo simile viene sfruttato attraverso un'altra tecnologia da Navizon per il suo Indoor Triangulation System: in questo caso l'area da monitorare viene coperta attraverso una rete Wi-Fi di nodi forniti dall'azienda, i quali hanno un raggio di azione stimato in 50 metri e non necessitano altro che di essere collegati ad una presa di corrente. Dopo una breve configurazione della rete attraverso un sito specifico, durante la quale si crea una sorta di mappa dell'edificio e vi si colloca ciascun nodo, ogni dispositivo mobile avente il Wi-Fi attivato può essere localizzato e visualizzato all'interno della mappa mantenendo sotto controllo tutti gli spostamenti dell'individuo che ne è in possesso. L'azienda fornisce inoltre dei tag specifici utilizzabili per esempio in alternativa agli smartphone che rendono più precisa la localizzazione: questi tag non sono altro che trasmettitori radio alloggiati all'interno di un involucro resistente assieme alle loro batterie, la loro presenza e i loro movimenti vengono poi rilevati dal sistema che attraverso la triangolazione determina la posizione corretta. Essi trasmettono un segnale ogni 2 secondi attraverso il classico protocollo IEEE 802.11b, sono identificati da un MAC address univoco di 48 bit e hanno una durata della batteria stimata in 6 mesi. A questo indirizzo è possibile visualizzare una dimostrazione del funzionamento dei tag e di questo sistema all'interno di un edificio.

1.5 Sintesi

Dopo aver analizzato in dettaglio tutte gli strumenti a nostra disposizione, riassumiamo attraverso la seguente tabella quanto visto in precedenza.

Figura 1.7: esempi di utilizzo dei tag

CARATTERISTICA	APPLE (IOS)	GOOGLE (ANDROID) & co.
Motion coprocessor	M7	PNI Sentral sensor fusion hub (hub/fusion chip)
Comunicazoni basso consumo	IBeacon	Bluetooth Low Energy (o Smart Bluetooth)
Proprietà temporali dell'evento	CMLogItem	proprietà timestamp di SensorEvent
Posizione	CLLocation	Location
Direzione	CLHeading	Bearing
Geofencing	CLBeacon / CL BeaconRegion / CLRegion	Geofence / ACCESS_FINE_LOCATION
Stato del dispositivo	CMMotionActivity	ACTIVITY_RECOGNITION
Accelerazione utente e gravità (accelerometro)	CMAcceleration (asse x,y,z)	TYPE_ACCELEROMETER / TYPE_LINEAR_ACCELERATION / TYPE_GRAVITY
Orientamento	CMAttitude (matrice rotazione, angoli Eulero, quaternione)	TYPE_ORIENTATION
Velocità rotazione (giroscopio)	CMGyroData	TYPE_ROTATION_VECTOR / TYPE_GYROSCOPE
Campo magnetico (magnetometro)	CMMagnetometerData	TYPE_MAGNETIC_FIELD
Contapassi	CMStepCounter	TYPE_STEP_COUNTER / TYPE_STEP_DETECTOR
Distanza dall'oggetto	X	TYPE_PROXIMITY
Temperatura del dispositivo	X	TYPE_TEMPERATURE
Temperatura ambientale	X	TYPE_AMBIENT_TEMPERATURE
Illuminazione ambientale	X	TYPE_LIGHT
Pressione ambientale	X	TYPE_PRESSURE
Umidità relativa dell'ambiente (%)	X	TYPE_RELATIVE_HUMIDITY
Oggetto aggregato	CMDeviceMotion	SensorEvent

Figura 1.8: tabella che associa ad ogni caratteristica gli strumenti a disposizione

Capitolo 2

ARCHITETTURA DI PROGRAMMAZIONE AGGREGATA

Solitamente quando si parla di mobilità si fa riferimento a dispositivi mobili che compiono evoluzioni spaziali in determinati istanti temporali: queste evoluzioni vengono portate a termine all'interno di sistemi complessi, i quali a loro volta possono essere costituiti da altri dispositivi mobili. C'è quindi la necessità di elaborare informazioni raccogliendo dati provenienti da diversi dispositivi con la finalità di situare e assegnare il giusto significato ai dati di ciascun dispositivo.

2.1 Da livello individuale a livello aggregato

L'elemento critico nel colmare il gap presente fra un utente e un sistema costituito da molti dispositivi, è quindi la raccolta dati da globale a locale, la quale deve essere facilitata dallo sviluppo di linguaggi di programmazione adatti a descrivere comportamenti aggregati. Tutti i domini che hanno a che fare con queste problematiche condividono una stretta relazione fra la computazione e la disposizione dei device nello spazio. Questi sistemi sono quindi considerati computer spaziali ovvero collezioni di dispositivi che computano localmente e sono distribuiti attraverso uno spazio fisico in cui:

- la difficoltà di muovere informazioni è strettamente dipendente dalla distanza dei dispositivi che devono comunicare;
- gli obiettivi funzionali del sistema sono definiti nei termini della struttura spaziale del sistema.

Questa correlazione fra computazione locale e globale può essere sfruttata per aiutare a colmare il gap esistente fra individuale e aggregato. La geometria e la topologia includono concetti intermedi come le regioni, il vicinato e i flussi, i quali possono anche essere utilizzati come blocchi per la programmazione di applicazioni in modo da venire automaticamente mappati col comportamento dei singoli dispositivi. Quanto detto finora viene rappresentato in maniera diretta dalla figura seguente:

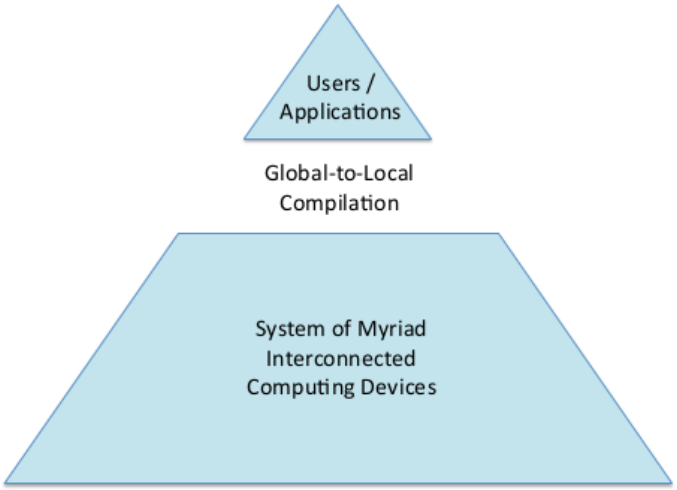

Figura 2.1: livelli generici della programmazione aggregata

Cerchiamo quindi di analizzare in dettaglio questa architettura piramidale espandendo i 3 livelli ed esplicitando ciò di cui dovremo occuparci nell'affrontare la programmazione aggregata.

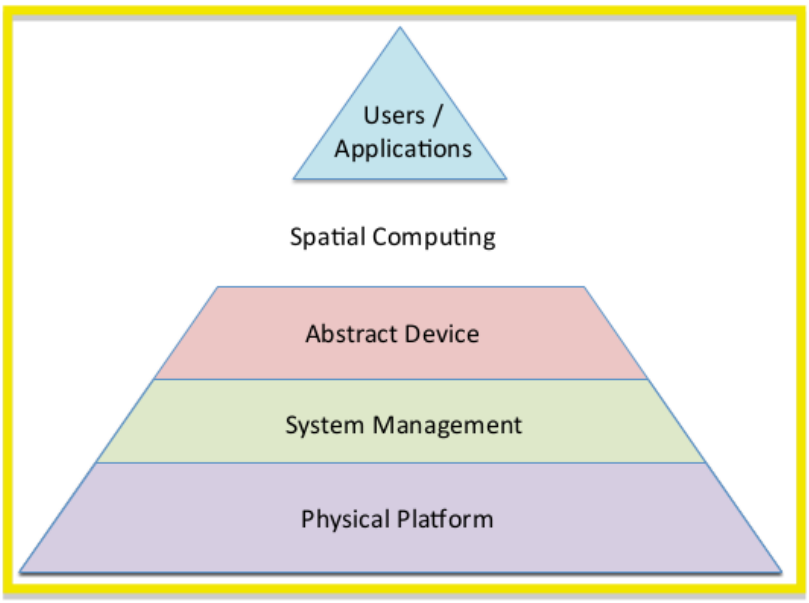

Figura 2.2: livelli specifici della programmazione aggregata

Nella figura in questione la dimensione della base di ciascun livello sta ad indicare la quantità di dettaglio implementativo, mentre i 3 livelli inferiori sono relativi esclusivamente a dispositivi individuali e locali. Analizziamo quindi la piramide partendo dal basso:

- **Piattaforma fisica:** tale livello alla base è il mezzo su cui verranno effettivamente realizzate le computazioni, questo può corrispondere ad un qualsiasi dispositivo da uno smartphone ad un device virtuale per le simulazioni.
- **Gestione del sistema:** è il livello dove risiedono il sistema operativo e tutti i servizi relativi presenti nella piattaforma, ad esempio per dispositivi embedded ci riferiamo ai driver e ai protocolli di rete di basso livello.
- **Dispositivo astratto:** nasconde i dettagli del dispositivo presentando un'interfaccia semplificata verso altri dispositivi con cui interagisce, per esempio potrebbe astrarre una complessa piattaforma embedded come un nodo di un grafo che esegue periodicamente computazioni sui suo lati.
- A questo punto si presenta il gap di cui abbiamo parlato finora, che consiste nel collegamento fra la programmazione di dispositivi individuali e le esigenze

degli utenti legate ad un livello aggregato. Tale livello, definito **computazione spaziale**, consiste nell'insieme di astrazioni e modelli che permettono quindi la connessione tra singoli e aggregati.

- **Utenti/Applicazioni:** elementi che vogliono avere a che fare con i comportamenti degli aggregati e non con i singoli dispositivi.

2.2 Modello architetturale di un sistema aggregato

Partiamo dal concetto che quello che chiamiamo sistema è un insieme di membri con una certa struttura spaziale che evolve nel tempo: definiamo quindi come prima proprietà del sistema la dimensione in termini di numero dei membri che lo compongono, essa non sarà certamente fissa ma con ogni probabilità varierà nel tempo rendendo il sistema dinamico.
Nel nostro caso ogni membro è costituito da un dispositivo dislocato nell'ambiente reale di nostro interesse, ma dobbiamo anche tenere in considerazione il fatto che l'ambiente non può essere ritenuto immutabile nel tempo quindi dovremo fare costantemente riferimento al sistema stesso e non all'ambiente che esso controlla o rappresenta. Ad esempio in figura è mostrata la pianta di un appartamento e sono raffigurati con bollini di forme e colori differenti numerosi dispositivi, in un ambiente di questo tipo possono cambiare elementi come la disposizione di mobili, divani e muri, quindi sarà fondamentale considerare le informazioni implicite ed esplicite provenienti dai dispositivi senza riferimenti all'ambiente.

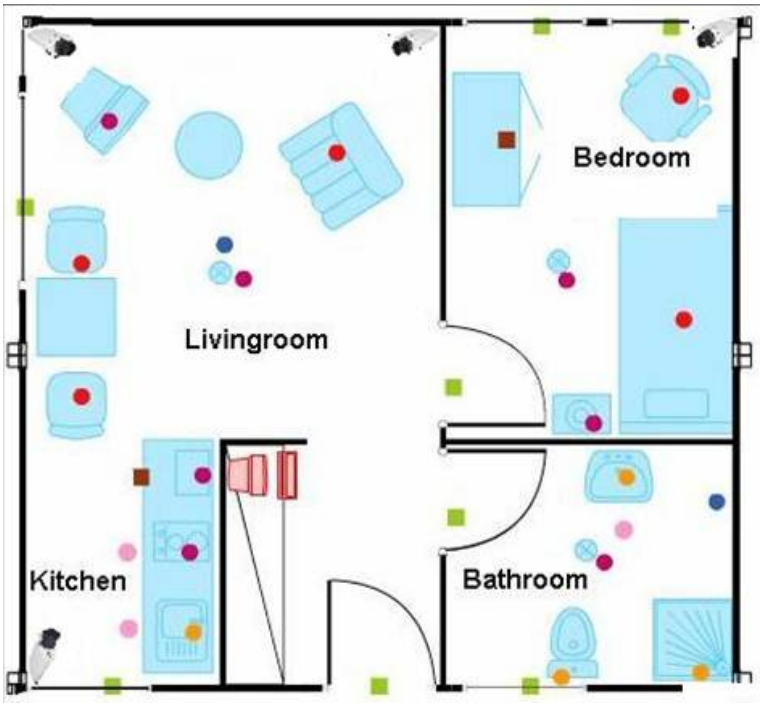

Figura 2.3: dispositivi disposti in un ambiente domestico

Nel definire un dispositivo dobbiamo innanzi tutto categorizzarlo secondo la sua attività, in base a quanto vedremo più dettagliatamente in seguito a proposito delle operazioni spazio-temporali, esso potrà quindi essere:

- un sensore, dispositivo esclusivamente in grado di rilevare caratteristiche riguardanti l'ambiente circostante, il sistema o le proprie caratteristiche rispetto ad essi;
- un attuatore, dispositivo in grado di modificare il sistema o le proprie caratteristiche rispetto ad esso;
- un ibrido, dispositivo avente sia le caratteristiche del sensore sia quelle dell'attuatore.

In seguito un dispositivo deve essere analizzato in base alla sua struttura:

- atomico, dispositivo con un'unico componente hardware in grado di rilevare o modificare una singola caratteristica;

- composto, dispositivo che presenta diversi dispositivi atomici racchiudendo almeno due componenti hardware in grado di rilevare o modificare caratteristiche diverse. In tal senso un dispositivo ibrido è nella maggior parte dei casi un composto.

Con il termine caratteristica indichiamo un aspetto specifico del dispositivo, dell'ambiente o del sistema, ci riferiamo quindi al genere di informazione trattata da un dato componente: ad esempio l'accelerazione per quanto riguarda l'accelerometro o la posizione per il GPS. Questa definizione consente di situare il dispositivo anche per quanto riguarda quindi il suo ambito di operatività ovvero il tipo di attività svolta, definiamo ora le principali aree di interesse:

- Tempo;
- Spazio dispositivo;
  - distanza;
  - posizione;
  - velocità;
  - accelerazione utente;
  - direzione;
  - rotazione;
- Misure ambientali;
  - accelerazione gravità;
  - campo magneticotemperatura ambiente;
  - illuminazione;
  - pressione;
  - umidità;
- Pattern.

Definire un sistema significa quindi collocare nell'ambiente un certo numero di dispositivi assegnandogli un determinato comportamento che può essere prestabilito dalla natura stessa del device o non prevedibile in quanto derivabile dall'interazione

con l'utente. Il sistema, dopo la fase di configurazione iniziale, può quindi essere sottoposto a cambiamenti dovuti essenzialmente solo all'inserimento/rimozione di alcuni dispositivi o allo spostamento dei dispositivi presenti (causato dall'intervento dell'utente o autonomo nel caso degli attuatori).

2.3 Modellazione device ideale e relative operazioni

Il modello del dispositivo astratto si occupa della correlazione dei device informatici e dello spazio che occupano specificando il modo in cui i dispositivi possono comunicare fra loro. In relazione alla discretizzazione dei dispositivi rispetto allo spazio, possiamo individuare 3 differenti categorie:

- i **modelli discreti** presuppongono un insieme di dispositivi che non riempiono lo spazio (reti sensoriali);
- i **modelli cellulari** assumono un insieme di dispositivi discreti che riempiono lo spazio (sistemi modulari robotici o automi cellulari);
- i **modelli continui** ipotizzano un continuo infinito di dispositivi che saranno poi approssimati attraverso l'hardware realmente a disposizione.

Identifichiamo ora tre **proprietà fondamentali della comunicazione** che descrivono i flussi di informazioni tra i dispositivi:

- **Regione di comunicazione:** è il rapporto tra i partner comunicativi di un dispositivo e lo spazio. I tipi più comuni che si incontrano sono un intorno (vicinato) a distanza limitata (anche se non necessariamente regolare) e la comunicazione globale;
- **Granularità della trasmissione:** è la modalità con cui i dispositivi trasmettono ai loro vicini. Lo stesso oggetto per tutti i vicini viene definito broadcast, un'oggetto diverso per ogni vicino rappresenta l'unicast, stesso oggetto per gruppi di vicini è chiamato invece multicast;
- **Mobilità del codice:** rappresenta la struttura del codice. I dispositivi possono avere al loro interno lo stesso programma uniforme (che può eseguire operazioni differenti a seconda dello stato e delle condizioni ambientali), in

alternativa possono avere programmi differenti ma statici, oppure è il codice a poter essere passato da un dispositivo all'altro.

Gli elementi di base della computazione spaziale sono costituiti dal volume spazio-temporale e dalle informazioni relative alla localizzazione nel suddetto volume; la dualità fra questi elementi implica quattro **classi generali di operazioni**, oltre a meta-operazioni che manipolano i calcoli:

- **Misurare Spazio-Tempo:** si tratta di operatori che prendono le proprietà geometriche del volume spazio-temporale e le traducono in informazioni. Esempi sono le misure di distanza, l'angolo, la durata del tempo, la zona, la densità e la curvatura.

- **Manipolare Spazio-Tempo:** questi operatori sono il contrario di quelli di misura, essendo attuatori che prendono informazioni e modificano le proprietà del volume di spazio-tempo. Esempi sono dispositivi che si muovono, cambiando curvatura, espandendo o contraendo localmente lo spazio, o cambiando le proprietà fisiche locali quali la rigidità che interesserà direttamente l'evoluzione fisica del sistema.

- **Calcolare Pattern:** queste operazioni che rimangono puramente nel mondo informativo possono essere visualizzate in un alto livello di astrazione, come pattern di calcolo spazio-temporali. Ad esempio, le strisce sono un pattern su spazio, un temporizzatore è un modello nel tempo, e un'onda sinusoidale di moltiplicazione è un modello nello spazio-tempo. Questa categoria comprende non solo il calcolo, ma la gran parte della comunicazione e qualsiasi sensore o attuatore puntuale che non interagisce direttamente con la geometria, ad esempio un sensore di luce o suono o un attuatore LED.

- **Evoluzione fisica:** Molti sistemi fisicamente istanziati hanno dinamiche intrinseche che causano il cambiamento della forma dello spazio nel tempo anche senza l'uso di attuatori per manipolare spazio-tempo. Gli esempi includono il moto inerziale di robot o le forze adesive che plasmano una colonia di cellule. Per loro natura, queste operazioni non sono direttamente parte dei programmi, ma i linguaggi devono assumere che tali dinamiche siano presenti nelle operazioni o abbiano attività indirizzate a controllarle.

Ogni computazione spaziale può essere descritta in termini di queste quattro classi di operazioni; tuttavia stiamo parlando di linguaggi e dobbiamo considerare anche le meta- operazioni che possono essere usate per combinare e modulare calcoli spaziali:

- le operazioni di astrazione e composizione nascondono i dettagli di implementazione dei calcoli spaziali e consentono loro di venire combinate insieme;
- le operazioni di restrizione modulano una computazione spaziale selezionando un particolare sottospazio su cui deve essere eseguito il calcolo.

Come analizzato nel capitolo precedente, sono presenti sul mercato dispositivi prodotti da vari OEM e che presentano diversi sistemi operativi in grado di sfruttare le caratteristiche hardwarepresenti al loro interno. Andiamo quindi ad analizzare tali caratteristiche situandole in base alle categorie dello spatial computing. Tutto ciò di cui abbiamo parlato consiste nella raccolta di dati provenienti direttamente da sensori implementati tramite hardware presente sul dispositivo, si tratta quindi di operazioni che **misurano lo spazio e il tempo**:

- Rilevazione istante temporale: timer;
- Rilevazione posizione: GPS, Wi-fi, id cella rete;
- Rilevazione accelerazione dispositivo lungo i 3 assi: accelerometro;
- Rilevazione accelerazione gravità lungo i 3 assi: accelerometro;
- Rilevazione rotazione del dispositivo: giroscopio+accelerometro;
- Rilevazione velocità di rotazione del dispositivo: giroscopio;
- Rilevazione campo magnetico: magnetometro;
- Rilevazione distanza dispositivo: sensore di prossimità;
- Rilevazione temperatura dispositivo: termometro;
- Rilevazione temperatura ambientale: termometro;
- Rilevazione illuminazione ambientale: sensore luminosità;
- Rilevazione pressione ambientale: barometro;

- Rilevazione umidità relativa ambientale: sensore umidità.

I casi difformi dai precedenti sono rappresentati da **meta-operazioni di composizione e astrazione** che elaborano dati grezzi provenienti anche da sorgenti diverse allo scopo di generare informazioni specifiche:

- Rilevazione numero passi effettuati;
- Rilevazione direzione di spostamento.

Infine ricadono nella categoria del **calcolo di pattern** alcune delle personalizzazioni introdotte da produttori come Samsung al sistema operativo Android di base o le ultime tecniche di identificazione Apple: anche in questo caso i dati di basso livello vengono combinati ma per identificare un vero e proprio pattern che caratterizzi il comportamento dell'utente:

- Rilevazione tipologia attività del dispositivo;
- Rilevazione posizione rispetto ad aree prestabilite (geofencing);
- Rilevazione gesture: sensore infrarossi passivo;
- Rilevatore impronte digitali: touch id.

Figura 2.4: sensori disponibili nel Samsung Galaxy S4

Figura 2.5: componenti hardware dell'Iphone 5s

Se vogliamo andare oltre le disponibilità hardware degli attuali dispositivi mobili intesi come smartphone e tablet, possiamo includere le caratteristiche che ci vengono offerte dai cosiddettti accessori, i quali ovviamente hanno la possibilità di interfacciarsi con tali dispositivi costituendone di fatto una loro naturale estensione.
In questi termini potrebbero risultare utili alle nostre necessità le cosiddette fasce per il fitness, come ad esempio la PSM Training ECHO di Zephyr, in grado di fornirci diverse misure fisiologiche dell'utente che stiamo monitorando:

- frequenza cardiaca;
- frequenza respiratoria;
- informazioni provenienti dall'accelerometro;
- intensità e carico dell'attività svolta;
- stima della temperatura corporea;
- zone dell'allenamento;
- postura;
- tipo di attività che si sta svolgendo.

Figura 2.6: fascia Zephyr PSM Training ECHO

Un'ulteriore passo che si può compiere in questo senso è quello di considerare come possibili estensioni anche sensori che consentano di rilevare la presenza di ostacoli o di movimenti.
A tal proposito, si è parlato in precedenza di infrarossi passivi per percepire particolari gesture effettuate dall'utente e quindi ci si riferiva alla parte del dispositivo rivolta verso l'utilizzatore stesso. Se teniamo invece in considerazione anche la parte posteriore, c'è la possibilità di utilizzare la fotocamera o un qualsiasi altro rilevatore, infrarossi o ultrasuoni che siano, allo scopo di esaminare lo spazio che ci si prospetta lungo la nostra direzione di movimento sfruttandone le caratteristiche.
In questo caso non stiamo ovviamente parlando di accessori immediati ma si rende necessario l'utilizzo di algoritmi specifici per analizzare i dati recuperati ed ottenere informazioni concrete come la distanza dall'oggetto più vicino o la pendenza del percorso su cui ci stiamo muovendo.
Come esempio concreto di queste tecnologie possiamo considerare il Bosch GLM 100, uno speciale metro laser in grado di funzionare sia autonomamente che abbinato ad uno smartphone tramite connessione bluetooth. Oltre alle classiche funzioni di misura lineare, di superficie e di volume, esso consente il trasferimento delle misure rilevate verso il proprio smartphone e l'associazione di tali misure ad una fotografia scattata con quest'ultimo: il tutto grazie ad un'applicazione ad hoc sviluppata da Bosch per IOS e Android.

Figura 2.7: caratteristiche e funzionamento del Bosch GLM 100

2.4 Definizione delle astrazioni spaziali

Dopo aver visto tutte le possibilità messe a disposizione dai dispositivi moderni, è giunto il momento di concentrarci esclusivamente su aspetti spaziali, possiamo quindi inizialmente separare le caratteristiche attraverso due tipologie di coordinate:

- **coordinate d'assetto:** avendo il dispositivo un volume, esse riguardano il suo orientamento e la disposizione dell'oggetto in riferimento a una posizione "di riposo" prestabilita (es. dispositivo a forma di parallelepipedo appoggiato con la base maggiore su un piano orizzontale con la faccia dello schermo rivolta verso l'alto);

- **coordinate posizionali:** con le dovute approssimazioni un dispositivo può essere rappresentato come un oggetto puntiforme senza un volume, in tale situazione ci si riferisce alla sua posizione nello spazio come a quella di un punto identificabile con il suo baricentro. In questa casistica rientrano anche le informazioni riguardanti gli spostamenti e quindi direzione, verso, velocità e accelerazione.

Quando si parla di spazio bisogna innanzi tutto effettuare una distinzione fra spazio fisico e spazio virtuale, a maggior ragione nel nostro caso in cui la localizzazione può avvenire attraverso meccanismi differenti.

La **posizione fisica** si riferisce appunto al luogo in cui è situato il dispositivo che sta eseguendo un programma o sui cui è montato hardware specifico per la localizzazione, stiamo parlando quindi di uno spazio in cui c'è la possibilità di muoversi in modo continuo. Essa può essere esplicitata in modi diversi:

- assoluta: latitudine, longitudine, altitudine;

- geografica: associazione con entità cartografiche (es. via Sacchi 3, Cesena, FC, Italia);

- organizzativa: si fa riferimento a una suddivisione dello spazio effettuata in base a esigenze strutturali e di coordinazione (es. stanza 5 del DISI).

Queste tre tipologie di identificazione della posizione definiscono un livello di dettaglio molto differente fra loro e devono quindi essere scelte in base alla necessità di precisione ma anche allo scopo della localizzazione: se ad esempio il sistema è

concentrato sul movimento di un utente all'interno di uno stesso dipartimento, a poco serviranno le corrdinate geografiche o l'indirizzo del dipartimento mentre avremo la necessità di sapere in quale stanza o corridoio si trova il dispositivo in questione. Mentre le prime due tipologie fanno uso di sistemi di riferimento definiti universalmente, nel terzo caso saranno i costruttori del sistema a dover delineare e nominare le zone di interesse secondo le caratteristiche dell'ambiente rendendo quindi necessaria una fase di configurazione iniziale in cui verranno definite le regioni significative.

La **posizione virtuale** si riferisce invece a concetti definiti dal sistema che abbiamo sviluppato, per ovvie ragioni in un sistema che utilizza la rete possiamo fare riferimento al nodo della rete a cui il dispositivo è connesso e in cui vengono eseguite le astrazioni delle operazioni. In questo caso lo spazio è discreto in quanto si passerà ad esempio da un nodo ad un'altro senza soluzioni di continuità; con questa struttura si rende possibile anche distinguere fra un posizionamento di tipo assoluto (indirizzo IP) e uno di tipo relativo (domini e sottodomini identificabili tramite DNS). In questo caso i ragionamenti devono essere quindi effettuati sul valore degli indirizzi IP dei dispositivi: la situazione in regime di connessione Wi-Fi alla stessa rete non prevede grossi intoppi per quel che riguarda le considerazioni fra dispositivi diversi in quanto la struttura della rete determina un assegnamento di indirizzi nella rete locale che ci consente di collocare un dispositivo nella suddetta rete. Stesso discorso non può essere effettuato in regime di connessione dati in quanto non c'è un meccanismo identificabile che ci permetta di estrapolare informazioni dall'indirizzo IP assegnato a ciascun dispositivo, tale comportamento è inoltre fortemente dipendente dai singoli operatori: non ci resta quindi che considerare l'id della cella a cui si è connessi valutando eventuali cambiamenti o variazioni della potenza del segnale.

Andiamo ora ad analizzare tutto quello che concerne il **moto** che può compiere un dispositivo autonomamente (per esempio un robot) oppure tramite l'intervento dell'utente. Il moto in sè non è altro che un cambiamento di posizione di un corpo in funzione del tempo, stiamo quindi parlando di uno spostamento che deve essere ovviamente caratterizzato:

- qualitativamente: tramite un punto di partenza, una direzione e un verso, oltre che un punto di arrivo (se esso è già terminato o se comunque ne è programmata la fine);

- quantitativamente: da dei valori che ne identificano la velocità e l'accelerazione (nel caso in cui non si tratti di un moto uniforme).

Le feature presenti negli OS mobili visti all'interno del capitolo 1 ci danno inoltre lo spunto per caratterizzare il moto secondo dei pattern riconducibili alle abitudini umane:

- camminata (movimento contraddistinto da velocità ridotta e pressochè costante, indicativamente poco sopra il m/s ovvero circa 4 km/h);
- corsa (movimento denotato da velocità media e pressochè costante, indicativamente sotto i 3 m/s ovvero 10 km/h);
- guida (movimento contraddistinto da velocità fortemente discontinue, che si aggirano in media attorno ai 18 m/s ovvero 65 km/h, intervallate da possibili momenti di sosta).

La descrizione è stata fin qui effettuata in linea puramente generale tenendo in considerazione soprattutto i criteri derivanti dall'analisi fisica di questo concetto, se esaminiamo invece l'aspetto riguardante lo spazio virtuale uno spostamento rilevabile può essere caratterizzato da:

- passaggio da un nodo ad un altro all'interno della stessa rete (Wi-Fi);
- passaggio da una cella ad un altra all'interno della rete dati;
- passaggio da 3g a Wi-Fi o viceversa (dovuto all'entrata/uscita dalla zona di copertura del router).

A livello implementativo, Android consente di rilevare tutti questi cambiamenti e di programmare quindi azioni in risposta agli eventi in questione: attualmente però c'è un solo limite ed è costituito dal fatto che il cambiamento di cella è rilevabile solamente a schermo acceso, si ritiene quindi indispensabile mantenere attivo il dispositivo altrimenti esso si accorgerà del passaggio solo al momento del risveglio.

2.5 Eventi e generatori di eventi

In seguito alle riflessioni svolte soprattutto all'interno del capitolo 2.2, nel modellare i generatori di eventi del sistema non possiamo che realizzare un mapping univo-

co con i dispositivi; nel fare ciò è però necessario effettuare una semplificazione assumendo che dopo la fase di configurazione il sistema sia immutabile a meno di eventuali effetti derivanti dalle attività dei dispositivi: stiamo quindi escludendo interventi umani differenti dal semplice trasporto di uno smartphone che considereremo come iniziativa intrinseca del dispositivo stesso. Più precisamente possiamo identificare un generatore di eventi con quelli che sono stati definiti dispositivi atomici o composti, quindi entità aventi una o più caratteristiche. Nel descrivere gli eventi effettuiamo subito una prima distinzione fra eventi di rilevazione ed eventi di trasformazione, tale separazione è facilmente derivabile dalla classificazione del dispositivo in sensore/attuatore. Ogni evento sarà quindi identificato da:

- tipologia dell'evento (rilevazione/trasformazione);
- responsabile della generazione dell'evento (dispositivo generatore);
- caratteristica dell'evento (nel modo in cui è stata descritta precedentemente e quindi intesa come ambito di interesse), prevede possibili sotto-categorie;
- misurazione temporale (a seconda della caratteristica possono essere disponibili istante di generazione e/o durata dell'evento);
- valore che quantifica la caratteristica dell'evento.

Dopo aver definito il modello architetturale, andiamo a enumerare e codificare i principali eventi possibili posizionandoli all'interno della seguente tabella: nella colonna "caratteristica" il campo preceduto da ":" indica una sottocategoria della categoria seguita da ":". In maniera semplicistica abbiamo deciso di modellare solo gli eventi di trasformazione riconducibili ad una specifica volontà esprimibile attraverso comandi diretti: per esempio si può ordinare ad un robot di raggiungere un determinato punto dello spazio, muoversi in una direzione con una determinata velocità e accelerazione e lo stesso può essere richiesto in linea di massima ad un essere umano; al contrario non è fattibile imporre in fase ti trasformazione le altre caratteristiche viste in fase di rilevazione.

TIPOLOGIA	GENERATORE	CARATTERISTICA	TEMPO	VALORE
Rilevazione	GPS	posizione: coordinate complete	istante	latitudine, longitudine, altitudine
Rilevazione	Wi-Fi / 3g	posizione: coordinate semplificate	istante	latitudine, longitudine
Rilevazione	Rilevatore ottico	posizione: distanza	istante	m
Rilevazione	GPS / Wi-Fi / 3g	posizione: movimento: direzione	istante e/o durata	equazione retta o valori geometrici che la definiscono
Rilevazione	Accelerometro	posizione: movimento: accelerazione	istante	valori 3 assi in m/s2
Rilevazione	GPS / Wi-Fi / 3g / Bluetooth	posizione: movimento: regione	istante	identificativo regione ed espressione entrata/uscita
Rilevazione	pedometro	posizione: movimento: contapassi	istante	valore adimensionale
Rilevazione	Sensore rotazione	posizione: rotazione: vettore	istante	valori adimensionali 3 assi
Rilevazione	Giroscopio	posizione: rotazione: variazione velocità	istante	valori 3 assi in rad/s
Rilevazione	Dispositivi complessi	stato: attività	istante e/o durata	non identificato / fermo / camminata / corsa / guida
Rilevazione	Magnetometro	stato: campo magnetico	istante	valori 3 assi in µT

Figura 2.8: tabella che individua gli eventi di nostro interesse

Nella tabella non è presente la velocità nella fase di rilevazione in quanto non vi sono meccanismi diretti di misurazione della velocità ma si utilizzano metodi che combinano dati provenienti da componenti diverse e non sono precisi in relazione a distanze brevi.

Rilevazione	Sensore gravità	stato: accelerazione gravità	istante	valori 3 assi in m/s2
Rilevazione	Termometro	stato: temperatura dispositivo	istante	°C
Rilevazione	Termometro	stato: temperaura ambiente	istante	°C
Rilevazione	Barometro	stato: pressione	istante	Pa
Rilevazione	Luxmetro	stato: illuminazione	istante	lx
Rilevazione	Igrometro	stato: umidità	istante	%
Trasformazione	Dispositivo con capacità motorie o trasportabile	posizione: coordinate complete	istante inizio movimento	latitudine, longitudine, altitudine
Trasformazione	Dispositivo con capacità motorie o trasportabile	posizione: coordinate semplificate	istante inizio movimento	latitudine, longitudine
Trasformazione	Dispositivo con capacità motorie o trasportabile	posizione: movimento: direzione	istante e/o durata	equazione retta o valori geometrici che la definiscono
Trasformazione	Dispositivo con capacità motorie o trasportabile	posizione: movimento: velocità	istante inizio movimento	valori 3 assi in m/s
Trasformazione	Dispositivo con capacità motorie o trasportabile	posizione: movimento: accelerazione	istante inizio movimento	valori 3 assi in m/s2
Trasformazione	Dispositivo con capacità motorie o trasportabile	posizione: movimento: regione	istante inizio movimento	identificativo regione ed espressione entrata/uscita

Figura 2.9: seconda parte della tabella degli eventi

Capitolo 3

LINGUAGGI DI COMPUTAZIONE SPAZIALE

Dopo la breve introduzione alla programmazione aggregata presentata nel capitolo 2.1, in cui si è parlato di caratteristiche generali e della struttura architetturale di programmazione, andiamo ora ad analizzare alcuni fra i principali linguaggi che presentano prerogative in grado di renderli efficaci nell'approccio allo spatial computing. Il fine di questa rassegna è quello di definire un domain specific language che:

- sia finalizzato alla programmazione di classi di computer spaziali;
- includa esplicitamente costrutti geometrici o topologici che operano implicitamente su aggregati di dispositivi informatici;
- permetta una combinazione infinita di sistemi da specificare.

Nel corso di questa rassegna saranno presenti anche due classi di sistemi che non rispecchiano questi requisiti: tali linguaggi infatti, nonostante siano stati sviluppati per scopi affini ai nostri, non presentano costrutti esplicitamente geometrici o topologici. Essi tipicamente tentano in qualche modo di abolire lo spazio e un confronto con questi ultimi ci aiuterà a far emergere i costi e i benefici derivanti dall'inserimento di costrutti spaziali all'interno di un DSL per la programmazione aggregata.
In riferimento alla piramide della programmazione aggregata, i diversi linguaggi copriranno considerazioni e aspetti differenti in base alle priorità di sviluppo: ogni

linguaggio che prenderemo in considerazione non ricoprirà mai quindi tutte le possibili caratteristiche in quanto alcune di esse saranno fuori dal sua portata o interesse. Per prima cosa bisogna sicuramente categorizzare il linguaggio stesso e capire quale sia il suo ambito di utilizzo:

- tipologia di linguaggio di programmazione (es. funzionale, dichiarativo, imperativo);
- design pattern utilizzato (restrizione, estensione o piggyback di un altro linguaggio, oppure definito da zero e quindi completamente inventato);
- piattaforme fisiche a cui è stato originariamente destinato (domini specifici o realmente universale);
- layer intermedio sul quale si focalizza
  - spatial computing: consideriamo quali astrazioni legate allo spazio sono supportate da ogni DSL;
  - abstact device: consideriamo come i dispositivi sono correlati con lo spazio e come le informazioni si muovono attraverso il sistema;
  - system management: non verrà preso in considerazione in quanto nei sistemi moderni esso è ampiamente disconnesso da considerazioni spaziali.

3.1 Computazione amorfa

La computazione amorfa è lo studio dei sistemi di calcolo composti da un vasto numero di dispositivi molto semplici e con la capacità di comunicare solo localmente. L'obiettivo della ricerca in questo ambito è quello di individuare principi ingegneristici simili a quelli sfruttati dai sistemi naturali, liberandosi di molte delle assunzioni sulle quali l'informatica si è sempre basata: essi si dividono in pattern e manifold languages.

3.1.1 Pattern languages

La maggior parte dei linguaggi emersi dalla computazione amorfa sono riconducibili alla formazioni di pattern robusti: tra i principali ricordiamo il Growing Point Lan-

guage (GPL) e l'Origami Shape Language (OSL). Entrambi i linguaggi sono tolleranti ai cambiamenti nelle loro condizioni di esecuzione ovvero partendo da condizioni iniziali deformate si otterranno pattern finali similmente distorti.

Il primo si basa su una metafora botanica ed esprime una struttura topologica in termini di punti di crescita che costruiscono un pattern trasmettendo attività attraverso lo spazio e tropismi che attraggono o respingono il movimento di questi punti attraverso segnali chimici simulati. La combinazione di queste due primitive consente al programmatore di specificare un grafico e il suo grado di sviluppo permettendo la creazione arbitraria di pattern topologicamente complessi.

Il secondo rappresenta invece il suo complemento in termini geometrici anzichè topologici: in questo caso il programmatore specifica una sequenza di pieghe selezionandole da un catalogo di possibili mosse tratte da degli assiomi. Queste pieghe vengono quindi compilate all'interno dei vari programmi locali in modo che le interazioni locali computino le linee di piegatura producendo infine la forma specificata.

3.1.2 Manifold languages

Dall'altro lato troviamo linguaggi legati allo spatial computing in maniera molto più generale: fra questi spicca senza dubbio Proto: esso è un linguaggio di programmazione versatile puramente funzionale con una sintassi simil-LISP che utilizza un'astrazione spaziale continua chiamata "mezzo amorfo" per vedere ogni calcolatore spaziale come un'approssimazione di un manifold spazio-temporale con un dispositivo in ogni punto. Le informazioni scorrono attraverso questo manifold ad una velocità limitata e ogni dispositivo ha accesso al recente passato degli altri dispositivi al'interno di un certo raggio di vicinanza. Le primitive Proto sono operazioni matematiche su campi, ovvero funzioni che associano ogni punto nel volume spazio temporale ad un valore; esse sono suddivisibili in 4 categorie:

- computazioni ordinarie puntuali;
- operazioni di vicinato che implicano comunicazione;
- operazioni di feedback che stabiliscono variabili di stato;
- operazioni di restrizione che modulano la computazione cambiando il suo dominio.

Proto funziona compilando un programma di alto livello ed eseguendolo in ciascun nodo della rete, localmente si parla di una Proto Virtual Machine eseguibile su piattaforme diverse e anche su un simulatore. In concreto i programmi interagiscono con il loro ambiente attraverso sensori e attuatori che misurano e manipolano lo spazio occupato dai dispositivi. I linguaggi di formazione di pattern usano un'ampia varietà di rappresentazioni di alto livello ma sono comunque limitati per quanto riguarda la portata: pattern di tipo diverso sono difficili da combinare e non possono essere composti in maniera pulita. Proto non offre al programmatore astrazioni di così alto livello ma le sue primitive hanno a che fare solo con vicinati locali nel volume spazio-temporale; nonostante questo, essendo un linguaggio con semantica funzionale definita in termini di operazioni di aggregazione, alcune funzioni della libreria standard colmano questo gap offrendo operatori a livello aggregato.

3.2 Modellazione biologica

I sistemi naturali biologici hanno spesso una struttura fortemente locale e spaziale, dai biofilm di organismi monocellulari ai tessuti degli animali multicellulari: questa spazialità si riflette in numerosi linguaggi sviluppati per modellare sistemi biologici e per progettare nuovi organismi. Linguaggi di modellazione biologica esplicitamente spaziali sono ad esempio L-systems e MGS: il primo usa un modello di riscrittura dei grafi per descrivere la crescita e la struttura delle piante specificandola in termini di leggi di modifica delle strutture geometriche locali, il secondo ha un approccio simile ma consente un calcolo completamente basato su regole più generali in cui la biologia viene utilizzata come ispirazione ma anche come area d'applicazione.

I programi MGS operano sia manipolando valori locali sia modificando a livello topologico le stesse strutture locali, questo consente di esprimere modelli complessi in maniera molto semplice accoppiando tale linguaggio con un modello fisico che ne regola la geometria. Esso consente una computazione generica su complessi topologici basata su regole in cui ciascuna entità interagisce con le altre del vicinato: le interazioni sono specificate attraverso trasformazioni, le quali rappresentano una sorta di riscrittura della struttura spaziale che è composta dallo stato locale di ciascuna identità.

Recentemente si è inoltre unito a questa famiglia Gro, un linguaggio simil-Python progettato per la simulazione stocastica di reti di regolazione genetica all'interno di

una colonia crescente di Escherichia coli: esso include nozioni riguardanti la velocità delle reazioni chimiche, la comunicazione diffusiva, la crescita delle cellule e permette al programmatore di costruire modelli chimici arbitrari per controllarli.

Sebbene molti dei linguaggi per lo spazio siano focalizzati su agenti individuali, questi che elevano il loro livello di astrazione verso l'aggregato forniscono un'ampia varietà di operazioni spazio-temporali attraverso costrutti di modellazione molto potenti capaci di manipolare lo spazio sia geometricamente che topologicamente.

3.3 Modelli agent-based

I modelli agent-based sono in grado di descrivere i seguenti elementi individualmente o assieme:

- un set di agenti, i loro attributi e i loro comportamenti;
- le relazioni che intercorrono fra gli agenti e i metodi di interazione;
- l'ambiente attraverso il quale gli agenti interagiscono.

I modelli comportamentali, come il modello ad agenti Belief-Desire-Intent (BDI), descrivono le caratteristiche intrinseche degli agenti: esse possono essere concettualmente ridotte ad una visione di agenti come sensori (che leggono dall'ambiente), effettori e attuatori (che lo cambiano), mappati fra loro da un comportamento ben definito. Le relazioni fra gli agenti sono tipicamente codificate come topologie: ad esempio una topologia di rete può offrire relazioni di comunicazione di basso livello fra agenti e allo stesso tempo una rete di copertura sociale può guidare la necessità di uno scambio di messaggi fra agenti. La modellazione dell'ambiente può variare ampiamente, a seconda dell'obiettivo che si vuole raggiungere, da semplici modelli geospaziali a modelli biologici complessi: spesso le informazioni ambientali sono percepite da un agente e nello stesso modo gli attuatori modificano il modello ambientale che gli agenti utilizzano a turno come una blackboard (vedi stigmergia). I DSL possono essere categorizzati nel seguente modo:

- **Linguaggi di modellazione grafica ad agenti:** estendono e/o restringono alcune forme di UML, si concentrano sulla modellazione delle caratteristiche

degli agenti e solo a volte sui pattern di interazione preferendo strumenti grafici piuttosto che linguaggi formali; AML ad esempio mira ad estendere la documentazione UML con concetti legati agli agenti.

- **Framework ad agenti:** estendono linguaggi general-purpose che impongono strutture comuni o specifiche sugli agenti. Adattandosi a tali strutture i programmatori hanno la possibilità di utilizzare tools offerti dal framework per simulazioni, logging e amministrazione. Jade ad esempio estende Java con una libreria per la costruzione e l'interazione di agenti FIPA-compliant.

- **Set di strumenti di modellazione e simulazione multi-agente:** si concentrano sulla modellazione e sulla simulazione delle interazioni ambientali e fra agenti all'interno di sistemi complessi. ASCAPE ad esempio è un'estensione di Java che permette di semplicare la composizione del modello ad agenti e l'esecuzione del loro comportamento usando rispettivamente astrazioni topologiche ed esecuzioni basate su regole, mentre NetLogo e StarLogo sono estensioni del linguaggio Logo che permettono rilevamento e controllo di agenti multipli.

I linguaggi di modellazione grafica ad agenti sono tipicamente concentrati verso gli utenti finali e spesso costituiscono semplicemente rappresentazioni grafiche o estensioni di altri linguaggi di modellazione ad agenti, paradigmi o meta-modelli: essi lavorano su un livello concettuale mappando azioni o comportamenti nei componenti stessi del sistema ad agenti, al contrario i framework descrivono nella maggior parte dei casi estensioni di linguaggi general-purpose che offrono strumenti comuni a designers e sviluppatori.
Fra queste, l'unica classe a mostrare proprietà spaziali è quella relativa al set di strumenti di modellazione e simulazione multi-agente: ciò è dovuto principalmente ad una stretta integrazione fra il linguaggio e gli ambienti di simulazione, che consente la presenza di caratteristiche non disponibili in altri sistemi distribuiti (distanza, movimenti). I DSL agent-based offrono spesso modalità discrete di interazione fra agenti sebbene alcuni offrano una discretizzazione cellulare.
Similmente, molti DSL multi-agente tentano di astrarre la nozione di rete nei confronti del programmatore offrendogli diverse tipologie di comunicazione globale: eccezioni note si trovano ancora una volta nella terza delle classi elencate in precedenza, dove la topologia e i collegamenti di rete consentono al programmatore di

simulare restrizioni di comunicazioni fra agenti. La granularità della comunicazione è tipicamente unicast (da agente ad agente), ma alcuni linguaggi come NetLogo consentono anche una tipologia multi-cast.
Infine, alcuni DSL agent-based consentono la mobilità del codice in quanto essa rappresenta un meccanismo utilissimo per gli algoritmi distribuiti: i set di strumenti di modellazione e simulazione multi-agente solitamente eseguono un singolo programma uniforme mentre i framework ad agenti tendono ad offrire caratteristiche per facilitare la mobilità del codice.

3.4 Reti di sensori wireless

Le reti di sensori wireless sono un campo di ricerca che ha a che fare con lo sviluppo di ampie reti composte da dispositivi che realizzano funzioni di percezione: i dispositivi nella rete sono solitamente formati da componenti particolari e includono un processore, un'interfaccia dicomunicazione wireless e uno o più sensori. L'obiettivo principale delle reti di sensori wireless, composte spesso da dispositivi statici, omogenei e non, è un efficiente raccolta di dati e la loro consegna ad un'interfaccia collegata con l'infrastruttura; essi sono dispositivi autonomi con quantità di energia a disposizione per il loro funzionamento limitata, per questo motivo le principali accortezze devono quindi essere rivolte all'efficienza energetica. Allo scopo di ottimizzare questi processi, è uso comune includere meccanisimi che abbiano un duty-cycle in cui lunghi cicli a basso consumo si alternino a brevi sequenze di attività, compromessi fra comunicazioni wireless dispendiose e processamenti locali a basso consumo, comunicazioni tramite salti e algoritmi distribuiti.

Raccogliere dati verso un'unico punto partendo da una larga rete, è considerato un processo non scalabile che ha il limite di banda come vincolo principale: tecniche come l'aggregazione dei dati, la selezione di un sottoinsieme di dati, il filtraggio e il preprocessamento sono quindi operazioni che devono essere svolte nella maggior parte dei casi. La raccolta dei dati e la loro diffusione è fortemente collegata alla topologia della rete sopra la quale si opera: per esempio organizzando la rete come un grafo si possono sfruttare più semplicemente algoritmi per dispositivi con risorse limitate; altri aspetti come i pattern comunicativi sono invece un blocco chiave nella costruzione del sistema ma sono spesso dettati dall'infrastruttura hardware sottostan-

te. I DSL sviluppati per funzionare in questo settore devono quindi automatizzare e astrarre alcuni dei seguenti meccanismi.

- Le piattaforme hardware e software sono spesso specifiche per ciascuna applicazione che viene sviluppata: le operazioni più comuni che possono essere automatizzate sono il controllo dei componenti attraverso un livello di astrazione che consente il supporto alla programmazione per ogni sistema operativo. Importante caratteristica è inoltre la possibilità di accendere e spegnere i componenti reagendo a paradigmi di programmazione event-driven.

- Il controllo della topologia e della comunicazione è stato effettuato virtualmente in tutte le applicazioni di reti di sensori: è fondamentale che vengano fornite primitive che astraggano i protocolli di comunicazione, che creino e mantengano una topologia; tutto ciò si basa su scoperta del vicinato, algoritmi di routing e protocolli di trasporto.

- La diffusione dei dati, essendo uno degli obiettivi principali delle applicazioni in questo campo, deve essere supportata con primitive di alto livello per la ricerca di oggetti specifici nella rete. Questo viene raggiunto combinando algoritmi di rete con protocolli di trasporto, che assicurano la consegna dei dati attraverso percorsi ottimali.

- I vincoli di efficienza energetica portano alla necessità di predefinire o gestire automaticamente il trade-off fra l'elaborazione e la comunicazione locale. Le stime delle metriche della qualità del servizio sono un meccanismo attraverso cui ciò viene raggiunto.

Sfortunatamente questi DSL sono molto limitati nelle loro funzionalità in quanto specificatamente progettati per applicazioni di distribuzione dei dati, un altro dei motivi per cui i linguaggi legati alle reti di sensori non rispecchiano i requisiti dello spatial computing va invece individuato nella staticità delle topologie presenti nella gran parte delle impostazioni iniziali dei sistemi.

3.4.1 Region-based

La maggior parte di questi DLS rientra nella categoria region-based, adempiendo alle necessità del programmatore di esprimere operazioni a livello di regioni (vicinato, insiemi) anzichè di dispositivi singoli. In alcuni linguaggi la discriminante per la definizione di vicinanza non è soltanto la distanza da un punto ma può essere anche la presenza di caratteristiche comuni nonostante la dislocazione su nodi differenti e lontani. Fra le caratteristiche dei linguaggi che rientrano in questa categoria, possiamo notare:

- presenza di operatori spaziali che consentono di indirizzare regioni della rete;
- posibilità di ottenere informazioni a proposito del trade-off fra risorse comunicative e computazionali;
- funzionalità per condividere dati fra i nodi in maniera trasparente al programmatore.

Regiment è il linguaggio più vicino allo spatial computing nel dominio delle reti di sensori wireless: questo è dovuto alla combinazione fra la flessibilità con la quale ogni utente può specificare le regioni e l'interfaccia funzionale offerta. Esso presenta un linguaggio funzionale che consente un semplice accesso ai flussi di dati (chiamati segnali): i dettagli di basso livello sono ottenuti tramite 4 successive traduzioni del programma iniziale in linguaggi differenti impilati uno sull'altro. Il concetto di regione comprende quindi varie rappresentazioni dello spazio, la regione può essere definita come:

- la distanza da un certo punto;
- l'insieme dei nodi compreso fra un certo numero di salti partendo da un nodo;
- l'insieme dei nodi che soddisfano determinati predicati.

3.4.2 Dataflow-based

I linguaggi basati sul flusso di dati sono situabili ad un livello di astrazione superiore: anche se la loro esecuzione utilizza la nozione di vicinato come parte dell'implementazione, l'utente non necessita di accedervi direttamente. Al contrario, le applicazioni possono essere specificate attraverso un grafico riguardante il flusso dei dati, in cui

l'utente descrive come i componenti software siano collegati tramite i dati stessi. La localizzazione del software, la comunicazione fra dispositivi, il posizionamento e il trasferimento dei dati nei vicinati, sono meccanismi costruiti all'interno dei linguaggi. Questa classe non presenta quasi nessuna caratteristica in comune con l'approccio di programmazione spaziale: l'obiettivo di questi linguaggi è quello di offrire funzionalità collegando i giusti componenti software, le caratteristiche spaziali devono quindi essere implementate anch'esse come componenti al di sopra di questa struttura.

L'esempio più semplice di questa categoria è Active Messages: un meccanismo che consente la comunicazine asincrona fra componenti tramite interfacce che forniscono comandi ed eventi. Il sistema risultante è simile ad un sistema composto da socket che ha il vantaggio di offrire modularità e di abilitare una computazione event-driven.
MiLAN offre invece una selezione di sensori e gruppi di nodi basati sulla qualità del servizio di raccolta dati: l'utente specifica l'esecuzione di una macchina a stati con requisiti di qualità del servizio per ogni transizione, quindi il sistema seleziona i sensori appropriati di percezione dei dati.

3.4.3 Vista centralizzata

I DSL a vista centralizzata sono fondamentalmente un insieme di linguaggi di alto livello che si rivolgono alle reti di sensori wireless con una prospettiva differente da quella dei DSL dataflow- based: la principale distinzione si manifesta con la mancata definizione a priori della raccolta dei dati e delle funzionalità di aggregazione, questi linguaggi consentono infatti all'utente di definire le funzionalità dell'applicazione per l'intera rete con un singolo programma.

Per esempio, Pleiades consente un'esecuzione sequenziale sull'intera rete di sensori: essa è vista come un oggetto centrale, contenitore di nodi, e l'utente può definire un ciclo in cui iterare attraverso tutti i nodi. Può inoltre essere specificata l'esecuzione parallela delle istruzioni di questo ciclo con il compilatore che si occuperà poi di trasformare il programma iniziale in una collezione di algoritmi distribuiti che emuleranno la sequenzialità sul sistema distribuito. Anche in questo caso non ci sono elementi in comune con l'approccio della computazione spaziale: lo sforzo compiuto sta nell'offrire all'utente una rappresentazione discreta dello spazio dove i dati di ogni

dispositvo possono essere indirizzati individualmente usando una programmazione sequenziale.

3.4.4 Agent-based

Questi linguaggi rappresentano uno speciale sottoinsieme della categoria dei DSL per reti di sensori wireless che consentono transizioni verso sistemi più completi come le reti mobili ad hoc. L'idea base è che gli agenti software contengano le funzionalità richieste per processare i dati ed eseguire aggregazione locale mentre stanno seguendo per esempio un particolare evento fisico attraverso la rete. Essi lavorano anche al di sopra della regione informativa ma consentono lo sviluppo di applicazioni più complesse rispetto alla raccolta dati e alla diffusione.

In Agilla, un framework ad agenti costruito su TinyOs, gli agenti si muovono fra i nodi assieme al loro stato: ogni nodo mantiene uno spazio di tuple che consente l'interazione fra gli agenti, i quali sono indirizzabili attraverso la posizione piuttosto che tramite l'indirizzo della rete. Le funzionalità base offerte al programmatore includono la lista dei vicini, informazioni riguardanti gli spazi di tuple e i nodi, oltre che lo spostamento degli agenti; esso fornisce un gestore agenti e implementa la gestione della memoria oltre che una macchina virtuale. La migrazione degli agenti viene offerta automaticamente dalla maggior parte dei framework ma, a causa della non affidabilità delle connessioni radio, c'è sempre una possibilità che il trasferimento fallisca: lo sforzo di assicurare una riallocazione sicura degli agenti rappresenta una fattore di caratterizzazione fra i diversi approcci.
I linguaggi appartenenti a questa classe hanno poco in comune con la computazione spaziale: come per i dataflow-based ci si concentra nello specificare le funzionalità dei blocchi costruttivi in relazione agli eventi/dati disponibili piuttosto che allo spazio e al tempo.

3.5 Pervasive computing

Il pervasive computing rappresenta lo scenario in cui le persone, immerse nel loro ambiente naturale, sono in grado di interagire automaticamente con sensori e attuatori sparsi ovunque in modo da consumare informazioni di interesse basate sulle loro preferenze e sulla loro posizione reagendo allo stesso tempo con la produzione di ulteriori informazioni per altri soggetti. A causa della complessità di tali sistemi,

vengono spesso adottate metafore ispirate dalla natura per il design e l'implementazione di tali sistemi. La rete computazionale sulla quale le applicazioni tipicamente girano assomiglia molto ad una WSN: essa contiene nodi mobili (smartphone, sensori, etc.) e si appoggia pesantemente alle tecnologie wireless in quanto i dispositivi sparsi per la rete sono difficilmente collegabili via cavo. A differenza dei wireless sensor network, il pervasive computing:

- sembra poter gestire un'ampio insieme di scenari di rete compresa la comunicazione globale;
- è meno vincolata dalle limitazioni energetiche o dal potere computazionale;
- è più aperta a gestire un insieme di dispositivi e contenuti differenti.

Le interazioni fra i dispositivi solitamente richiedono sia tecniche di auto-organizzazione per far emergere proprietà globali, sia mezzi espressivi per elaborare informazioni.

Ci sono realmente pochi esempi di DSL pervasivi che offrono operatori spaziali: la maggior parte di questi linguaggi (come Linda) si appoggia a strutture basate su spazi di tuple per astrarre completamente la rete dal programmatore. Tutti i DSL pervasivi utilizzano un programma immobile ed uniforme mirato per dispositivi discreti, nonostante ciò il dispositivo può essere mosso da agenti esterni come utenti che lo utilizzano e lo portano con sè.

3.5.1 Tuple-based

Il principale esempio di un framework di programmazione per il pervasing computing che incorpori concetti spaziali è TOTA (Tuples On The Air): un middleware per la condivisione di dati tramite rete, stabilito su spazi di tuple e in grado di supportare la coordinazione basata sui campi. Ogni nodo ospita uno spazio di tuple, dal quale le tuple possono diffondere in rete tramite i propri vicini creando campi spaziali: TOTA per la comunicazione utilizza un meccanismo publish-subscribe per una comunicazione multicast globale con un range specifico di vicinanza. Ogni tupla presente in un nodo racchiude;

- un contenuto (i dati);
- una regola di diffusione (la politica per cui una tupla deve essere clonata o diffusa);

- una regola di mantenimento (la politica per cui le tuple si evolvono in base a eventi o tempo trascorso).

Questi comportamenti vengono descritti in Java rendendo così TOTA un DSL di estensione. L'unico operatore spaziale supportato direttamente in TOTA è il concetto di vicinanza, mentre altre nozioni come la posizione assoluta/relativa possono essere utilizzate ma devono essere programmate ad un livello superiore posizionato sopra TOTA: l'utilizzo di questi strumenti consente meccanismi spaziali più avanzati che altrimenti non sarebbero supportati nativamente. Alla mancanza di primitive per la misurazione o la gestione dello spazio, si può ovviare ad esempio attraverso la diffusione di una tupla riguardante il gradiente, la quale mantiene la distanza stimata dalla tupla sorgente che l'ha originata tramite un semplice meccanismo di conteggio dei salti.Un approccio alternativo è quello di codificare esplicitamente all'interno delle tuple il loro ambito spaziale utilizzando coordinate relative o globali, e quindi usare la localizzazione dei dispositivi per determinare la loro distribuzione.

Geo-Linda è un esempio di questi linguaggi, esso combina la gestione delle tuple di Linda con i concetti di indirizzamento geometrico di SPREAD: le tuple vengono lette e pubblicate su un assortimento di primitive geometriche, tutte definite in relazione ad un dispositivo. Il linguaggio introduce inoltre ulteriori primitive che consentono di determinare grossolanamente il movimento dei dispositivi analizzando la comparsa e scomparsa delle tuple. Tutti questi linguaggi utilizzano un programma immobile destinato a dispositivi discreti, i quali però possono essere spostati da agenti esterni come gli esseri umani che li trasportano e li usano.

Un'altra dimostrazione di questa classe può essere individuata in Zone-of-Influence, il cui modello è stato introdotto per descrivere se un dispositivo pervasivo rientra o meno nella zona di influenza dell'attività di un altro a seconda della loro posizione relativa nello spazio. Tecnicamente una ZoI rappresenta una regione spaziale ma possono essere usate astrazioni spaziali come:

- davanti/dietro;
- vicino/media distanza/lontano;
- stessa direzione/direzione opposta;
- nozioni simil-insiemistiche legate alle aree.

3.5.2 Modelli ispirati a reazioni chimiche

Questi modelli traggono ispirazione dalle reazioni chimiche per determinare come i dati si diffondono attraverso la rete. Seguendo l'idea originaria di TOTA, sono stati svolti lavori allo scopo di creare DSL per la coordinazione di applicazioni di computazione pervasiva dedicati alla gestione dell'interazione fra agenti anzichè al loro comportamento. In questi linguaggi di reazioni semantiche ispirate alla biochimica, ogni reazione detta il modo in cui la popolazione delle tuple, sparse per la rete, può evolversi e diffondersi: l'evoluzione è espressa in modo da simulare esattamente la chimica, in cui un valore di concentrazione viene mantenuto in ciascuna tupla ed è aggiornato usando modelli chimici stocastici. In questi casi, la cosa più importante è il modo in cui viene realizzata la diffusione: le reazioni producono delle cosiddette firing tuples, che le pianificano per il trasferimento ad un dispositivo vicino, la destinazione è stabilita in modo probabilistico prendendo in considerazione un tasso di trasferimento che caratterizza ogni canale di interazione nodo-nodo. In questo modo si astrae dai meccanismi di matching semantico, che rimane comunque un ingrediente essenziale dei DSL per il pervasive computing.
In alternativa gli aspetti spaziali possono essere gestiti come una qualsiasi altra informazione semantica: la presenza, la posizione relativa, la distanza e l'orientamento sono trattati come una tupla riferita ad un oggetto e il suo spostamento viene realizzato modificando una sua proprietà specifica. In questi modelli, a livello comunicativo, c'è anche la possibilità di indirizzare messaggi individuali e mirati (unicast) al contrario di quanto detto in precedenza per TOTA.

3.6 Robotica

I sistemi multi-robotici tendono a essere spaziali per motivi legati alla località della loro comunicazione e alle interazioni fisiche dei robot. Nella robotica degli sciami, le entità si presentano tipicamente senza contatto fisico e possono essere ampiamente diffuse nello spazio: gli obiettivi sono tipicamente specificati in termini di percezione e attuazione delle interazioni con l'ambiente esterno, come mappare oppure ricercare e salvare. Nella robotica modulare invece, i soggetti sono tipicamente in contatto e funzionano insieme allo scopo di creare forme fisiche desiderate.

3.6.1 Robotica di gruppo

La robotica di gruppo enfatizza i sistemi composti da numerosi robot con un vasto numero di agenti, individualmente molto semplici, e le loro interazioni locali. Fra gli impegni principali vi sono operazioni robuste e scalabili impiegate in applicazioni come il controllo dell'ambiente, la ricerca e il salvataggio tramite veicoli aerei: tutte attività che traggono vantaggio dall'abilità di coprire uno spazio molto ampio in parallelo.

Algoritmi efficienti per questi problemi richiedono solitamente che i soggetti si localizzino sia globalmente che localmente rispetto agli altri: queste capacità fisiche si prestano inoltre a misurare direttamente e a manipolare lo spazio-tempo così come effettuato in linguaggi quali Proto. Esso garantisce la possibilità di determinare un sistema di coordinate (preferibilmente in maniera indiretta, ad esempio derivandolo dal cambio dei vicini) oltre che quella di comunicare sia localmente che globalmente allo scopo di scambiare informazioni di stato.

Un'altra classe è rappresentata da quei linguaggi i cui individui membri del sistema non hanno accesso a localizzazioni relative e/o assolute, ne deriva l'obbligo a imitare le caratteristiche sociali degli insetti le cui capacità di localizzazione si riducono allo sfruttamento di campi di forze (feromoni) o di altri fattori ambientali piuttosto grezzi. Il comportamento di questi elementi può essere solitamente descritto attraverso una FSM con transizioni sia deterministiche che probabilistiche: in questo caso lo spazio viene spesso astratto assumendo una distribuzione spaziale media dello sciame.

3.6.2 Robotica modulare

La robotica modulare rappresenta quel campo in cui i robot sono riconfigurabili in quanto costruiti appunto da moduli che hanno la possibilità di attaccarsi/staccarsi autonomamente fra di loro in modo da riarrangiarsi in forme differenti. La promessa di questi sistemi è quella di aumentare notevolmente la versatilità, grazie alla loro abilità di riconfigurarsi in robot con funzioni differenti, e la robustezza grazie alla loro potenzialità di autoripararsi: essi propongono sfide importanti nei confronti dei DSL per la programmazione global-to-local considerando la loro composizione di decine/centinaia di attuatori e sensori distribuiti.

Un DSL appartenente a questa classe è Meld, un linguaggio di programmazione logica dichiarativo il cui obiettivo è quello di semplificare la programmazione di robot modulari tenendo in grande considerazione l'espressività e la dimensione del codice risultante. Meld è inoltre capace di esprimere algoritmi di computazione spaziale come la diffusione a gradiente, il percorso più breve e la localizzazione, tutte primitive situate al livello dell'abstract device e molto importanti per esprimere cambiamenti morfologici. Non fa parte invece delle suepossibilità la presenza di primitive che consentano di risolvere problemi di computazione global- to-local per la generazione di pattern arbitrari. I problemi di generazione di movimento possono essere invece espressi tramite una manipolazione dello spazio-tempo basata su pattern computazionali: molto spesso questo è raggiunto tramite controllo centralizzato ma un buon numero di linguaggi spaziali sono comunque emersi in questo campo. Anche in questo caso, come visto in precedenza, si può utilizzare Proto ma adottando attuazioni differenti dato che i dispositivi rimangono fissi rispetto agli altri. Il programmatore manipola la forma con operazioni come l'aggiustamento della curvatura, la scala e l'aggiustamento della densità: questa rappresentazione continua è più indiretta ma astrae la scelta di una piattaforma specifica.

3.7 Computazione parallela e riconfigurabile

Nella computazione parallela e riconfigurabile, l'obiettivo primario è una rapida esecuzione di calcoli complessi: nel primo caso ci si affida ad architetture composte da processori multipli mentre nel secondo si considerano elementi di processo configurabili. In entrambi i casi comunque, l'estrema velocità alla quale le computazioni devono essere eseguite, indica i ritardi dovuti alle comunicazioni fra i dispositivi come un interesse dominante: i device vengono disposti in uno spazio 2D o 3D molto ristretto e di conseguenza il costo comunicativo deriva tipicamente dalla distanza fra i dispositivi stessi. Questi sistemi hanno così abbracciato a lungo il concetto di spazialità nonostante la computazione che deve essere eseguita spesso non presenti una struttura con un mapping diretto verso queste architetture; di conseguenza i linguaggi di programmazione in questione utilizzano la spazialità a livelli più bassi rispetto alle architetture sulle quali si concentrano. In questa categoria rientrano i linguaggi topologici, dove la località è esplicita ma deve astrarre dalle dimensioni dell'hardware su cui la computazione avrà luogo: essa viene vista come uno scambio di informazioni

fra un insieme di processi.

Un classico esempio di questi DSL è rappresentato da MPI, un'estensione di libreria al C e al Fortran in cui i processi devono interagire attraverso scambio di messaggi tramite memoria condivisa. Tutti questi linguaggi includono dichiarazioni esplicite di località, come luoghi e regioni, che limitano l'interazione e possono essere combinati gerarchicamente in luoghi di aggregazione. Altra classe considerabile all'interno di questo macro-insieme è rappresentata dai linguaggi basati sul concetto di campo: essi eseguono una connessione esplicita fra le strutture hardware che occupano lo spazio e la computazione utilizzando array multi-dimensionali, risultando così esplicitamente spaziali nonostante supportino solo un range molto ristretto di operazioni su griglie rettilinee.

Nessuna delle tipologie di linguaggio presentate in questa sezione supporta la misura e la manipolazione dello spazio evidenziando che l'hardware è essenzialmente statico rispetto ai programmi. Il programmatore si ritrova quindi in un ambiente idealmente rettilineo in cui è compito esclusivo del compilatore e del servizio di gestione del sistema realizzare la connessione fra computazione e hardware: col passare del tempo però, la continua evoluzione dell'hardware disponibile porterà questioni di densità di potenza e di performance variabili.

3.8 Calcolo formale concorrente e distribuito

Il calcolo formale è un linguaggio primitivo usato per descrivere in maniera astratta alcune caratteristiche e comportamenti di un sistema di interesse, allo scopo di ragionare circa le sue proprietà e le sue possibili implementazioni. Alcuni esempi includono le algebre di processo usate per modellare il sistema distribuito di comunicazione dei processi e i modelli di computazione a membrana usati per ragionare circa i sistemi ispirati alla chimica.

Molti dei linguaggi appartenenti a questa classe e fortemente collegati alla computazione spaziale, sono estensioni dell'algebra di processo π-calculus, la quale astrae intenzionalmente dai concetti topologici della rete computativa modellando l'intero sistema come una composizione piatta di processi che interagiscono tramite canali. 3π, ad esempio, è stato sviluppato proprio come un'estensione del π-calculus con l'idea di modellare lo spazio dove i processi eseguono operazioni in 3 dimensioni: ogni processo ha una posizione ed un orientamento spaziale, entrambi codificati

nei cosiddetti dati spaziali. Oltre che accedervi, un processo è in grado di mandare o ricevere dati geometrici attraverso dei canali ed evolversi in un altro processo ovunque esso stia rappresentando una forma di movimento: in questo modo i dati geometrici vengono manipolati in maniera astratta, ovvero solo da operazioni di spostamento del frame di riferimento (traslazione, rotazione, scala). Nonostante sia presente un'unico sistema di coordinate spaziali, i processi non conoscono la loro posizione rispetto ad esso ma possono solamente compararla in relazione a quella degli altri soggetti del sistema, i dati vengono quindi manipolati in maniera astratta, citati solo dalle operazioni di spostamento di contesto.

Un'altra considerazione importante è la mancata presenza della nozione di tempo: i processi infatti eseguono semplicemente in ogni nodo e si sincronizzano tramite lo scambio dei messaggi (sia l'invio che la ricezione sono rappresentate tramite operazioni bloccanti). Questa comunicazione può essere attuata solo nel caso in cui il mittente conosca l'identificativo univoco del destinatario e indipendentemente da qualsiasi nozione posizionale, ogni nozione geometrica riguardante la comunicazione locale deve quindi essere codificata al di sopra di questo modello.

Un'approccio originale è quello intrapreso in Ambient calculus e nei suoi derivati: i processi eseguono in un sistema spaziale di compartimenti gerarchicamente innestati. Ogni processo è situato in un compartimento e può eseguire un numero di operazioni informate spazialmente come:

- distruggere la membrana del compartimento al quale appartengono;
- muoversi al di fuori del compartimento corrente;
- entrare in un compartimento nidificato;
- creare un nuovo compartimento.

La comunicazione non è una primitiva offerta dal linguaggio, quindi deve essere implementata attraverso interazioni come la diffusione di processi messaggeri all'interno e all'esterno dei compartimenti.

3.9 Analisi finale

Dopo quanto visto in precedenza, possiamo dividere i DSL in 4 macro-categorie:

- **Linguaggi che astraggono dal dispositivo:** spesso non sono particolarmente spaziali ma tentano di semplificare la programmazione aggregata eliminando possibili complicazioni dalla prospettiva del programmatore: questi linguaggi permettono forti astrazioni a livello di comunicazione locale per supportare la costruzione di algoritmi distribuiti. Solitamente essi offrono una grande capacità di controllo su come un programma viene implementato localmente ma poco o nulla a livello di operazioni spaziali sugli aggregati, problema che viene lasciato risolvere direttamente al programmatore.

- **Linguaggi di pattern:** si trovano ad un livello di astrazione superiore e si concentrano sulla costruzione di pattern distribuiti sullo spazio. In particolare i linguaggi topologici specificano i pattern in termini di relazioni di connettività e gerarchiche fra gli elementi, mentre quelli geometrici utilizzano forme matematiche nello spazio.

- **Linguaggi di movimento delle informazioni:** si focalizzano sul raccogliere informazioni sintetizzate in regiorni differenti dello spazio-tempo e nel consegnarle in altre regioni. Il programmatore solitamente specifica che informazioni vuole raccogliere e dove necessita che esse siano spostate, ma non come si debbano muovere a tal fine.

- **Linguaggi spaziali general purpose:** sono comunque linguaggi domain-specific, in quanto specializzati per elaboratori spazio-temporali, ma sono generici nel senso che la loro applicabilità si estende a diversi domini. Essi combinano le forti astrazioni spaziali della formazione di pattern con l'abilità di controllare l'implementazione delle dinamiche dei linguaggi astratti. Consentono inoltre una vista spaziale aggregata che permette di astrarre dai dispositivi individuali.

Capitolo 4

SCENARI APPLICATIVI

Dopo aver apprezzato le capacità hardware e computazionali in possesso di un ipotetico dispositivo ideale presente sul mercato, e aver analizzato ciò che ci viene offerto dai linguaggi relativi allo spatial computing, proviamo ad individuare i possibili scenari applicativi e le relative esigenze presentateci dalla vita quotidiana rimanendo comunque consapevoli del fatto che vedremo solo una parte di un insieme non numerabile.

4.1 Scenari SAPERE

All'interno della nostra università è presente un gruppo di ricerca che si occupa di un importante progetto a livello europeo riguardante il futuro del pervase computing, il cui nome SAPERE deriva direttamente da Self-Aware Pervasive Service Ecosystems. Nell'ambito di questo progetto sono già stati presi in esame alcuni casi di studio che ovviamente hanno a che fare direttamente o indirettamente anche con le problematiche dello spatial computing, quindi cominciamo subito con l'analizzare gli scenari in questione.

4.1.1 Visita di un museo

Prendiamo in considerazione un grande museo e una varietà di turisti che si muovono al suo interno, assumendo che ciascuno di essi sia dotato di un dispositivo provvisto di funzionalità wireless e con un qualche software a livello utente. Sarà realistico presumere che all'interno del museo siano presenti una serie di dispositivi di altro

genere, che chiameremo nodi, facenti parte di una rete distribuita associata a stanze, corridoi, opere d'arte, sistemi di allarmi o sistemi di aria condizonata. Questi nodi, oltre ad offrire connettività wireless ai turisti, possono essere sfruttati sì allo scopo di monitorare e controllare la zona, ma anche per fornire ai visitatori informazioni che li aiutino a raggiungere i loro obiettivi: ad esempio orientarsi all'interno del museo, trovare specifiche opere d'arte, coordinare le loro attività con quelle di altri gruppi di turisti in modo da evitare affollamenti oppure facendo in modo di riunirsi in zone adeguate. I primi esempi possono essere considerati banali ma la reale utilità si affianca ad una maggiore complessità proprio nel momento in cui entrano in gioco diverse entità a cui i suggerimenti devono essere forniti compatibilmente con le necessità altrui e quindi si rende necessaria la coordinazione.

A prescindere dai diversi utilizzi che possono essere considerati in situazioni del genere, rimangono stabili i seguenti requisiti del sistema:

- **adaptivity:** la disposizione delle opere d'arte può variare col passare del tempo, di conseguenza può mutare anche la topologia della rete con dinamiche e per motivi differenti, i quali devono essere affrontati a livello applicativo senza intervento umano;

- **context awareness:** ai turisti devono essere fornite informazioni contestuali senza che essi si basino su nozioni pregresse che potrebbero nel frattempo risultare non più attuali;

- **semplicità:** sia i dispositivi facenti parte della rete distribuita, sia quelli in possesso degli utenti, devono essere considerati come entità con limitate capacità computazionali e breve durata della batteria, oltre che risorse comunicative ridotte.

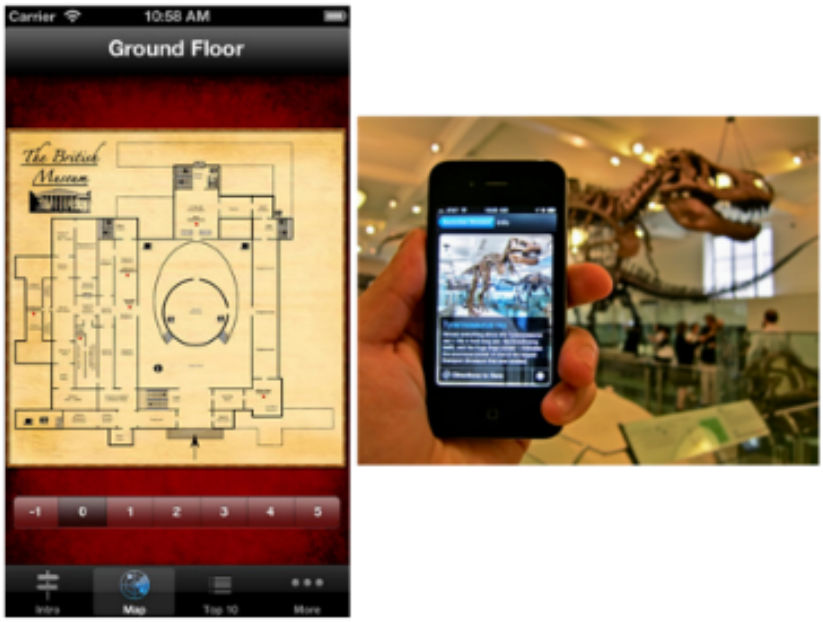

Figura 4.1: app del British Museum e del Museo di storia naturale americana

4.1.2 Guida della folla

In questo caso il nostro ambiente di riferimento è uno spazio continuo bidimensionale composto da diverse stanze connesse da corridoi stretti, il tutto ovviamente coperto da una densa griglia di dispositivi computazionali che corrispondono a quelli che nell'esempio precedente abbiamo chiamato nodi. Ogni nodo ha il compito di:

- organizzare attività di coordinazione interagendo coi nodi vicini;
- realizzare una stima del numero di persone presenti in quella zona.

Gli utenti del sistema hanno come obiettivo il raggiungimento di un punto di interesse posizionato all'interno dello spazio in questione, percorrendo il tragitto più veloce e sfruttando informazioni a ciò pertinenti provenienti dai propri device mobili oppure dai display pubblici dislocati nell'ambiente. Si parte ovviamente dal presupposto che il percorso più veloce non corrisponde al percorso più breve in quanto se tutti seguissero quel percorso, i corridoi sarebbero sovraffollati con conseguenti perdite di tempo. Il desiderio è quindi quello di ottenere un sistema che, in base a interazioni locali, sia capace di evitare affollamenti adattandosi dinamicamente ad ogni situazione emergente e imprevista. In questo caso non ci interessano algoritmi di previsione

ma ci limitiamo a considerare la diffusione di notizie sul numero di persone nelle diverse stanze in modo da rendere tali posti meno invitanti nel raggiungere il punto di interesse. L'idea di partenza per risolvere il problema è quella di generare un gradiente computazionale a partire dal punto di interesse mantenendo così in ogni nodo informazioni sulla distanza dalla sorgente, si deve poi aggiungere un fattore legato al livello di affollamento in modo da aumentare la distanza stimata nei punti in cui si sta creando un afflusso considerevole deviando gli utenti verso percorsi più lunghi ma meno affollati.

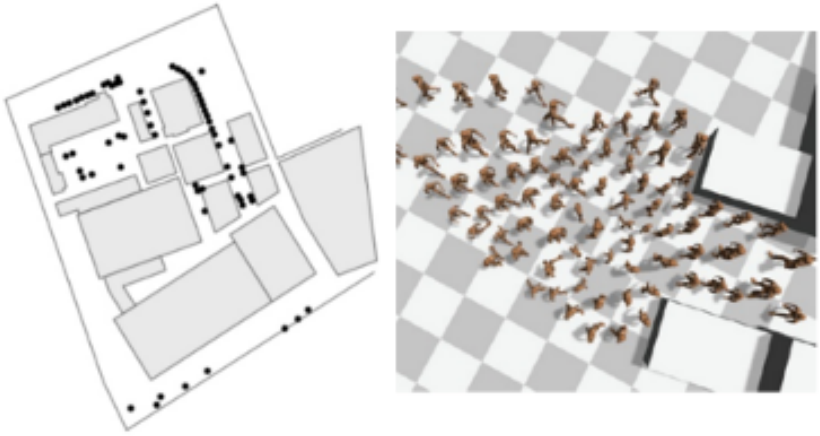

Figura 4.2: guida della folla vista dall'alto

4.1.3 Movimenti in aeroporto

Riprendiamo l'idea dei display pubblici dislocati nell'ambiente e di quelli privati dei nostri smartphone, entrambi utilizzabili allo scopo di fornire informazioni specifiche ai clienti, e consideriamo come ambiente quello di un aeroporto. Il contenuto informativo di questi display è attualmente statico e uguale per tutti i passeggeri che si aggirano per l'aeroporto, oltre che spesso ridondante in quanto ripetuto su diversi display: il nostro obiettivo è quello di creare una infrastruttura di servizio realmente generica, aperta e adattabile alle diverse situazioni che si possono presentare. Abbiamo la necessità che le informazioni siano mostrate in base allo stato corrente dell'ambiente circostante sia in termini fisici che sociali: ad esempio in base alla temperatura rilevata dai sensori circostanti e ai profili dei passeggeri, una pubblicità di un gelato potrebbe essere più indicata rispetto a quella di un liquore.

Anche in questo caso, fra i requisiti fondamentali del sistema non possono mancare:

- **adaptivity:** i servizi e le infrastrutture pervasive devono auto-adattarsi e gestire autonomamente le loro proprietà, in modo da sopravvivere alle eventualità senza alcun intervento umano e a costi limitati;

- **situatedness:** avendo a che fare con le attività spaziali e sociali degli utenti, esso deve essere in grado di interagire col mondo circostante reagendo in maniera automatica ai cambiamenti. Per esempio l'emergere di informazioni nuove e più importanti dovrebbe far sì che esse si distribuiscano in modo totalmente spontaneo attraverso tutti i display disponibili in modo da sostituire gradualmente quelle precedenti;

- **diversità:** l'infrastruttura deve essere in grado di tollerare modelli aperti di produzione di servizi e garantire il loro utilizzo senza limitare il numero e le classi di facilities potenzialmente offerte; inoltre dovrebbe poter approfittare dell'innesto di nuovi servizisfruttandoli per migliorare ed integrare quelli già presenti laddove non ci siano problemi di sicurezza a impedirlo.

Infine, l'infrastruttura che regola i display non deve solo essere intrinsecamente aperta a tutti i tipi di servizi di visualizzazione che possono essere aggiunti al sistema, ma deve anche consentire agli utenti di caricare informazioni in modo da arricchire l'offerta informativa o di adattarla alle loro esigenze. Per esempio un passeggero potrebbe voler visualizzare un contenuto presente sul suo dispositivo privato in uno schermo più grande oppure condividerlo con qualcuno che gli sta vicino.

Quest'ultimo particolare apre uno spiraglio verso sistemi in cui i dispositivi degli utenti si comportano come prosumers ovvero non rappresentano un semplice client consumatore delle risorse provenienti dai nodi server, ma costituiscono una parte attiva del sistema nel quale contribuiscono producendo non solo informazioni riguardanti la loro posizione e il loro stato. Un'altro possibile scenario inerente all'ambiente aeroportuale ma non direttamente ai display pubblici, può passare attraverso i dispositivi personali dei passeggeri: essi, dopo aver effettuato l'accesso al sistema e aver passato la loro carta di imbarco in un apposito sistema, potranno ricevere avvisi e consigli a seconda della loro posizione e delle procedura effettuate o ancora da effettuare. Consideriamo ad esempio l'estrema comodità di ricevere notifiche che ci

avvisano sullo scadere del tempo a disposizione per intraprendere una determinata procedura calcolandolo in base alla nostra prosizione, al numero di persone stimate in coda e all'età dell'utente; il tragitto da percorrere potrebbe addirittura essere illuminato utilizzando metodi di proiezione specifici con colori differenti a seconda dell'utente che lo deve compiere.

Figura 4.3: possibile funzionamento del sistema desiderato

4.2 Altri scenari

Dopo aver visto scenari in qualche modo già presi in considerazione dal progetto SAPERE, andiamo ora ad esplorare ulteriori ambiti che permettano di sfruttare nella vita quotidiana le possibilità offerte dallo spatial computing.

4.2.1 Rilevazione ore lavorative

Nella maggior parte dei luoghi di lavoro vengono utilizzati i cosiddetti badge per rilevare i momenti di inizio e fine della giornata di lavoro di un qualsiasi dipendente o salariato in generale: spesso si passa un semplice cartellino magnetico all'interno di apposite strutture di lettura non troppo avanzate, in altri casi si utilizzano invece lettori RFID in grado di evitare problematiche legate alla contraffazione delle tessere stesse eliminando anche la necessità di contatto col lettore. In ogni caso però, questo genere di rilevazione non è in grado di evitare i tanti casi venuti alla ribalta ultima-

mente grazie a trasmissioni televisive di inchiesta, le quali mostrano episodi in cui i badge vengono passati da lavoratori terzi oppure sono gli stessi lavoratori ad uscire dal posto di lavoro subito dopo aver segnalato il loro ingresso.

Supponendo di identificare ogni dipendente con un dispositivo personale e univoco, prendiamo ad esempio il proprio smartphone su cui deve essere reperibile nei confronti dell'azienda per cui lavora, le furbizie di cui parlavamo poc'anzi possono essere eliminate semplicemente inserendo un sistema di controllo di posizione all'interno del posto di lavoro. Problemi risolvibili:

- il lavoratore non può consegnare il suo smartphone ad un collega perchè deve essere rintracciabile attraverso quel dispositivo;
- nel caso in cui decidesse di "rischiare" lasciandolo in ufficio, lo storico delle posizioni mostrerebbe che il device è rimasto sul posto di lavoro dal giorno precedente;
- con un semplice sistema di analisi dello storico delle posizioni si possono analizzare dati comportamentali dei vari dipendenti allo scopo di individuare mali costumi da eliminare.

Figura 4.4: Microsoft e Blackberry security badge

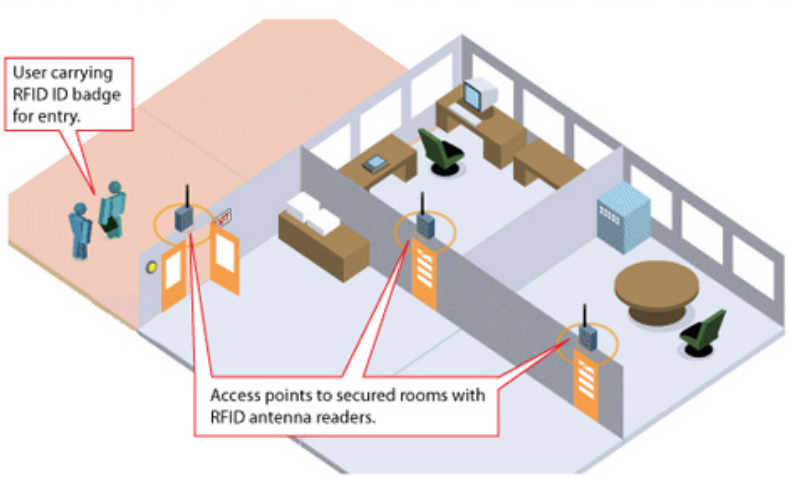

Figura 4.5: Badge RFID e sistemi di sicurezza

4.2.2 Shopping e acquisti fisici

Dall'ultimo punto dello scenario precedente deriva ciò di cui parleremo ora: nonostante la crescita esponenziale dei negozi e degli utenti che compiono acquisti online, in molti ambiti continua ad essere quanto meno essenziale effettuare compere recandosi in negozi fisici. I venditori sono spesso costretti a indire pesanti indagini di mercato attraverso lunghi e noiosi questionari allo scopo di carpire le abitudini dei consumatori mentre tutto ciò potrebbe essere evitato molto semplicemente. Il metodo meno invasivo prevede il semplice controllo dello storico delle posizioni dell'utente all'interno del negozio senza identificare nè la sua identità nè direttamente i prodotti acquistati: in questo modo a seconda della disposizione dei prodotti conosciuta a priori dal venditore avremo dati abbastanza generici ma già molto significativi semplicemente seguendo un cliente attraverso il suo indirizzo IP o MAC address. Se volessimo effettuare invece indagini più approfondite bisognerebbe permettere la registrazione dell'utente in modo da carpire alcune informazioni personali come il sesso, l'età e la categoria sociale alla quale appartiene: tali informazioni, combinate con quelle relative alle posizioni e con eventuali altre provenienti da un meccanismo associato all'emissione dello scontrino, permetterebbero di ottenere risultati inarrivabili attraverso una qualsiasi altra tecnica classica.
Come visto all'interno del primo capitolo, in determinati casi l'utente potrebbe non essere solamente un produttore di informazioni, ma anche ricevere notifiche sul pro-

prio smartphone riguardanti offerte e prodotti che potrebbero interessarlo in base alla zona in cui si trova o ad acquisti effettuati in precedenza.

Figura 4.6: prototipo di una possibile app

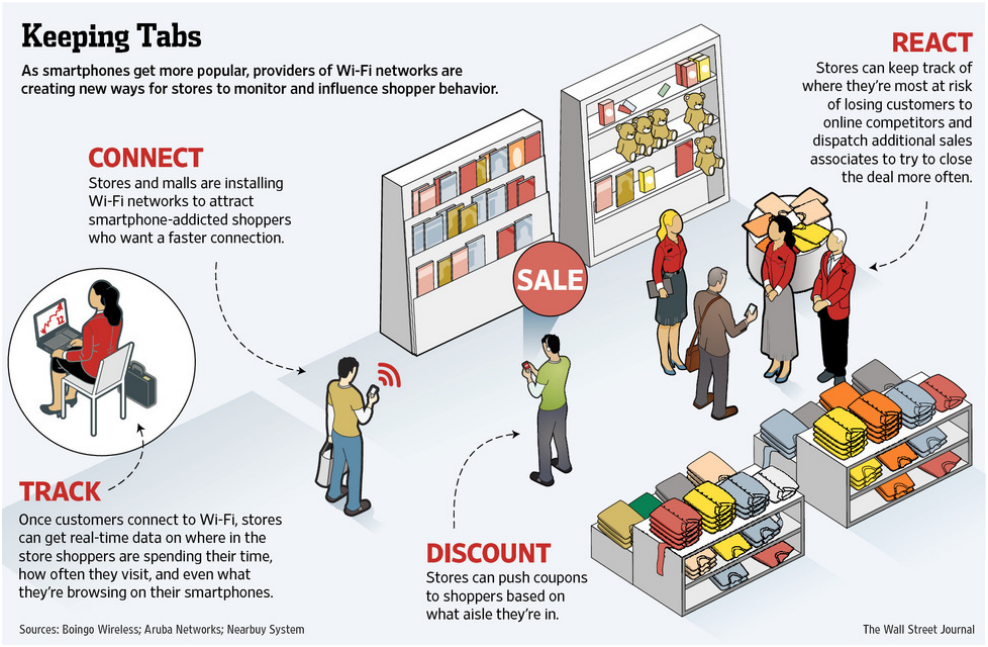

Figura 4.7: possibile utilizzo di queste tecnologie

4.2.3 Mappe e navigazione

Infine, ultimo ma sicuramente non per importanza, andiamo a considerare i potenziali sviluppi per quella che è stata probabilmente la prima funzionalità per cui si sono utilizzate le posizioni e la mobilità: ovvero la creazione di mappe e il loro impiego nelle istruzioni di navigazione stradale. Fino a qualche tempo fa i limiti del GPS all'interno di strutture chiuse o anche lievemente schermate avevano limitato il mapping ad ambienti esterni, come abbiamo visto in precedenza però, sono ora presenti tecnologie alternative che hanno aperto nuove possibilità e hanno dato il via all'indoor mapping, di cui abbiamo visto alcune applicazioni negli scenari precedenti.In questo senso, dopo aver mappato la zona, certi dati possono essere utilizzati nelle maniere più disparate in base al tipo di struttura presente e al suo utilizzo; al contrario l'outdoor mapping vede ormai ridursi i possibili sviluppi in quanto si tratta in ogni caso di strade con presenza di strutture di vario genere.

Dopo la rivoluzionaria idea di Google Street View, che ha permesso una ricostruzione visiva e realistica di gran parte delle strade mondiali, rimane ora la possibilità di migliorare le informazioni sul traffico attraverso una forma indiretta di crowdsourcing. Al momento infatti le informazioni riguardanti il traffico vengono per lo più ottenute grazie a strumenti ottici oppure tramite segnalazioni dirette degli utenti (vedi Waze), mentre in questo senso rimangono ancora abbastanza inutilizzati i dati rilevabili dagli smartphone. Attraverso algoritmi non troppo complicati, si possono infatti considerare le posizioni e quindi la velocità di quei dispositivi che a partire da un certo momento si sono spostati con velocità simili accostabili a quelle di una macchina: in questo modo avremo a disposizione la velocità media e una stima della densità dei veicoli per quasi tutte le strade percorribili. Tale meccanismo andrebbe comunque integrato con segnalazioni spontanee di incidenti o avvisi dovuti a situazioni particolari ma un'informatizzazione così globale comporterebbe senza dubbio un vantaggio molto considerevole.

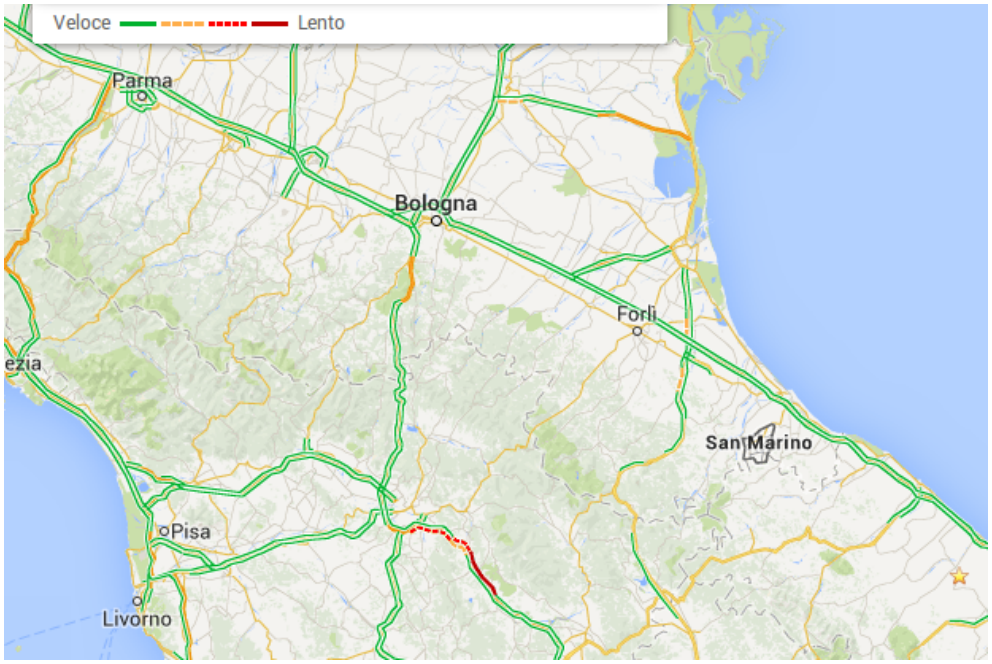

Figura 4.8: interfaccia di Google Maps che mostra il traffico in tempo reale

Capitolo 5

DEFINIZIONE MODELLO ARCHITETTURALE ASTRATTO

Dopo aver effettuato una panoramica preliminare riguardante le piattaforme fisiche esistenti, aver introdotto la struttura generale di un sistema di programmazione aggregata e aver riassunto le principali caratteristiche dei linguaggi di computazione spaziale, è giunto il momento di collegare questi elementi tramite la definizione dell'architettura astratta di un ipotetico middleware di riferimento.

5.1 Identificazione eventi indispensabili

Riprendiamo quanto descritto nella tabella di fine capitolo 2.5 (in cui venivano codificati i principali eventi e definite le loro caratteristiche) andando quindi ad effettuare un'analisi riguardante la loro origine, con questo termine intendiamo riferirci alla generabilità delle informazioni relative a tali eventi distinguendole fra:

- **elementi primitivi**, la caratteristica dell'evento e il valore ad esso associato non sono recuperabili in maniera differente, questo evidenzia la loro necessità e la mancata possibilità di arrivare ad essi attraverso la combinazione di altri eventi;

- **elementi derivati**, la caratteristica dell'evento o il valore ad esso associato non è presente in una sola linea della tabella e non è quindi generabile in maniera univoca, questo sta a significare che si possono ridurre il numero di entry di quella tabella selezionando il generatore con le caratteristiche che preferiamo.

Cerchiamo quindi di ridurre quanto visto in precedenza ai soli elementi primitivi semplificando così gli oggetti del nostro ragionamento ed evidenziandoli in grassetto.

- **Posizione (coordinate):** con il GPS abbiamo la possibilità di ottenere le coordinate complete (latitudine, longitudine e anche altitudine), cosa che non è possibile avere tramite Wi-Fi e connessione dati a meno di non incrociare i dati con complesse mappe altimetriche o con quelli provenienti da eventuali barometri, riteniamo insufficiente anche la sola presenza di una scheda GSM;

- Posizione (direzione e distanza): esse vengono rilevate tramite il collegamento di due posizioni successive;

- Posizione (velocità): anche per la velocità non esiste un metodo di rilevazione diretto quindi se si vuole approssimare il moto ad un movimento uniforme si dovrà dividere la distanza fra due punti (ottenuta tramite triangolazione) per il tempo trascorso fra i due rilevamenti;

- **Posizione (accelerazione):** in questo caso abbiamo uno strumento diretto per la misurazione di tale caratteristica, che altrimenti non potrebbe essere calcolata con precisione.

- Posizione (regione): se ci interessa percepire gli spostamenti di un utente in riferimento ad una regione predefinita, per ovvi motivi dobbiamo tenere conto delle coordinate posizionali ma in molti casi la regione è interna o molto ridotta tanto che i dati provenienti dal GPS non sono sufficientemente precisi e ci si affida al segnale Wi-Fi;

- Posizione (contapassi): questo valore può essere derivato da un valore di distanza semplicemente dividendo per la misura media di un passo umano;

- **Rotazione (vettore):** per quanto riguarda la rotazione del dispositivo, l'apposito sensore è l'unico modo che abbiamo a disposizione per rilevare i dati in questione;

- Rotazione (variazione velocità): stesso ragionamento può essere effettuato a proposito del giroscopio, in questo caso però non lo inseriamo fra gli elementi indispensabili in quanto ci accontentiamo delle informazioni precedenti;
- Stato (attività): questa caratteristica è chiaramente derivata da una serie di valori raccolti in diversi ambiti e combinati attraverso regole ben precise.

Non prenderemo in considerazione ulteriori caratteristiche di rilevazione presenti nella tabella in quanto il nostro obiettivo rimane al momento prettamente spaziale, stesso discorso vale per quanto riguarda gli eventi di trasformazione, che ricondurremo ad attività dell'utente. Dall'analisi condotta in questo capitolo, per il nostro scopo emerge l'indispensabilità dei seguenti elementi:

- GPS;
- una qualsiasi connessione (Wi-Fi o 3G);
- un accelerometro;
- un sensore che misura la rotazione del dispositivo in termini vettoriali.

5.2 Generazione informazioni derivate

Tramite i 4 elementi individuati nella sezione precedente e i dati da essi ricavabili, siamo in grado di generare la maggior parte delle informazioni che ci interessano per la nostra analisi e che erano state presentate nella tabella sopracitata.
Dopo una breve analisi volta solamente ad effettuare la scrematura richiesta, andiamo quindi a considerare nel dettaglio come ricostruire gli eventi non primitivi e a valutarne l'effettiva accuratezza, nel corso di questo percorso faremo riferimento alle possibilità offerte da Android dal momento che sarà il sistema operativo di riferimento nel proseguimento di questa tesi.

- Distanza fra due punti: nei moderni dispositivi mobili non esiste un componente che permette di calcolare direttamente la distanza, per questo motivo possiamo solamente ricostruire questo genere di dato. Le conoscenze in termini geografici, le formule matematiche e alcuni servizi preconfezionati offerti agli sviluppatori, consentono però di ricavare questa informazioni senza nessuna difficoltà e con un livello di accuratezza abbastanza elevato. Se si prendono

in considerazione le coodinate GPS, probabilmente avremo interesse a misurare distanze superiori alle decine di metri, e di conseguenza il margine di errore che siamo disposti a tollerare non sarà inferiore ad esempio ai 50 m. Prendiamo ora in considerazione il sistema operativo Android, come abbiamo visto al capitolo 1 esso utilizza il concetto di Location per operare con le posizioni dei dispositivi, e ci viene incontro mettendo a disposizione due semplicissimi metodi:

```
Location locationA = new Location("point A");
locationA.setLatitude(latA);
locationA.setLongitude(lngA);
Location locationB = new Location("point B");
locationB.setLatitude(latB);
locationB.setLongitude(lngB);
float distance = locationA.distanceTo(locationB);

float[] results=new float[5];
float distance = Location.distanceBetween(latA, lngA, latB,
    lngB, results);
```

Nel caso in cui fossimo interessati a distanze ad esempio inferiori ai 20 m, dovremo invece andare a considerare i dati provenienti dall'accelerometro, intraprendendo calcoli molto più complicati e soprattutto soggetti a errori e imprecisioni generabili da diversi fattori ma soprattutto dovuti alla continua variazione di tali informazioni.

- Direzione: identifichiamo il termine in questione con l'angolo che il vettore congiungente il punto di partenza e il punto di arrivo forma con uno di quelli identificati dai punti cardinali. Anche in questo caso, come in precedenza, per distanze non troppo ravvicinate il calcolo va ricondotto a forme trigonometriche legate alla forma sferica della Terra e quindi produce risultati significativi senza troppi sforzi computazionali. Anche in questo caso prendiamo in considerazione le API Android e notiamo come il tutto sia immediato; riconducendoci al primo metodo del punto precedente:

```
float bearing = locationA.bearingTo(locationB);
```

Non abbiamo però ancora specificato a cosa si riferisce il valore di ritorno di tale metodo, a tal proposito riportiamo direttamente quanto Google definisce nelle sue API:

> *Returns the approximate initial bearing in degrees East of true North when traveling along the shortest path between this location and the given location. The shortest path is defined using the WGS84 ellipsoid. Locations that are (nearly) antipodal may produce meaningless results.*

- Velocità: anche in questo caso non disponiamo di un componente hardware che misuri direttamente il valore di questo evento quindi dobbiamo ricorrere a calcoli legati a semplici formule cinematiche.
  - GPS: nel caso si voglia considerare una sorta di velocità media che approssimi a 0 le variazioni di velocità effettuate durante un percorso, ci basterà dividere la distanza fra due punti (vedi sopra) per il tempo trascorso fra i due rilevamenti.
  - Accelerometro: nel caso si intenda invece misurare velocità puntuali, dovremo ricorrere alla formula V=V 0 +at e quindi considerare a intervalli molto ridotti i valori rilevati dall'accelerometro. In questo secondo caso, come anticipato in precedenza durante la discussione sulla distanza, sono presenti diversi fattori di disturbo che possono provocare misurazione inesatte.
- Regione: se identifichiamo una regione approssimando la sua definizione a quella di una cerchio di centro e raggio definiti, per verificare la presenza all'interno della regione non dovremo fare altro che confrontare la distanza di un punto dal centro con il valore del raggio e verificare che la prima sia minore della seconda. Questo meccanismo, anche se molto semplice, viene comunque wrappato all'interno di appositi metodi che ci consentono di definire e interagire con un oggetto Geofence.
- Contapassi: la situazione è ovviamente affine a quanto visto per la distanza fra due punti, in questo caso per ottenere il numero dei passi dovremo dividere la distanza per il valore medio di un passo. Se invece preferiamo

utilizzare i dati provenienti dall'accelerometro, dovranno essere fatte diverse considerazioni in merito ai valori medi di accelerazione e alle tempistiche, una possibile implementazione è reperibile su github.com/bagilevi/android-pedometer/blob/master/src/name/bagi/levente/ pedometer/StepDetector.java. Le API Android ci vengono incontro offrendoci la possibilità di gestire specifici sensori attraverso la registrazione come listener di eventi TYPE_STEP_COUNTER e TYPE_STEP_DETECTOR. Il primo tipo di evento tiene conto dei passi effettuati dall'avvio del telefono e mantiene il timestamp del primo passo, il secondo invece viene generato ad ogni passo e restituisce il valore simbolico 1.

- Attività: come già anticipato nel capitolo 1 e precedentemente in questo capitolo, questo genere di informazioni è fortemente derivato e viene garantito attraverso un livello superiore fornito dai realizzatori dei sistemi operativi. Non sappiamo quindi come le informazioni base vengano combinate e quali siano i parametri utilizzati, ma possiamo solamente usufruire di elaborazioni altrui. Android propone queste funzionalità sotto il nome di Activity Recognition restituendo una lista di attività ordinate per "confidence" decrescente.

5.3 Elementi infrastrutturali necessari

Dopo aver evidenziato gli elementi indispensabili a livello di piattaforma fisica selezionando i vari componenti hardware, dobbiamo considerare ciò che non può mancare nel nostro modello architetturale a livello di costrutti e/o meccanismi strettamente legati alla natura dell'infrastruttura che andiamo a definire.
Vista la natura del nostro problema, ci è sembrato abbastanza lineare prendere in considerazione un middleware che fosse basato su una filosofia ad agenti, e in tal senso saranno necessari i seguenti elementi:

- meccanismi di identificazione degli agenti;
- strumenti di comunicazione;
- meccanismi di gestione dei movimenti degli agenti;
- componenti al contorno di gestione dei servizi;

- gestione agenti esterni al middleware.

Ricordiamo innanzitutto la struttura base di un sistema MAS (Multi Agent System), in cui troviamo:

- gli **agenti**, elementi fondamentali che rappresentano componenti del sistema proattivi, in grado di incapsulare l'esecuzione autonoma di attività all'interno di un ambiente;
- gli **artefatti**, componenti passivi del sistema interpretabili come risorse e mezzi appositamente costruiti per supportare le attività cooperative e competitive degli agenti;
- i **workspaces**, contenitori concettuali di agenti ed artefatti, utilizzabili per descrivere la topologia dell'ambiente ed offrire una modalità di definizione della nozione di località.

Quindi proviamo a definire schematicamente l'architettura che andremo poi a specializzare, cominciando con l'identificare 5 possibili livelli di interesse, visualizzabili nel disegno seguente.

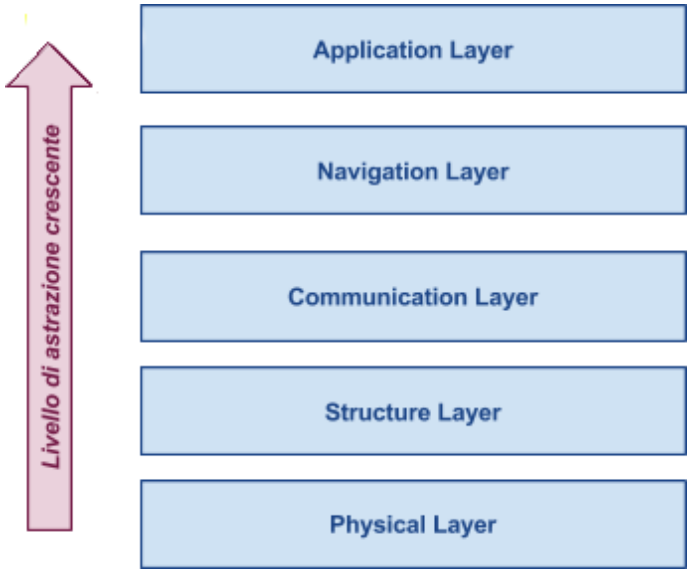

Figura 5.1: livelli del modello infrastrutturale

Analizziamo ciascun layer partendo da quelli più semplici e aumentando la complessità delle astrazioni richieste per la costruzione di ciascun livello.

- **Il livello fisico** indica tutto ciò che si rende necessario per la formazione di una rete ed è quindi relativo ai meccanismi di interazione più primitivi legati alla diffusione broadcast verso la rete di appartenenza.

- **Il livello strutturale** è quello che consente di introdurre e definire astrazioni spaziali come i workspace, mantenute poi dai vari componenti fisici presenti nel sistema. La possibilità di creare strutture spaziali stabili e capaci di sopravvivere adattandosi alle condizioni di funzionamento della rete, non viene vista come una finalità ma come un'opportunità da sfruttare per costruire attività di più alto livello: ne sono un esempio i concetti derivanti di località e vicinato, ovvero l'origine delle gerarchie orizzontali presenti nel sistema, oppure i meccanismi di localizzazione da essi derivanti e necessari per il funzionamento di queste tipologie di sistemi. In questo momento non ci riferiamo alla definizione di un ambiente specifico di ciascun caso ma lo intendiamo come l'insieme delle regole generiche di creazione riferibili al middleware e non alle singole applicazioni.

- **Il livello comunicativo** rappresenta le astrazioni e le infrastrutture necessarie alla creazione degli spazi di tuple e alla gestione dei canali di messaggistica diretta permettendo quindi di accrescere le capacità di astrazione, sfruttando le strutture messe a disposizione dai livelli precedenti.

- **Il livello di navigazione** è legato a tutto quanto verrà detto in termini di mobilità e deve quindi forzatamente poggiare su strumenti comunicativi adeguati che rendano possibile il trasferimento non solo di messaggi protocollari ma anche di veri e propri agenti: la necessità di specificare un layer separato sta nella centralità di tali attività all'interno di sistemi legati a informazioni spaziali.

- **Il livello applicativo** riguarda la presenza e i comportamenti che gli agenti assumono all'interno di un'applicazione specifica, occupandosi di tutto ciò che concerne le attività consentite dalla natura degli agenti e del sistema.

5.4 Architettura nel dettaglio

Proseguiamo la nostra analisi specificando nel dettaglio quanto anticipato per i singoli layer fino a raggiungere un'architettura astratta in grado di sopperire a tutte le nostre necessità, computazionali e non solo.
L'esigenza primaria è quella della costruzione del sistema, consideriamo quindi un componente responsabile della registrazione degli agenti e della loro identificazione: per questioni di comodità assumiamo che assegni ad ogni agente che richiederà di far parte del sistema un ID numerico crescente col quale esso verrà riconosciuto all'interno del sistema stesso. Assegnamo a tale componente, che chiameremo appunto responsabile, l'ID interno 0 e la capacità di offrire un servizio di "pagine bianche" che mantenga i riferimenti agli agenti presenti nel sistema locale facendo attenzione a non creare ambiguità con altri agenti presenti in sistemi diversi dello stesso ambiente. Al momento della creazione, ogni agente si deve obbligatoriamente registrare tramite un responsabile ad un workspace, altrimenti viene considerato orfano e non fa parte del sistema.

Nel caso siano definibili puntualmente coordinate omologabili e universali, interpretabili in maniera univoca e sempre disponibili, si può considerare l'ipotesi di identificare gli agenti tramite la loro posizione anzichè assegnare loro un identificativo numerico che nulla ha a che fare con le loro caratteristiche di nostro interesse. In questa situazione, si renderebbe necessaria una sorta di collaborazione fra il livello di navigazione e quello della struttura, in modo da garantire precisione e consistenza nel mantenere aggiornate tali aspetti in base al comportamento di ciascun agente, oltre ad un'organizzazione di naming differente.

Consideriamo inoltre la necessità di definire sistemi innestati in modo da mantenere una certa modularità e rendere ciascuno di essi più gestibile: per questo motivo ogni responsabile manterrà anche un identificativo del sistema che gestisce e sarà legato a caratteristiche posizionali/di rete (nel disegno l'indirizzo IP ma potrebbero essere utilizzate anche coordinate spaziali). Dopo aver definito ciascun sistema locale con il termine **workspace**, quella che segue è l'architettura risultante da quanto detto finora.

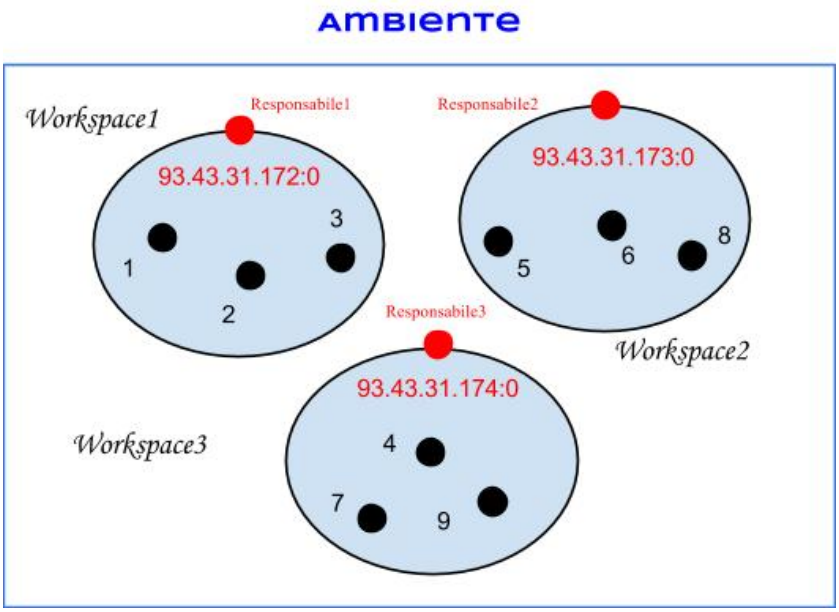

Figura 5.2: primo prototipo del modello architetturale

In questo modo abbiamo definito il livello superiore del nostro middleware, ovvero quello che racchiude solamente i soggetti principali del sistema e quindi gli agenti: essi hanno fra le loro **caratteristiche** fondamentali quelle **comunicative** e quelle legate alla **mobilità**. Si avverte quindi immediatamente il bisogno di effettuare una vera e propria scelta concettuale nell'utilizzo di meccanismi a **spazio di tuple** o di quelli più tradizionali a **messaggi**: nel nostro caso abbiamo deciso di utilizzare entrambi i metodi ma con finalità ben differenti. Se infatti preferiamo una comunicazione diretta fra gli agenti e i responsabili per meccanismi di coordinazione che coinvolgono entità superiori, lo stesso non si può dire nel caso in cui siano gli agenti dello stesso livello ad avere necessità comunicative: nel secondo caso infatti, la natura distribuita del sistema e soprattutto la collaborazione che si richiede in sistemi aggregati, rendono indispensabile la presenza di strumenti che consentano comunicazioni più globali e una coordinazione più potente, mentre al primo va ricondotta la tematica della mobilità in cui sono sufficienti comunicazioni specifiche e mirate.

Come abbiamo già anticipato, un agente non è legato al sistema da cui è stato generato, ma mobile e quindi in grado di cambiare la sua posizione: i sistemi sono quindi permeabili, ovvero cambiano in struttura e dimensione a seconda del comportamento degli agenti: essi palesano i propri movimenti cambiando workspace

attraverso la deregistrazione dal vecchio e la registrazione al nuovo mediante comunicazione ai responsabili. Nel caso in cui i workspace siano dislocati realmente in nodi della rete differenti, il trasferimento di un agente o di un componente in genere, rende necessaria la presenza nel middleware dei seguenti requisiti:

- sistema di denominazione degli agenti;
- servizio di registrazione;
- canali di comunicazione e tipologie di messaggi definiti;
- possibilità di accesso all'interno del middleware per recuperare codice, dati e stato dell'agente.

Un possibile **protocollo** da seguire per questo tipo di operazioni potrebbe essere il seguente:

1. l'agente comunica al responsabile locale la sua volontà di migrare;
2. il responsabile locale sospende l'agente e ne recupera codice, stato e dati;
3. il responsabile locale contatta il responsabile dell'altro workspace cominciando il trasferimento;
4. il responsabile remoto crea e registra il nuovo agente partendo da codice, stato e dati che gli sono stati inviati;
5. il responsabile locale elimina la sua copia dell'agente;
6. il responsabile remoto avvia la sua copia dell'agente.

In questo modo si avrebbe un completo controllo del trasferimento degli agenti e tramite messaggi strutturati ed appartenenti ad una ontologia ben definita, grazie a richieste e conferme distribuite su più step, anche la possibilità di spostare un agente senza avere una clonazione attiva (stesso agente attivo in due workspace differenti) per nessun lasso di tempo. Ovviamente un protocollo così definito, può essere molto semplicemente adattato a casi in cui si renda necessaria una clonazione attiva, basti pensare ad esempio alla necessità di modellare tanti organismi con le stesse caratteristiche, e consente attraverso la specifica di banali parametri alcune scelte a livello

di esecuzione (resettare l'agente o farlo riprendere da dove era rimasto trasferendo o meno lo stato e i dati di sua appartenenza).

Come anticipato, per ciò che riguarda invece meccanismi di comunicazione fra agenti dello stesso rango, è preferibile utilizzare una tipologia particolare di **memoria condivisa**, soprattutto se si pensa al fatto che essa sia concettualmente distribuita su più nodi: in questo modo mittente e destinatario vengono in qualche modo disaccoppiati e scaricati di responsabilità legate alla conoscenza strutturale completa del sistema. Trovandoci in ambito mobile, le risorse computazionali e di memorizzazione sono limitate: pensare quindi di dover mantenere consistente su più nodi una stessa base conoscitiva si rivela un'idea piuttosto sconveniente. A questo punto non ci resta che scegliere il nodo con maggiori possibilità e affidare a quell'host il compito di conservare la conoscenza del sistema: anche in questo caso però, possiamo ridurre la necessità di interazione con l'esterno distinguendo due tipologie di **operazioni**, ovvero quelle **locali** e quelle **remote**. Le prime richiedono risorse e sforzi molto limitati essendo volte a mantenere lo stato del sistema consistente con tutti gli agenti appartenenti allo stesso workspace dell'operante, le seconde mirano invece allo stato del sistema globale e sono quindi legate a informazioni di interesse collettivo situate sul nodo base. Questa scelta, che sembra erroneamente concentrare l'intero carico del sistema su un'unico nodo, in realtà punta a garantire una ragionevole scalabilità incoraggiando appositi design che promuovano le interazioni locali fra agenti situati a distanze limitate.

La definizione dei workspace e la conseguente suddivisione appena conclusa a proposito delle operazioni, evidenzia come si siano in qualche modo automaticamente definiti i concetti di locality e di vicinato, analizzando la stessa situazione sotto due aspetti differenti: a questo livello si tratta comunque di concetti semplicistici e generici che possono poi essere raffinati a seconda del dominio applicativo tramite astrazioni spaziali definibili in modi diversi. Come spesso accade, alle pratiche in qualche modo legate alla memoria condivisa, vengono associati metodi che consentono la reazione o l'attivazione di un comportamento specifico in seguito al verificarsi di un determinato evento: ciò deve quindi essere realizzato tramite tecniche publish/subscribe di livello più o meno alto che possono sfruttare meccanismi bloccanti in fase di lettura.

Tornando indietro alla definizione del sistema, con questo tipo di composizione si rende necessaria la presenza di componenti esterni ai workspace al solo scopo

di offrire le strutture necessarie al completamento delle operazioni remote, che nel nostro caso saranno anche legate ai servizi. Inseriamo all'interno di questo contesto un componente per ciascun generatore di eventi, prendendo in considerazione solo il risultato della selezione effettuata al capitolo precedente:

- GPS: **gpsManager**;
- una qualsiasi connessione: **connectionManager**;
- accelerometro: **accelerationManager**;
- vettore rotazione: **rotationManager**.

Tali componenti saranno responsabili della gestione degli eventi: prendiamo ad esempio un dispositivo potenzialmente rappresentabile da un agente, esso disporrà o meno di determinate caratteristiche hardware fra quelle elencate sopra, in base a ciò l'agente corrispondente comunicherà con il gestore del servizio associato ottenendo in risposta la possibilità di interagire con la piattaforma specifica oppure una negazione.

Allo stesso modo possono essere inseriti nel middleware tanti componenti, ciascuno responsabile per un servizio specifico e registrato presso un responsabile servizi globale che ha la funzione di "pagine gialle" reindirizzando gli agenti al componente di interesse. In questo modo abbiamo definito una architettura modulare sia orizzontalmente che verticalmente considerando la separazione fra workspace e la divisione a livelli di responsabilità, lasciando massima libertà alla creazione di servizi. A seconda del servizio, delle sue caratteristiche e dell'interesse generato, andrà poi deciso se posizionare il componente in questione come oggetto locale ad uno specifico workspace o se renderlo disponibile a tutto l'ambiente collocandolo nel già citato nodo base. Per componenti esterni al middleware (associamo tale situazione ad elementi al di fuori dell'ambiente considerato) sarà invece un responsabile superiore ad offrire un canale di comunicazione o servizi specifici di interazione con gli elementi non appartenenti al middleware.

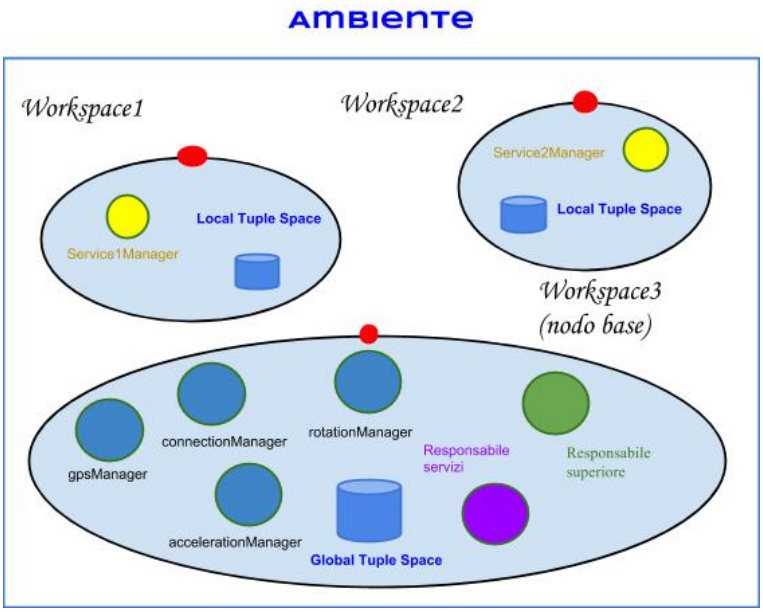

Figura 5.3: modello architetturale finale

5.5 Considerazioni in ottica programmazione aggregata

Vediamo ora di contestualizzare il modello architetturale in base alle considerazioni proposte all'interno del capitolo 2 riguardanti alcune proprietà relative alla programmazione aggregata. Nel processo di trasformazione da concetti individuali a concetti aggregati, erano stati individuati 5 livelli all'interno di una struttura piramidale, ci resta quindi da capire a che punto si posizioni quanto abbiamo definito in questo capitolo.
La scelta si riduce senza dubbio fra il livello di gestione del sistema e quello del dispositivo astratto: se andiamo ad analizzare nel dettaglio le caratteristiche del primo, capiamo però che esso si trova ad un livello specifico troppo basso e rappresenta la base per la costruzione della nostra architettura. Al contrario, il livello del dispositivo astratto identifica pienamente ciò di cui ci stiamo occupando in quanto ci consente appunto di astrarre da alcuni dettagli dei singoli dispositivi offrendo la possibilità di elevarci concettualmente.

In seguito abbiamo parlato anche di modelli di dispositivi, dobbiamo quindi capire quali di questi modelli può essere supportato dall'architettura appena presentata.

Avendo preso in considerazione device riconducibili a sensori di diverso genere, risulta abbastanza scontato il supporto a **modelli discreti** che quindi non riempiono lo spazio ma costituiscono delle reti più o meno fitte. In questo settore rientrano ovviamente tutti i moderni smartphone, che verranno quindi utilizzati nel proseguo della tesi per la realizzazione concreta del nostro progetto.Non ritroviamo invece, elementi che possano essere riconducibili direttamente a modelli biologici o cellulari, anch'essi discreti ma in grado di riempire lo spazio: nel nostro caso infatti ciascun dispositivo assume valori spaziali ben definiti che nulla hanno a che fare con il riempimento di determinate aree. A maggior ragione riteniamo al di fuori della competenza di tale architettura anche i modelli continui, per gli stessi motivi presentati poc'anzi; l'unico elemento che potrebbe alludere alla continuità sarebbe riconducibile ad un'eventuale memoria condivisa in grado di annullare distanze geografiche.

Per quanto riguarda i tipi di comunicazione che erano stati individuati, non abbiamo ragioni per doverne escludere alcuno: la nostra architettura non preclude infatti la possibilità di comunicazioni locali nè globali, così come vi è la possibilità di effettuare trasmissioni broadcast, unicast e multicast.

5.6 Scenari applicativi supportati

Nel capitolo precedente abbiamo presentato diversi scenari applicativi, ora cerchiamo quindi di analizzare quali di essi possano essere affrontati positivamente attraverso il modello architetturale descritto e quali punti deboli possano invece essere individuati.

- **Visita al museo:** fra i requisiti fondamentali avevamo sottolineato la necessità di adaptivity legata ai cambiamenti della topologia del sistema e la possibilità di comportarsi in base a nozioni non pregresse ma acquisibili a tempo di esecuzione: entrambi gli aspetti vengono pienamente garantiti dalla caratteristica della mobilità e dall'esistenza di una memoria condivisa a spazi di tuple. La presenza di responsabili specifici per ogni servizio e la possibilità di comunicazione con tutti gli elementi del sistema consentono inoltre di definire una determinata politica di accessi e di governance in base alla presenza all'interno di uno o dell'altro workspace.

- **Guida della folla:** il sistema di base richiesto per questo tipo di scenario è fondamentalmente molto simile a quello precedente soprattutto per le carat-

teristiche enunciate in precedenza. Oltre a ciò troviamo concetti legati alla diffusione, alla distanza e al vicinato, che possono essere coperti tranquillamente attraverso le strutture di questa architettura semplicemente grazie ad una gestione particolare dei messaggi e o dello spazio di tuple.

- **Movimenti in aeroporto:** in questo scenario troviamo soprattutto elementi legati alla situatedness e alla diversità, entrambi supportati anche se a livelli concettuali differenti. Nel primo caso le informazioni spaziali possono essere sfruttate per definire il comportamento degli oggetti appartenenti al sistema in quanto non abbiamo imposto restrizioni sulla staticità degli elementi, nel secondo caso invece, la modularità dell'architettura permette di sviluppare sopra di essa servizi e sistemi di natura differenti senza alcuna criticità.

Gli altri scenari proposti al di fuori del progetto SAPERE possono essere in qualche modo ricondotti alle tematiche affrontate fin qui, quindi possiamo affermare con soddisfazione che in linea di massima il nostro modello architetturale è in grado di gestire quanto visto negli scenari del capitolo precedente. In questo senso ci si dovrà però concentrare sui meccanismi che definiscono i costrutti di vicinanza e di diffusione in modo da rifinire e specializzare tali concetti che al momento sono stati presentati solo in forma embrionale.

Capitolo 6

MAPPING ARCHITETTURA ASTRATTA SU MIDDLEWARE ESISTENTE

In seguito alla definizione più o meno dettagliata riguardante i diversi aspetti di una possibile architettura astratta e dei meccanismi necessari alla creazione di un middleware general purpose che ci permetta di sfruttare appieno le tecnologie mobili presenti allo stato dell'arte, andremo a considerare come tale architettura e tali astrazioni possono essere mappate su middleware già esistenti e tramite quali metodi questo mapping risulta più efficace.
Nel nostro caso, vista la precedente esperienza personale e il consolidato supporto alla piattaforma mobile Android, è stato preso in considerazione JADE: un framework ad agenti che andremo ora ad introdurre brevemente.

6.1 Java Agent DEvelopment Framework

Jade è un framework software completamente implementato in Java che semplifica la realizzazione di sistemi multi-agente attraverso un middleware FIPA-compliant e un insieme di strumenti che supportano le fasi di debugging e di sviluppo, esso è distribuito da Telecom Italia in maniera open source sotto licenza LGPL (Lesser General Public License Version 2).

6.1.1 Aspetti generali

La piattaforma ad agenti può essere distribuita attraverso macchine che non necessitano della presenza dello stesso sistema operativo e che possono essere controllate attraverso una GUI remota: la configurazione può essere cambiata a tempo di esecuzione spostando gli agenti da un host ad un altro a seconda delle esigenze, ma il supporto alla mobilità è limitato a spostamenti realizzati all'interno di una stessa istanza del middleware.

La problematica della concorrenza viene affrontata da Jade seguendo due modelli differenti a seconda degli ambiti di interesse:

- inter-agent, ogni agente ha il suo thread Java dedicato e viene schedulato in maniera preemptive;
- intra-agent, il comportamento di ciascun agente viene definito attraverso diversi behaviour fra loro cooperativi e quindi non interrompibili.

Jade, tramite l'ACC (Agent Communication Channel) presente in ogni piattaforma, offre un meccanismo di trasporto basato su messaggi asincroni, in cui questo sistema di message passing distribuito controlla tutti gli scambi effettuati all'interno della piattaforma, siano essi locali o remoti, provvedendo alla serializzazione e deserializzazione del contenuto. Jade distingue due tipi di comunicazione, a seconda della posizione degli agenti coinvolti:

- intra-platform, se scambiati tra agenti appartenenti a container diversi della stessa piattaforma;
- inter-platform, se scambiati tra agenti appartenenti a piattaforme differenti.

La consegna locale avviene tramite event-dispatching mentre quella fra container diversi sfrutta Java RMI, in caso di piattaforme diverse si utilizza invece HTTP o XMPP: il contenuto dei messaggi può essere definito tramite ontologie (quindi metamodelli) a loro volta user-defined e di conseguenza customizzabili a seconda delle esigenze applicative, stiamo quindi parlando di meta-modelli specifici. Tali ontologie hanno il limite di consentire solo due tipi di relazioni ("is a", "part of"), ma tale soluzione richiede una limitata capacità di calcolo e permette quindi di essere utilizzata anche su dispositivi mobili, oltre che garantire la possibilità di realizzare ontologie utilizzabili nel maggior parte dei casi.

Seguendo le disposizioni FIPA, il middleware presenta i seguenti elementi:

- Agent Management Service (AMS);
- Directory Facilitator (DF);
- Message Transport Service (MTS).

La natura distribuita del sistema viene realizzata attraverso i cosiddetti agent container: una tecnica che non appartiene alle specifiche FIPA ma che consente la presenza di diverse istanze dello stesso middleware distribuite su uno o più nodi. La mobilità è quindi concepita come un trasferimento degli agenti da un container ad un altro all'interno dello stesso middleware, la cui unica istanza si trova distribuita su più host. Un insieme di servizi estendibili offrono agli agenti un supporto verso il middleware, l'architettura di migrazione di cui abbiamo parlato prima è appunto implementata tramite uno di questi servizi.
Come anticipato in precedenza, ci vengono messi a disposizione dei tool che consentono di analizzare il comportamento degli agenti e lo stato del middleware:

- Sniffer and Introspector agents;
- Graphical user interface (RMA).

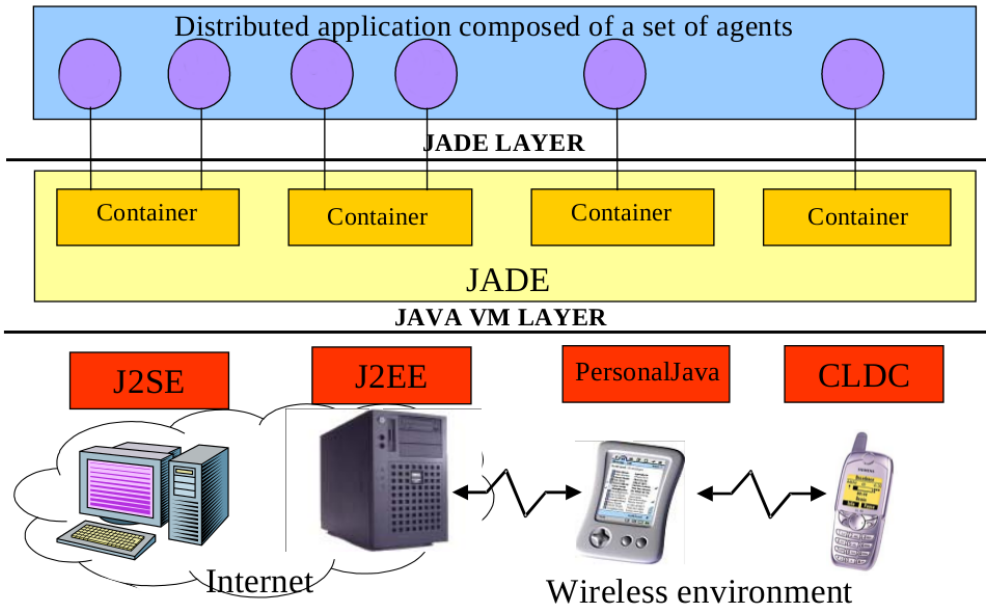

Figura 6.1: architettura generale di Jade

6.1.2 Alcuni dettagli implementativi

Nella piattaforma sono presenti diversi host, ognuno dei quali rappresenta uno o più container di agenti e quindi un ambiente completo: il primo ad essere avviato viene definito main container,esso è responsabile del mantenimento di un registro contenente i riferimenti a tutti gli altri container della stessa piattaforma.
In ogni piattaforma è presente almeno un AMS (Agent Management System) che tiene traccia di tutti gli agenti presenti anche tra i container remoti: esso deve essere contattato dagli agenti per la registrazione alla piattaforma, senza la quale l'agente non esiste. Le registrazioni permettono di mantenere un servizio di pagine bianche col quale reperire gli agenti tramite il loro identificativo e indipendentemente dalla loro posizione momentanea.
In ogni piattaforma esiste un DF (Directory Facilitator) che tiene traccia di tutti i servizi offerti dagli agenti di quella piattaforma: mantiene un servizio publish/subscribe di pagine gialle ma in precedenza deve ovviamente essere contattato da ciascun agente che vuole offrire un qualsiasi servizio.

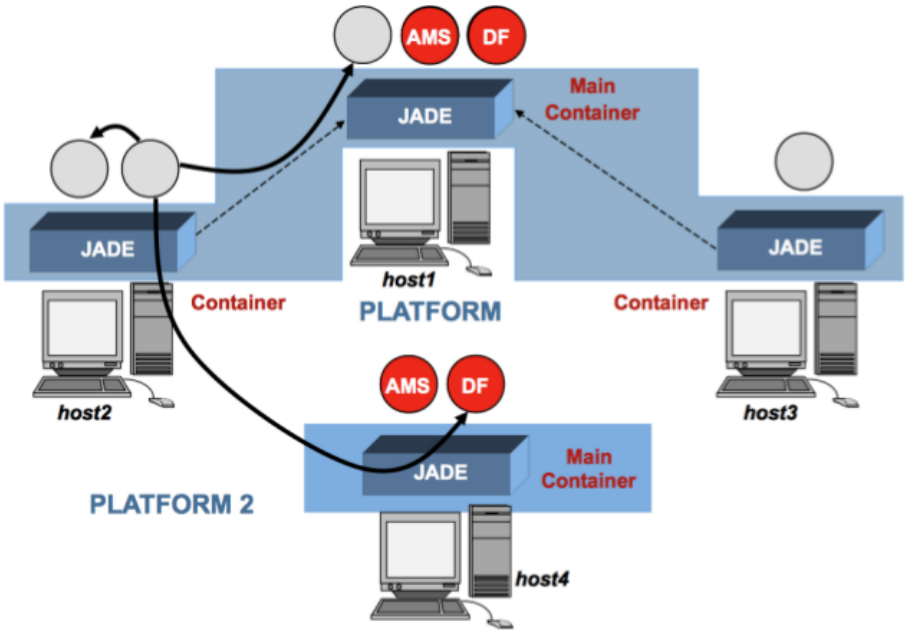

Figura 6.2: strutture principali in Jade

Di seguito andremo ad esplorare nel dettaglio alcune tematiche particolari.

MODELLO DEGLI AGENTI, CICLI DI VITA E BEHAVIOUR

Ogni classe di agenti deve estendere jade.core.Agent mentre ciascuna istanza di tali classi viene eseguita su un singolo thread.Ogni agente ha un globally unique name (aid) nella forma local_name@platform_name, ma il nome completo viene utilizzato solo in caso di comunicazione fra piattaforme diverse. La business logic di ogni agente viene descritta in termini di behaviours mentre la comunicazione viene affidata a messaggi che seguono le specifiche strutturale FIPA ACL.
Durante il suo ciclo di vita l'agente si trova in uno dei seguenti stati:

- Initiated, creato ma non ancora registrato e quindi privo di aid;
- Active, registrato e in grado di eseguire i propri behaviours;
- Waiting, bloccato nell'attesa di un messaggio;
- Suspended, stoppato senza che nessuno dei suoi behaviours sia in esecuzione;
- Transit, è in corso il suo processo di migrazione;

- Unknown, fase conclusiva che segue la sua deregistrazione.

Di seguito le operazioni che comportano cambi di stato:

- doSuspend() da ACTIVE o WAITING a SUSPENDED;
- doActivate() da SUSPENDED a qualsiasi stato in cui si trovasse al momento della chiamata alla doSuspend();
- doDelete() da un qualsiasi stato a UNKNOWN;
- doWait() da ACTIVE a WAITING;
- doWake() da WAITING a ACTIVE;
- doMove() da un qualsiasi stato a TRANSIT.

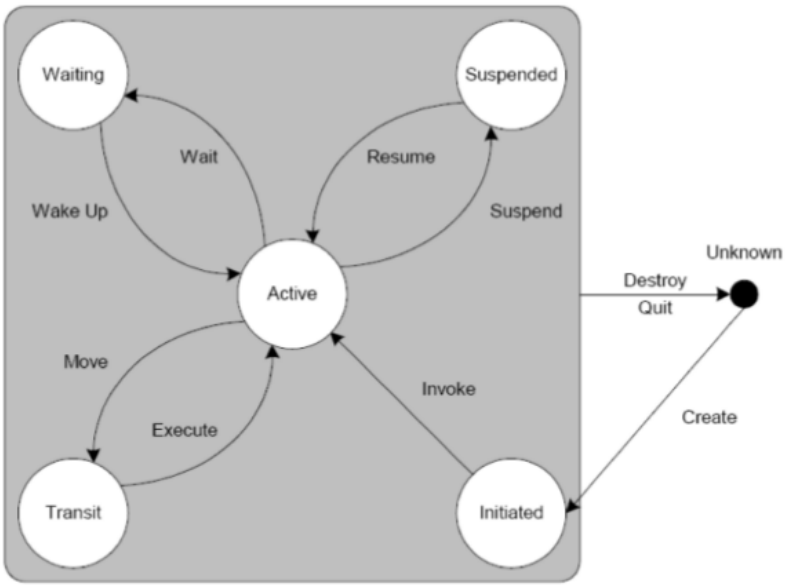

Figura 6.3: ciclo di vita di un agente Jade

La sequenza delle operazioni che vengono eseguite in fase di costruzione dell'agente è la seguente:

1. Viene eseguito il costruttore dell'agente.
2. Viene assegnato all'agente un AID in base alla piattaforma su cui è stato creato.
3. In seguito alla chiamata del metodo register() l'agente viene registrato all'AMS.
4. L'agente entra nello stato attivo.
5. Viene eseguito il corpo del metodo setup().
6. Comincia lo scheduling dei behaviours associati all'agente.

Un behaviour estende la classe jade.core.behaviours.Behaviour e viene definito come una sequenza di attività da eseguire allo scopo di completare un task, tali attività possono essere proattive e quindi eseguite spontaneamente dall'agente, oppure reattive e quindi in risposta ad eventi verificatisi come la ricezione di un messaggio o la scadenza di un timeout. Essi vengono eseguiti concorrentemente da un thread Java utilizzando uno scheduler round- robin non-preemptive, completamente trasparente all'utente: il corpo del metodo action descrive lo specifico task dell'agente mentre done viene utilizzato per controllare la condizione di terminazione del task.
I comportamenti non sono eseguibili contemporaneamente nè interrompibili, quindi può verificarsi uno switch solo nel caso in cui quello in esecuzione termini; nel caso in cui completi la sua sequenza di azioni senza che la condizione di terminazione sia soddisfatta, esso viene reinserito all'interno del pool di scheduling.

MOBILITÀ

Jade supporta meccanismi di mobilità strong, questo termine indica la possibilità di trasferire/ clonare non solo il codice dell'agente ma anche il suo stato seguendo questa sequenza:

1. si ferma l'esecuzione dell'agente nel container locale;
2. si muove/clona l'agente nel container remoto (se il codice non è disponibile nel nuovo container esso viene recuperato automaticamente dalla piattaforma Jade);

3. si riprende l'esecuzione dell'agente nello stesso punto esatto in cui era stata interrotta. Per essere trasferito o clonato, ogni agente deve appartenere ad una classe Serializable.

Il luogo in cui gli agenti possono trasferirsi è rappresentato dall'interfaccia jade.core.Location, la quale viene implementata da due classi differenti a seconda degli scenari in questione:

- ContainerID per la mobilità interna alla piattaforma;
- PlatformID per la mobilità fra piattaforme differenti (richiede la presenza di un add-on da installare.

COMUNICAZIONE

I messaggi asincroni in ricezione vengono immagazzinati in una coda identificabile con una mailbox: l'agente viene notificato ad ogni nuovo inserimento ma sarà solo lui a decidere quando e quali messaggi processare. Esso ha a disposizione le seguenti primitive:

- send() invia un messaggio asincrono;
- receive() recupera in maniera asincrona il primo messaggio dalla mailbox (se ce ne sono);
- receive(MessageTemplate) esegue una receive selettiva;
- blockingReceive() esegue una receive sincrona sospendendo i comportamenti di tutti gli agenti e non solo di quello chiamante;
- blockingReceive(long) esegue una receive sincrona a tempo;
- blockingReceive(MessageTemplate) esegue una receive sincrona e selettiva;
- blockingReceive(MessageTemplate, long) esegue una receive sincrona, selettiva e a tempo.

Il meccanismo di recupero selettivo segue un preciso pattern specificato al momento della chiamata di tale metodo senza seguire la coda presente; nel caso in cui il tentativo di recupero non produca risultati, l'eventuale scadere del timeout risveglia

l'agente che proseguirà il proprio behaviour.

CONTENUTO DEL MESSAGGIO

Ciascun messaggio è ovviamente costituito da una serie di campi che devono essere riempiti (manualmente, tramite metodi specifici che lo fanno semi-automaticamente, o completamente in maniera automatica):

- sender, campo settato in automatico che identifica il mittente;
- receiver, campo da settare che identifica il destinatario (anche indirizzi mutipli);
- performative, indica la tipologia di messaggio possibile:
  - CFP (Call For Proposal) per ottenere proposte
  - INFORM per mettere qualcuno a conoscenza di qualcosa
  - PROPOSE per proporre qualcosa
  - REQUEST per richiedere un servizio
  - SUBSCRIBE per sottoscriversi a una notifica
  - AGREE per segnalare consenso
  - REFUSE per rifiutare una proposta

6.2 Mapping dei principali elementi infrastrutturali

Partiamo dall'analizzare quelli che erano stati considerati gli elementi fondamentali per un middleware ad agenti, effettuando un parallelo con le strutture presenti all'interno di Jade: essendo esso un framework di questo tipo, ci aspettiamo che tutti questi requisiti vengano in qualche modo soddisfatti.

- meccanismi di identificazione degli agenti -> Agent Management Service (AMS);
- strumenti di comunicazione -> Agent Communication Channel (ACC) e Message Transport Service (MTS);
- meccanismi di gestione dei movimenti degli agenti -> Location;
- componenti al contorno di gestione dei servizi -> Directory Facilitator (DF);

- gestione agenti esterni al middleware -> non sono presenti meccanismi diretti ma è consentità la mobilità e si possono richiedere informazioni ai DF anche di altre piattaforme.

Abbiamo poi parlato dei componenti strutturali imprescindibili all'interno di un Multi Agent System:

- gli agenti, vengono ovviamente mappati in maniera diretta con gli agenti Jade di cui abbiamo elencato le caratteristiche in precedenza;
- i workspaces, vengono rappresentati dal concetto di container che rispecchia perfettamente le peculiarità spaziali di cui abbiamo parlato nel capitolo precedente.
- gli artefatti, Jade non prevede al suo interno astrazioni che possano rimandare a tale concetto.

ARTEFATTI

La suddetta mancanza risulta decisamente rilevante in quanto sarà molto difficile fare a meno di questo costrutto o comunque saremo costretti a dover utilizzare workaround perlomeno complessi. Cerchiamo quindi di capire più a fondo gli aspetti fondamentali che verranno a mancare per analizzare possibili soluzioni alternative. Come anticipato nel capitolo precedente, la nozione di artefatto è volta ad individuare ogni entità passiva che appartiene all'ambiente di lavoro e che esiste al di fuori delle responsabilità degli agenti, ma che è creata e utilizzata da questi ultimi per portare a termine le proprie attività. Un artefatto può essere descritto tramite 4 elementi base:

- l'interfaccia di utilizzo;
- le istruzioni operative;
- la funzione;
- la struttura e il comportamento.

La funzione e il relativo comportamento vengono partizionati in un insieme di operazioni che gli agenti possono attivare operando sull'interfaccia di utilizzo dell'artefatto in questione: essa offre infatti tutti i controlli che permettono all'agente di interagire

governando l'esecuzione delle operazioni e percependo eventi generati dall'artefatto come risultato della terminazione di tali operazioni o del cambiamento del suo stato interno.

In un certo senso, gli artefatti possono essere visti come un modo naturale di implementare servizi, senza la necessità di "agentificarli" come avviene invece ad esempio in Jade dove vengono concepiti come un concetto puramente architetturale. In qualche modo gli artefatti sono riconducibili agli oggetti nella programmazione orientata agli oggetti e ai monitor come vengono definiti in quella concorrente: i campi descrivono infatti lo stato interno e i meccanismi nascosti agli agenti, mentre il comportamento viene strutturato in un insieme di operazioni che definiscono la funzione complessiva. Considerando l'artefatto un componente condiviso fra gli agenti, deve essere dedicata particolare attenzione all'aggiornamento del suo stato e per questo, se si vuole offrire la possibilità di concorrenza e interleaving, si richiede la suddivisione delle operazioni in più step eseguibili atomicamente.

È abbastanza immediato osservare che non tutte le entità coinvolte in un MAS siano adatte per essere modellate tramite un sistema orientato o governato da un obiettivo, esse possono ovviamente essere racchiuse all'interno di agenti, ma questa soluzione si rivelerebbe uno stratagemma piuttosto che una scelta ingegneristica: modellare aspetti di un sistema attraverso astrazioni che non sono state concepite per questo, ha un impatto negativo soprattutto quando il sistema tenderà a diventare complesso e il dominio richiederà forme di controllo dinamiche ed evoluzione.
Essendo però gli artefatti dispositivi computazionali progettati per offrire un qualche tipo di servizio sfruttabile dagli agenti, una soluzione a nostra disposizione sarà quella di realizzarli attraverso gli agenti stessi effettuando possibilmente una distinzione fra i veri agenti proattivi e queste entità passive.

KERNEL SERVICES

Jade non mette quindi a disposizione un'astrazione assimilabile alla nozione di artefatto che abbiamo appena analizzato, ma ci presenta un concetto simile a quello di servizio (che abbiamo comunque introdotto in precedenza) sfruttando meccanismi di basso livello senza interessare direttamente gli agenti. Consideriamo ora un'immagine che ci mostra l'architettura dei singoli nodi sezionata verticalmente, per capire la struttura dei diversi strati che li compongono.
Ogni container Jade si trova alla sommità della struttura che compone ciascun nodo,

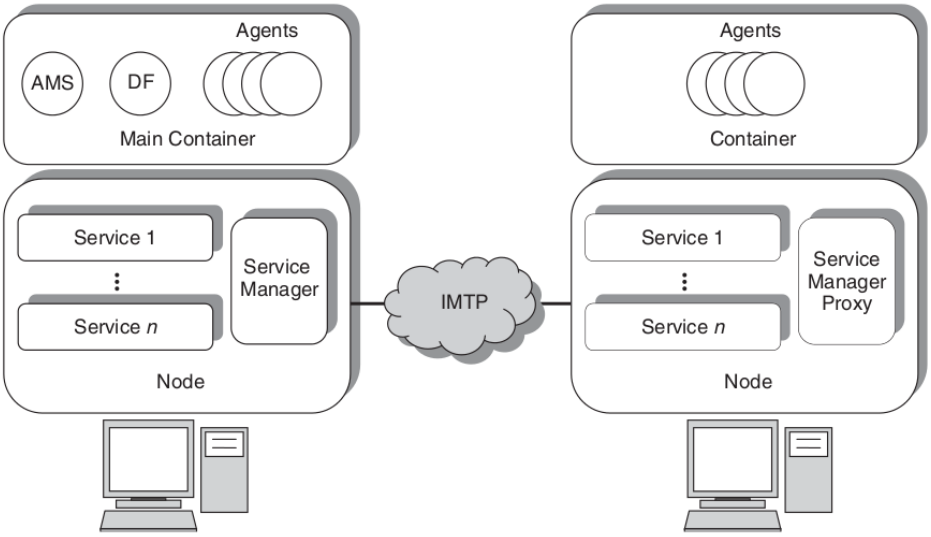

Figura 6.4: architettura dei filtri distribuiti

ed è progettato per contenere gli agenti, ogni nodo può ospitare particolari servizi definiti kernel services: ciascun servizio implementa un insieme di caratteristiche a livello di piattaforma che possono essere raggruppate seguendo la loro coesione concettuale. Alcuni esempi pratici sono individuabili nel Messaging Service che ha che fare con la consegna dei messaggi ACL scambiati fra gli agenti, nell'Agent Management Service che controlla tutte le operazioni relative all'attivazione e alla terminazione degli agenti e nel Mobility Service che implementa tutti i meccanismi riguardanti la migrazione degli agenti fra i container. Il componente che gestisce l'attivazione dei servizi all'interno dei nodi è chiamato Service Manager: in realtà ne esiste una sola istanza e risiede all'interno del nodo che ospita il main container, tutti i Manager presenti nei nodi periferici sono solamente proxy all'unico elemento reale. Il compito di questo elemento è quello di mantenere una tabella di tutti i nodi presenti nella piattaforma e la lista di tutti i servizi attivi su ogni nodo.

Come è possibile percepire dall'immagine precedente, possiamo individuare due tipologie di flussi riguardanti i servizi: quelli verticali intra-nodali e quelli orizzontali inter-nodali. Il comando verticale che parte da una richiesta specifica dell'agente, si trasmette all'interno del nodo attraverso una catena di filtri specifici, ognuno dei quali effettua una parte delle operazioni di pre-processamento necessarie, fino ad arrivare al sink sorgente finale che completa il compito. Nel caso in cui il servizio interessi

più nodi appartenenti alla stessa piattaforma, ognuno con la necessità di interagire al fine di completare il servizio, il sink sorgente dovrà preoccuparsi di individuare quale sia il container d'interesse da contattare e il relativo slice, ovvero il componente remoto responsabile della ricezione delle richieste. A quel punto potrà eseguire lui stesso l'operazione richiesta dal comando orizzontale oppure lanciare personalmente un comando verticale delegando al target sink che lo smisterà all'interno del nodo tramite una serie di filtri.

Per i servizi nativi presenti in Jade, le operazioni a livello di agenti vengono solitamente gestite dal container sottostante il quale è responsabile del passaggio delle informazioni dagli agenti allo specifico servizio per il processamento. Se si adottasse lo stesso metodo anche per i servizi user-defined questo comporterebbe la necessità di modificare il codice del container ogni volta che si definisce un servizio specifico: questo e altri motivi hanno portato alla ridefinizione dell'architettura che è ora completamente modulare e permette di risolvere questa tematica attraverso un'astrazione chiamata serviceHelper. I serviceHelper consentono agli agenti di accedere direttamente alle feature offerte dai servizi: ciascun agente può richiedere l'helper relativo ad un servizio invocando il metodo getHelper() e specificando come parametro di ingresso il nome del servizio. Esso restituirà un oggetto che implementa l'interfaccia jade.core.ServiceHelper, si rende quindi necessario un downcast prima di poter utilizzare i metodi specifici del servizio. Questa risulta quindi essere la seconda opportunità a nostra disposizione per realizzare in qualche modo le entità individuate come artefatti.

Nel capitolo precedente abbiamo individuato una struttura a livelli per la nostra architettura astratta, cerchiamo ora di capire se e come essa possa essere rappresentata in Jade.

- Il **livello fisico** viene reso attraverso le virtual machine Java presenti nei vari host e i collegamenti di rete che ne permettono l'interazione;
- il **livello strutturale** è costituito dai concetti Jade di piattaforma, container e da tutti quei componenti come l'AMS (Agent Management System) e il DF (Directory Facilitator) che definiscono i pilastri del sistema.
- il **livello comunicativo**, come esposto in precedenza, viene realizzato mediante un kernel service (Message Transport Service) ed è riconducibile a tutto ciò che è relativo all'ACC (Agent Communication Channel).

- il **livello di navigazione** è rappresentato anch'esso da un kernel service, ovvero il Mobility Service che implementa tutti i meccanismi riguardanti la migrazione degli agenti e la mobilità in generale.

- il **livello applicativo** viene gestito dallo sviluppatore definendo i vari cicli di vita degli agenti, i loro comportamenti e le loro interazioni comunicative.

Analizziamo ora le corrispondenze riguardanti gli altri aspetti più specifici. Nel capitolo precedente abbiamo parlato di due componenti che gestiscono la registrazione e l'identificazione degli agenti definendoli responsabili e disponendoli nei vari workspace. In Jade, come già anticipato in precedenza, tutti i meccanismi in questione sono presi in carico dall'AMS, il quale mantiene anche un servizio di reindirizzamento agli agenti specifici desiderati: in questo caso ci affidiamo quindi completamente ai meccanismi di gestione offertici da Jade senza complicare la situazione con soluzioni parallele poco sensate. Questo comporta la presenza di un solo "responsabile superiore" (l'AMS) presente nel main container e la rinuncia ai vari responsabili presenti in ciascun container (i corrispondenti dei workspace) rimandando tutto il controllo nella mani di una sola entità.

Avevamo inoltre parlato di due metodi di comunicazione differenti a seconda delle esigenze strutturali:

- quello diretto tramite messaggi per meccanismi di coordinazione e mobilità con entità superiori, viene automaticamente offerto da Jade attraverso le regole definite per ciascuno dei suddetti meccanismi. Lo scambio di messaggio viene considerato da Jade locale se effettuato all'interno della stessa piattaforma quindi non vi è alcun tipo di complicazione nel caso in cui due agenti appartenenti a container differenti debbano comunicare fra loro; l'unica questione si pone nel momento in cui il mittente deve individuare le generalità del destinatario, ma Jade provvede offrendo diverse alternative:

  - usando il nome locale nel caso in cui lo si conosca;
  - chiedendolo all'AMS;
  - chiedendolo al DF;
  - tramite la RMA (GUI).

- quello indiretto tramite spazio di tuple per collaborazione fra entità di pari livello, non è previsto dal framework che stiamo analizzando e deve quindi essere introdotto per mezzo di agenti pseudo-artefatti o kernel service. Essendo la memoria condivisa in un sistema distribuito un add-on non elementare, rimandiamo al paragrafo successivo le scelte in merito.

Per quanto riguarda la mobilità, si era presentato un possibile protocollo costituito da 6 punti da adottare in fase di richiesta di trasferimento degli agenti: anche in questo caso, come in quello dei responsabili, ci adeguiamo alle strutture offerte da Jade in quanto esse rispecchiano in maniera molto fedele le dinamiche di quanto avevamo proposto nel capitolo precedente. Abbiamo quindi definito 4 componenti legati direttamente alla generazione degli eventi spaziali a cui siamo interessati (gpsManager, connectionManager, accelerationManager, rotationManager): essi sono completamente identificabili con la nozione di artefatto che, come abbiamo già visto, non è presente in Jade e deve quindi essere interpretata anche in questo caso con la solita scelta fra kernel service e agentificazione.

6.3 Scelte progettuali

Dopo aver analizzato il mapping dei principali elementi infrastrutturali ed aver sollevato alcune problematiche in merito, effettuiamo ora delle scelte progettuali a tale proposito inserendo nella nostra piattaforma alcune astrazioni spaziali di nostro interesse.

6.3.1 Limitazione delle comunicazioni e selezione posizionale dei destinatari

Abbiamo visto come Jade renda trasparente al programmatore la comunicazione all'interno della stessa piattaforma, consentendo a ciascun agente di parlare liberamente con ogni altro agente interno anche se appartenente ad un altro container. I concetti di località e vicinato, tanto cari all'ambito spaziale di cui ci stiamo occupando, ci pongono però la questione della selezionabilità degli agenti in base a caratteristiche posizionali, in particolare la possibilità di inviare determinati messaggi solo ad agenti appartenenti allo stesso container.

Per analizzare questa situazione dobbiamo approfondire il meccanismo di selezione dei destinatari che abbiamo affrontato solo sommariamente in precedenza: la selezione dei destinatari può essere effettuata chiedendo all'AMS una lista di agenti che rispecchi una certa descrizione. La ricerca nell'AMS viene effettuata specificando:

- l'agente che compie la ricerca;
- una descrizione che rappresenti un filtro di ricerca;
- vincoli sul numero degli agenti che si vuole come risultato della ricerca.

```
SearchConstraints c = new SearchConstraints();
c.setMaxResults ( new Long(-1) );
agents = AMSService.search(this, new AMSAgentDescription (), c );
```

L'unico modo di soddisfare le nostre esigenze è approfondire quali possibilità possono essere sfruttate a livello di AMSAgentDescription, scopriamo così che in questo modo non è possibile ottenere ciò che stiamo cercando. Jade mette invece a disposizione una JADE-Agent-Management ontology, la quale descrive tutte le operazioni di gestione della piattaforma che possono essere richieste all'AMS.
Notiamo quindi che le classi QueryAgentOnLocation e WhereIsAgentAction offrono allo sviluppatore esattamente le astrazioni spaziali di cui avevamo bisogno, senza la necessità di doverle in qualche modo costruire attraverso meccanismi più complessi come la registrazione di particolari servizi/descrittori al DF. Di seguito un possibile esempio del messaggio di richiesta da inviare all'AMS.

```
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
request.addReceiver(getAMS());
request.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
request.setOntology(MobilityOntology.NAME);
request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
Action act = new Action();
act.setActor(getAMS());
QueryAgentsOnLocation action = new QueryAgentsOnLocation();
Location myLocation = here();
action.setLocation(myLocation);
act.setAction(action);
```

Action name	Class	Description
create-agent	CreateAgent	Create an agent of a given class with a given name in a given container
kill-agent	KillAgent	Kill a given agent
kill-container	KillContainer	Kill a given container
shutdown-platform	ShutdownPlatform	Shut down the whole JADE platform
query-platform-locations	QueryPlatformLocationsAction	Retrieve the list of all containers in the platform. This is returned as a jade.util.leap.List of Location (actually ContainerID) objects
query-agents-on-location	QueryAgentsOnLocation	Retrieve the list of agents currently living in a given container. This is returned as a jade.util.leap.List of AID objects
where-is-agent	WhereIsAgentAction	Retrieve the ContainerID of the container where a given agent currently lives
install-mtp	InstallMTP	Activates a new MTP of a given class in a given container. The URL of the installed MTP is produced as the result
uninstall-mtp	UninstallMTP	Deactivates an MTP with a given URL in a given container

Figura 6.5: Jade-Agent-Management ontology

```
getContentManager().fillContent(request, act);
```

Nei linguaggi che abbiamo affrontato all'interno del capitolo 3, molti dei quali tuple-oriented, venivano offerti meccanismi legati all'hop-count conseguente alla diffusione di un particolare gradiente che definiva distanze e località ben definite, in modo da concedere la possibilità di simulare determinate restrizioni in fase di comunicazione.
Come già anticipato in precedenza, nel nostro caso non abbiamo la possibilità di estendere il concetto di località andando oltre quello di container, e il meccanismo dell'hop count dovrebbe essere realizzato tramite messaggi seguendo una certa logica prestabilita da una qualche entità superiore in base a specifiche regole a priori, perdendo in qualche modo il significato posizionale (dinamico) originale.

6.3.2 Realizzazione dei 4 componenti e loro funzionamento

Abbiamo introdotto dei componenti per ogni elemento hardware indispensabile alla generazione di particolari eventi spaziali, dobbiamo quindi ora definire come essi verranno inseriti nel contesto di una piattaforma Jade e quali saranno i meccanismi che regoleranno il loro funzionamento. Partiamo dal presupposto di identificare un

dispositivo attraverso un agente, ciascuno di essi avrà al suo interno determinate caratteristiche hardware che gli consentiranno di rilevare aspetti diversi della sua posizione e del suo comportamento: nel caso in cui l'agente in questione volesse ottenere determinati valori, esso dovrebbe effettuare una richiesta al componente in questione, il quale gli fornirebbe i dati attraverso un determinato meccanismo di interazione con la parte fisica del dispositivo e tramite la generazione di una sorta di evento.

Tale analisi evidenzia la necessità per il componente di interagire con il livello fisico di ciascun dispositivo, o di delegare ad altri l'incarico in questione: questo ci suggerisce che l'effettivo responsabile dell'operazione deve trovarsi sullo stesso nodo del dispositivo.Riteniamo quindi che l'idea migliore sia quella di identificare i servizi offerti dai 4 componenti all'interno di uno o più kernel service disposti nel nostro middleware e quindi sotto ogni nodo facente parte della nostra piattaforma, evitando dispendiosi mandati e la creazione di ulteriori entità di supporto. Nei capitoli successivi approfondiremo poi alcuni aspetti implementativi.

6.3.3 Medium di coordinazione e spazi di tuple

In un ambiente strettamente legato al pervasive computing, soprattutto per quanto riguarda i possibili scenari applicativi, un metodo comunicativo che si limita alla possibilità di scambiare messaggi asincroni, rende il nostro sistema modesto e circoscritto, con ridotte potenzialità: la coordinazione fra gli agenti può comunque essere effettuata ma i procedimenti risultano molto più complessi e meno agili.

L'inserimento di un livello in grado di garantire i meccanismi di riferimento per la gestione di una memoria condivisa distribuita sotto forma di spazio di tuple, porterebbe al livello comunicativo un consistente arricchimento che consentirebbe ad esempio l'utilizzo del matching semantico. In realtà, Jade presenta già una qualche forma di matching che può essere sfruttata durante la selezione degli agenti sotto forma di specifici vincoli nelle richieste al DF, ma questo ha ovviamente poco a che fare con la possibilità di sfruttare un tale meccanismo per quanto riguarda il contenuto dei messaggi anzichè per gli agenti. Al momento, essendo comunque in fase di prototipazione e non avendo inizialmente complesse necessità a livello di coordinazione, preferiamo concretizzare questo concetto realizzando una sorta di blackboard gestita da un agente: esso sarà stabile all'interno del main container e manterrà al suo

interno una struttura dati sulla quale copierà il contenuto informativo dei messaggi e dalla quale preleverà le informazioni richieste tramite una semplice forma di matching; per performance e caratteristiche specifiche la struttura dati verrà realizzata adottando una serie di hash map. Siamo comunque consapevoli che nel momento in cui il sistema dovesse richiedere ulteriori funzionalità, sia a livello di specifiche operazionali su spazi di tuple, sia per quanto riguarda il concetto di centri di tuple programmabili, basterà appoggiarsi sulle strutture offerte da TuCSoN: questo progetto rappresenta un middleware di coordinazione e offre a sua volta alcune astrazioni legate ai centri di tuple contenuti in ReSpecT. Entrambi i progetti sopracitati sono java-based quindi pienamente compatibili con il framework che stiamo prendendo in considerazione: un esempio di integrazione viene descritto in una precedente tesi del collega Nicola Dellarocca, anch'esso appartenente alla Seconda Facoltà di Ingegneria di Cesena, intitolata "Coordination as a service in Jade".

La semplificazione effettuata in questo paragrafo è volta a non sovraccaricare aspetti sì importanti, ma non del tutto centrali in questo testo, preferiamo infatti concentrarci su eventuali astrazioni spaziali inerenti al nostro ambito di interesse lasciando comunque ampie possibilità di estensione per quanto riguarda gli altri aspetti: nel momento in cui la suddetta blackboard dovesse essere sostituita dai concetti presentati in TuCSoN e ReSpecT, cambieranno semplicemente alcuni metodi di interazione con il medium di coordinazione, ma non verrà assolutamente stravolta la struttura del sistema.

6.3.4 Astrazioni spaziali

Il survey sui linguaggi effettuato nel capitolo 3, aveva lo scopo di evidenziare le prerogative presenti nei linguaggi moderni in grado di affrontare tematiche spaziali: da questa rassegna prendiamo spunto per considerare alcune astrazioni che potrebbe essere utili introdurre nella nostra piattaforma. Analizziamo dunque alcuni aspetti e discutiamo sulle loro possibilità di inserimento nel nostro middleware in modo da arricchire quello che era stato definito livello di navigazione:

- avevamo esposto la possibilità di **identificare gli agenti attraverso le loro coordinate posizionali** in modo da poterne indirizzare una certa quantità in base a delle proprietà spaziali di nostro interesse. In Jade non è presente un meccanismo diretto che permetta di rinominare gli agenti cambiando il nome

che gli è stato assegnato in fase di creazione; in alternativa possiamo pensare alle coordinate quali proprietà indispensabili dell'agente che vengono registrate presso il DF come se fossero un servizio a disposizione, in tal modo chi ne richiedesse le informazioni potrebbe effettuare una sorta di selezione in merito. Questo meccanismo risulta essere abbastanza forzato e spostamenti continui porterebbero a una serie di deregistrazioni/riregistrazioni decisamente costose presso il DF: riteniamo quindi che questo stratagemma sia realizzabile solo in caso di movimenti saltuari o di aggiornamento della posizione ad intervalli non troppo ravvicinati.

- La presenza di un numero finito di agenti è discordante con l'idea di un'**astrazione spaziale continua** come quella di un mezzo amorfo in cui siano ovunque presenti delle ipotetiche entità computazionali: nel nostro caso le informazioni non scorrono tramite questo mezzo ma attraverso i canali specifici portatori di messaggi e sulla blackboard che funge da punto di raccolta. In questo modo riusciamo a presentare un abbozzo di semantica funzionale ma, per quanto detto al punto precedente, difficilmente potremo rappresentare concetti di vicinanza o distanza specifica che vadano oltre la località definità dai container.

- Si era inoltre parlato di **strutture gerarchiche e di grafi** che gestissero l'organizzazione degli agenti, in questo senso Jade non offre strutture specifiche ma consente la creazione innestata di agenti da parte di altri agenti. Questo meccanismo permette la creazione di legami fra gli agenti che possono essere interpretati generando una sorta di albero genealogico, ovviamente ogni agente padre ha i riferimenti a tutti gli agenti figli che ha creato quindi si possono comporre semplicemente dei grafi, anche se questo è abbastanza distante dal concetto originario a cui eravamo interessati.

- In un sistema multi-agente, va da sè che ci sia spesso la necessità di **interagire con più di un agente per volta**, e di conseguenza si presentino situazioni in cui si debbano selezionare i partecipanti all'azione. Nel nostro caso, si tratta per la gran parte delle occasioni di definire i destinatari di determinati messaggi, e quindi non precisamente di effettuare operazioni su insiemi di agenti, resta il fatto che in entrambi i casi le nostre possibilità si limitano alle funzionalità

offerte dall'AMS e dal DF e non siano certo specifiche e spiccate come quelle viste in precedenza.

- Avevamo introdotto l'utilizzo del **modello ambientale** da parte degli agenti come una **blackboard** da cui leggere e su cui scrivere, essendo esso una risorsa in comune a disposizione di tutti gli agenti risultava senz'altro una possibilità sfruttabile in diverse maniere. Nel nostro caso abbiamo dovuto introdurre la blackboard proprio per consentire questo tipo di meccanismo senza limitare l'interazione a semplici messaggi. Così facendo l'ambiente viene in qualche modo rappresentato da un artefatto posizionato nel main container ma raggiungibile da tutti gli agenti della piattaforma.

- Essendo i sistemi complessi composti da più entità, si presenta l'occasione di **riunire e processare dati aggregati**, magari anche in seguito a una selezione su regole/predicati di natura diversa. Abbiamo provveduto a rendere possibile un meccanismo simile con l'introduzione della blackboard, la quale permette ad un agente esterno, che potremmo definire elaboratore, di selezionare le informazioni depositate dai worker e processarle secondo le proprie necessità. A seconda della natura e degli scopi del sistema, la struttura dati dovrà essere ovviamente organizzata e predisposta in maniera diversa, dando la possibilità di scegliere con parametri vari e specifici quello a cui l'elaboratore è interessato.

- In alcuni casi, soprattutto in sistemi biologici, si faceva riferimento alla **fusione di elementi inizialmente distinti**, come potrebbe tra l'altro accadere in seguito all'unione di due componenti differenti a favore di un'unica entità robotica. Bisogna innanzitutto definire se i comportamenti dei singoli componenti rimangano immutati o se la connessione provochi mutamenti sostanziali:

  - nel primo caso, restando le sottoentità praticamente inalterate, non vi sono problemi di rappresentazione ma al massimo di indirizzamento univoco in quanto Jade continuerà a interpretarle come due entità distinte;
  - nel secondo caso invece, non abbiamo nemmeno la possibilità di rappresentazione, in quanto essa richiederebbe la costruzione a run-time di un agente con un comportamento non identificabile a priori. L'unica possibilità che abbiamo a disposizione è quella di trasferire i dati in possesso di una sottoentità e di fermare l'agente corrispondente, delegando

al destinatario il compito di rappresentanza anche del mittente: questo metodo riesce a rappresentare degnamente solo situazioni in cui viene effettuata una inclusione a puro scopo informativo, in seguito alla quale la componente inclusa ha terminato il suo compito.

6.4 Eventuale inserimento di strutture topologiche

Abbiamo parlato in precedenza di sfruttare le coordinate degli agenti e di utilizzare in modi diversi quelle che sono le informazioni spaziali a nostra disposizione, quanto però è stato detto va ricondotto a meccanismi che si situano a livello applicativo e richiedono quindi uno sforzo operativo da parte degli sviluppatori.

Quello che vorremmo garantire è invece un'infrastruttura in grado di offrire un protocollo che definisca e mantenga strutture topologiche da poter sfruttare a livello applicativo: in questo modo miglioreremmo il semplice concetto di container specificando ulteriori gradi organizzativi. Inizialmente potremmo utilizzare i classici container jade integrando le loro funzionalità tramite agenti topologici che mantengano riferimenti catturati in fase di creazione e avvio: in questo modo non potremmo garantire però requisiti di dinamicità in quanto tutti gli agenti dello stesso container avrebbero caratteristiche uniformi associate al container stesso, e solo attraverso la mobilità, avrebbero l'opportunità di variarle. In sintesi questo genere di soluzione non garantirebbe la possibilità di organizzare dinamicamente gli agenti ma offrirebbe una sorta di servizio a un livello superiore che specializzi container statici.

Indipendentemente dalla presenza all'interno di uno o dell'altro container, potrebbe avere senso offrire un altro tipo di meccanismo di specializzazione degli agenti, molto più dinamico e in grado di consentire raggruppamenti articolati legati anche a gerarchie verticali. Con un semplice kernel service custom potremmo associare a ciascun agente un attributo con significato legato alla gerarchia o alla topologia del sistema, con la possibilità di definire politiche di accesso o restrizioni nelle comunicazioni. In questo modo si lascerebbe allo sviluppatore dell'applicazione la libertà di interpretare a suo modo i valori degli attributi, di settarli sia in fase di creazione che in reazione a eventi del sistema, e di definire le politiche di funzionamento di quest'ultimo.

Con un esempio banale, possiamo assegnare un attributo numerico agli agenti in modo che i valori delle decine indichino informazioni topologiche orizzontali mentre

i valori delle centinaia definiscano gerarchia verticali. Possiamo quindi proseguire stabilendo che agenti di un livello inferiore possano comunicare con elementi di livello superiore solo in risposta a precise richieste provenienti dall'alto o che agenti dello stesso livello possano comunicare solo se la differenza del valore numerico sia inferiore a una soglia prestabilita. In questo modo, oltra a lasciare assoluta libertà in fase di costruzione dell'applicazione, abbiamo la possibilità di regolare le attività del sistema impostando dei filtri sottostanti la piattaforma: all'interno di tutto ciò potrebbero comunque continuare ad avere senso agenti topologici che osservino gli eventi e abbiano il compito di aggiornare i valori degli attributi citati poc'anzi.

Capitolo 7

IMPLEMENTAZIONE APPLICAZIONE CONCRETA

Dopo aver presentato la nostra architettura astratta, aver analizzato le possibilità che ci offre un framework ad agenti come Jade e aver descritto alcune scelte progettuali, andiamo ora ad esaminare come è stato realizzata un'applicazione concreta che implementi i principali punti oggetto della nostra discussione.

7.1 Descrizione della struttura e del funzionamento del sistema

Abbiamo deciso di realizzare un sistema distribuito su almeno 2 host che permetta la generazione e il recupero di informazioni spaziali provenienti da un dispositivo mobile:

- il primo nodo è responsabile del lancio della piattaforma Jade e dell'attivazione di due agenti che si occupano della gestione delle informazioni prodotte dal dispositivo, il tutto viene realizzato tramite normale sviluppo Java;

- il secondo nodo è rappresentato da un dispositivo Android in grado di produrre informazioni spaziali, in termini di sviluppo stiamo quindi parlando di un app mobile che sfrutta l'estensione Jade Android, la quale permette l'utilizzo di Jade Leap su tale sistema operativo.

Mentre il secondo nodo in linea di massima è identificabile con una sola entità, la stessa cosa non può essere assunta anche per il primo: al lancio della piattaforma Jade ci preoccuperemo infatti anche della creazione di due agenti. In termini di componenti effettivi del sistema individuiamo quindi:

- **DeviceAgent**, produce informazioni che invia al BlackboardAgent;
- **BlackboardAgent**, raccoglie informazioni dal DeviceAgent in una struttura dati specifica;
- **OperatorAgent**. richiede al BlackboardAgent i dati spaziali relativi ad uno specifico device.

In seguito descriveremo un ulteriore agente presente nel nodo Android il cui compito è al momento marginale rispetto a ciò che stiamo descrivendo.

7.2 App Android

L'app Android rappresenta il fulcro del nostro sistema in quanto è responsabile della generazione delle informazioni spaziali a cui siamo interessati: essa è composta da due Activity con 2 layout ben differenti:

- nella prima activity viene presentata all'utente la possibilità di connettersi ad una piattaforma Jade già esistente;
- nella seconda activity ci vengono invece messi a disposizione i mezzi per la generazione, la visualizzazione e l'invio dei dati.

La connessione alla piattaforma Jade provvede alla creazione di un secondo container differente rispetto a quello principale generato dal lancio via PC, secondo quanto consigliato dalla manualistica Jade è infatti preferibile che gli agenti mobili vengano avviati su una struttura a parte. Il comportamento dell'app prevede che il DeviceAgent ricerchi tramite il Directory Facilitator un agente in grado di fornire un servizio di shared-memory chiamato "blackboard": esso sarà comunque in grado di generare informazioni, ma finchè non troverà un agente diquesto tipo ovviamente proseguirà senza inviarle.

Nel capitolo precedente avevamo anticipato che avremmo realizzato i 4 componenti generatori degli eventi spaziali attraverso un kernel service custom: nel nostro

caso esso è in realtà abbastanza atipico in quanto non intercetta comandi verticali e orizzontali ma serve esclusivamente ad inserire un livello intermedio fra quello fisico delle API Android e le chiamate dei nostri agenti. Abbiamo quindi definito un servizio "HwAccessService" con 3 variabili interne che rappresentano la disponibilità di altrettante componenti hardware, viene data per scontata la presenza di un modulo Wi-Fi o 3g senza il quale non potrebbe funzionare l'applicazione. Tale servizio viene dunque avviato al momento della connessione precedente al passaggio di activity, partendo da valori settati inizialmente a false: in corrispondenza dell'attivazione del DeviceAgent viene dunque creato anche un HwAccessControllerAgent che si preoccupa di verificare l'effettiva disponibilità dei sensori fisici settando di conseguenza le variabili sopra citate. Sarà poi il DeviceAgent, al momento della pressione del pulsante di generazione, a verificare il valore di tali variabili e in base a ciò, ad avere o meno il permesso di richiedere un certo tipo di informazioni.

Questo meccanismo di interazione fra agenti e servizi viene realizzato tramite l'HwAccessHelper che rappresenta l'estensione del ServiceHelper base, mentre il nostro HwAccessService deve specificare la classe FEService in quanto i meccanismi legati ai servizi su piattaforma LEAP sono leggermente differenti. Una volta ottenuti i permessi per l'accesso ai componenti hardware, le informazioni vengono recuperate attraverso i metodi delle classi Interaction:

- per quanto riguarda **accelerazione e vettore di rotazione**, basterà definire il corpo del metodo onSensorChanged implementando quindi un SensorEventListener e sfruttando i valori degli eventi generati ad ogni movimento del dispositivo;

- per quanto riguarda la **connessione**, l'invocazione del metodo getActiveNetworkInfo su un ConnectivityManager ci fornirà tutto ciò di cui abbiamo bisogno;

- per quanto riguarda la **posizione**, il meccanismo è leggermente più complesso perchè sfrutta una combinazione dei dati provenienti da connessione di rete e gps. Essendo i dati gps più precisi ma più lenti da ottenere, preferiremo le informazioni da rete solo se più recenti di almeno 10 secondi rispetto alle ultime ottenute da gps, ricordiamo inoltre che i dati di rete non ci permettono di stabilire l'altitudine, cosa ovviamente possibile tramite gps.

Andiamo quindi ad analizzare nel dettaglio le caratteristiche specifiche dell'app Sensor Manager approfondendo anche attraverso alcuni stralci di codice.

7.2.1 FirstActivity

Viste le possibilità di gestione di alcuni componenti hardware presenti sui moderni smartphone, ci sembrava utile permettere il funzionamento dell'app anche nel caso in cui non sia disponibile una connessione oppure non sia attiva una piattaforma Jade che metta a disposizione una sorta di memoria condivisa.Per questo motivo la schermata iniziale mostra, oltre alla possibilità di inserire le informazioni necessarie alla connessione, anche un pulsante che consente l'avvio della parte principale dell'applicazione senza sfruttare la piattaforma Jade presente su uno pseudo-server.

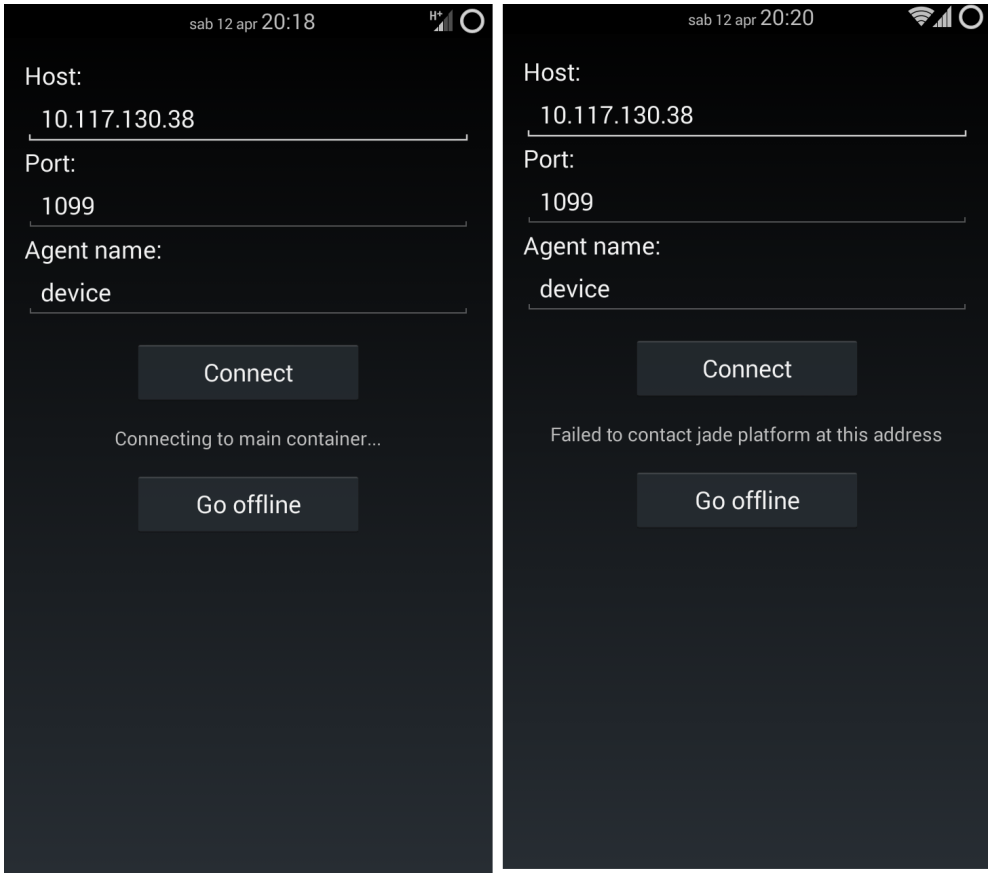

Figura 7.1: app in fase di connessione e dopo connessione fallita

Analizziamo quindi il comportamento dell'activity in base alla pressione dell'uno o dell'altro pulsante. Nel caso della pressione del tasto "offline" verrà immediatamente lanciata l'activity successiva senza passare alcun parametro:

```
buttonOff.setOnClickListener(new View.OnClickListener() {
       public void onClick(View v) {
              StartActivity.this.intent = new
                  Intent(getBaseContext(), SecondActivity.class);
              startActivity(StartActivity.this.intent);
       }
});
```

Ovviamente, più complesso sarà il procedimento da effettuare nel caso rimanente, in quanto dovranno svolgersi le seguenti attività:

- recupero dei parametri settati nelle caselle di inserimento testo;

```
StartActivity.this.host = eh.getText().toString();
StartActivity.this.port = ep.getText().toString();
StartActivity.this.nickname = en.getText().toString();
```

- creazione e binding del serviceConnection;

```
serviceConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className,
            IBinder service) {
                microRuntimeServiceBinder =
                    (MicroRuntimeServiceBinder) service;
                startContainer(host,port);
        }
};
bindService(new Intent(getApplicationContext(),
    MicroRuntimeService.class),
serviceConnection, Context.BIND_AUTO_CREATE);
```

- avvio del container Jade specificando alcune proprietà tra cui il nome del servizio custom da lanciare;

```
String services="service.HwAccessService";
Properties pr = new Properties();
pr.setProperty(Profile.MAIN_HOST, h);
pr.setProperty(Profile.MAIN_PORT, p);
pr.setProperty(Profile.JVM, Profile.ANDROID);
pr.setProperty(Profile.SERVICES, services);

microRuntimeServiceBinder.startAgentContainer(pr, new
    RuntimeCallback<Void>() {

        @Override
        public void onSuccess(Void thisIsNull) {
                startAgent(nickname, DeviceAgent.class.getName());
```

```
                startAgent(nickname + "-controller",
                    HwControllerAgent.class.getName());
        }
};
```

- avvio dei due agenti (DeviceAgent e HwControllerAgent);

```
private void startAgent(final String name, final String
    agentClass) {
        microRuntimeServiceBinder.startAgent(name, agentClass,
            new Object[] {
                getApplicationContext(), this }, new
                    RuntimeCallback<Void>() {
                ...
                })
        };
}
```

- avvio dell'attività successiva passando il nickname dell'agente

```
intent = new Intent(getBaseContext(), SecondActivity.class);
intent.putExtra("nick", StartActivity.this.nickname);
startActivity(StartActivity.this.intent);
```

7.2.2 HwControllerAgent

Prima di entrare nel dettaglio per ciò che riguarda le caratteristiche dell'agente, intendiamo presentare quello che è convenzionalmente il metodo da utilizzare nelle interazioni fra agenti e activity:

- gli agenti devono implementare un'interfaccia che metta a disposizione determinati metodi e registrare la relativa O2AInterface.ù

```
public interface ControllerAgentInterface extends Parcelable {
        void activityToAgent(String string);
        void agentToActivity(boolean[] avail);
}
```

```
HwControllerAgent extends Agent implements
    ControllerAgentInterface{

        @Override
        protected void setup() {
                ...
                registerO2AInterface(ControllerAgentInterface.class,
                    this);
        }
}
```

- quando l'activity necessita di interagire con l'agente, non dovrà fare altro che invocare il metodo specifico dopo aver ottenuto un riferimento alla sua interfaccia specificando il nome dell'agente.

```
ControllerAgentInterface devIntCtrl =
    MicroRuntime.getAgent(nickname +
    "-controller").getO2AInterface(ControllerAgentInterface.class);
devIntCtrl.activityToAgent("check-hardware");
```

- quando l'agente vuole comunicare con l'activity non dovrà fare altro che inviare un messaggio broadcast a cui l'activity dovrà ovviamente essere reattiva registrando in precedenza un BroadcastReceiver.

```
MyReceiver myReceiver = new MyReceiver();
filter = new IntentFilter();
filter.addAction("hardware-results");
registerReceiver(myReceiver,filter);

private class MyReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
                String action = intent.getAction();
                if (action.equalsIgnoreCase("hardware-results")) {
                        gps = intent.getBooleanExtra("gps", false);
                        acc = intent.getBooleanExtra("acc", false);
                        rot = intent.getBooleanExtra("rot", false);
```

```
                }
            }
    }
```

Tornando all'agente in questione, esso rappresenta una sorta di mediatore fra il livello fisico e il custom service che regola i permessi di accesso da parte dell'applicazione: si occupa infatti della fase di configurazione dei parametri che definiscono la presenza o l'abilitazione ad accedere ai 4 componenti hardware. Attraverso un OneShotBehaviour, l'agente effettua un check sulla piattaforma e setta di conseguenza le variabili del nostro custom service.

```
addBehaviour(new OneShotBehaviour() {

  @Override
  public void action() {
       try {
              HwControllerAgent.this.help = (HwAccessHelper)
                  getHelper("service.hwAccess");
       } catch (ServiceException e1) {
              e1.printStackTrace();
       }
       if(help != null) {
              PackageManager pm =context.getPackageManager();
              if
                  (pm.hasSystemFeature(PackageManager.FEATURE_LOCATION_GPS))
                  {
                     help.setGpsAvailable(true);
              }
              if
                  (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
                  != null) {
                     help.setAccAvailable(true);
              }
              if
                  (mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR)
                  != null) {
                     help.setRotAvailable(true);
              }
```

```
        } else {
                log(false, "service helper not retrieved");
        }
  }
});
```

In seguito esso sarà in grado di rispondere alle richieste provenienti dalle activity, sia per fornire tali dati, sia per dare la possibilità di abilitare/disabilitare gli accessi cambiando il valore delle variabili sopracitate.

```
@Override
public void activityToAgent(String string) {
        if(string.equals("check-hardware")) {
                try {
                        help =
                            (HwAccessHelper)getHelper("service.hwAccess");
                        gpsAv = help.isGpsAvailable();
                        accAv = help.isAccAvailable();
                        rotAv = help.isRotAvailable();
                } catch (ServiceException e) {
                        e.printStackTrace();
                }
        } else {
                if (string.equals("switchGps")) {
                        gpsAv = setGpsAvailable(!help.isGpsAvailable());
                } else if (string.equals("switchAcc")) {
                        accAv =
                            help.setAccAvailable(!this.help.isAccAvailable());
                } else if (string.equals("switchRot")) {
                        rotAv =
                            help.setRotAvailable(!this.help.isRotAvailable());
                }
        }
        agentToActivity(new boolean[]{ this.gpsAv, this.accAv,
            this.rotAv });
}
```

```
@Override
public void agentToActivity(boolean[] avail) {
        broadcast = new Intent();
        broadcast.setAction("hardware-results");
        broadcast.putExtra("gps", avail[0]);
        broadcast.putExtra("acc", avail[1]);
        broadcast.putExtra("rot", avail[2]);
        context.sendBroadcast(this.broadcast);
}
```

7.2.3 SecondActivity

Quest'activity rappresenta il fulcro dell'applicazione e viene gestita attraverso un'action bar contenente 7 fragment diversi come mostrato nell'immagine seguente: ognuno di essi mette a disposizione funzionalità legate ai diversi generatori di eventi che abbiamo identificato nei capitoli precedenti.

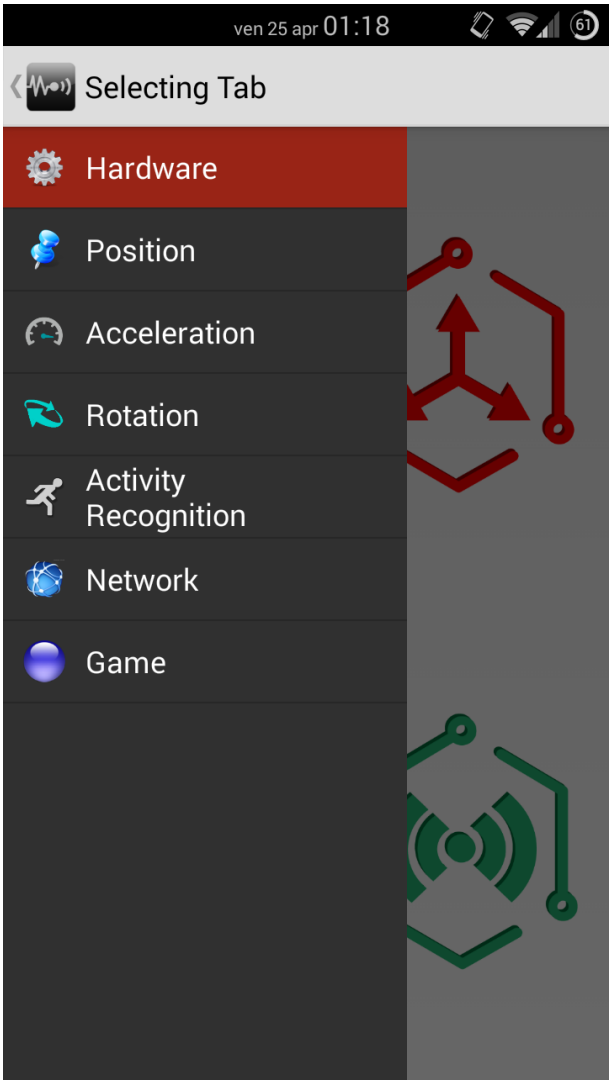

Figura 7.2: tab disponibili

Tab Hardware

Il tab "Hardware" risulta significativo solo nella versione "online" dell'app, come abbiamo mostrato in precedenza infatti, l'agente HwController viene attivato e svolge il suo compito solamente in questo caso, mentre nella versione "offline" tutti i componenti verranno abilitati di default.

Figura 7.3: screenshot che mostra l'aspetto del tab hardware

Il funzionamento di questo tab viene definito all'interno del FragmentAll e, come vediamo nell'immagine soprastante, consiste in 4 immagini reattive alla pressione da parte dell'utente: in ordine orario le icone rappresentano gps, accelerazione, connessione e rotazione. Inizialmente le icone saranno tutte di colore rosso, a indicare la non presenza dei rispettivi componenti hardware, saranno quindi colorate di verde a seconda dei risultati comunicati dall'HwControllerAgent. In seguito l'utente potrà abilitare/disabilitare l'accesso al componente in questione semplicemente premendo sull'icona corrispondente, e ovviamente come risposta visiva immediata, otterrà il

cambiamento del colore.

Tab Position

Il tab "Position" consente di utilizzare i servizi di localizzazione messi a disposizione da Google: essi sono in grado di sfruttare sia la connessione a internet sia un componente come il GPS, la differenza sta nel fatto che i dati provenienti dalla seconda opzione saranno più precisi e comprensivi dell'altitudine.

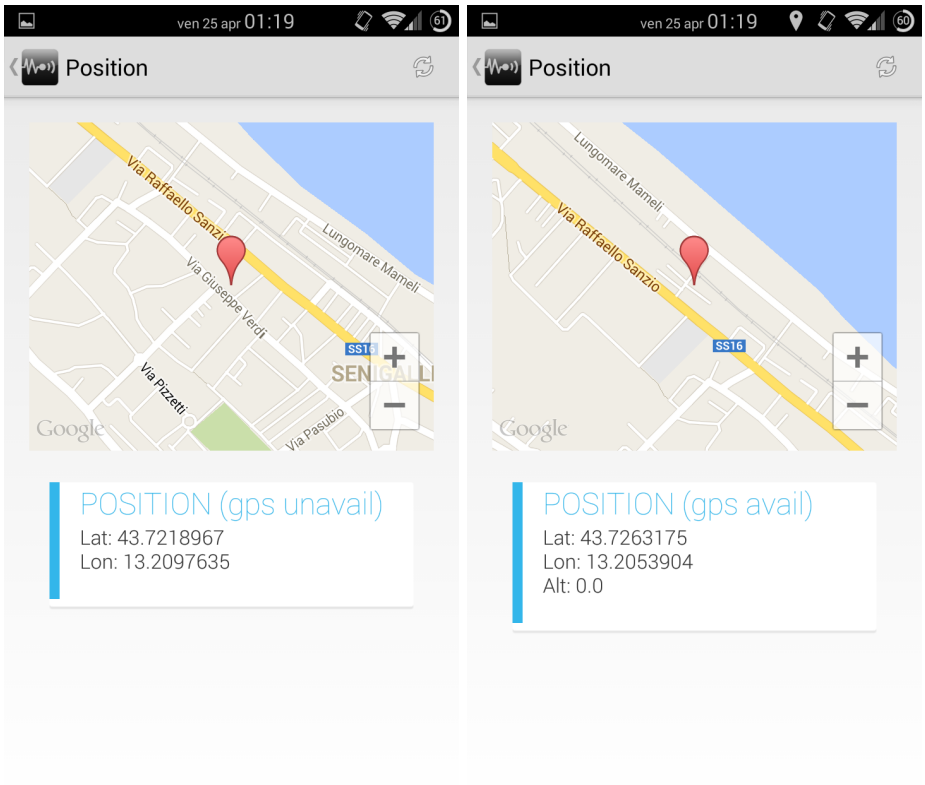

Figura 7.4: screenshot che mostra l'aspetto del tab position

Come mostrano le immagini, il PositionFragment racchiude un ulteriore fragment contenente una mappa, un'icona nel menu in alto a destra per richiedere i dati e una scheda che mostra i dati in questione. Nella versione "offline", il pulsante richiama direttamente il metodo printPos della classe PositionInteraction, mentre in quella "online" si passa attraverso il DeviceAgent in modo da dargli l'opportunità di inviare alla blackboard i dati ottenuti.

Nel caso in cui si richieda il "refresh" almeno due volte ottenendo una posizione

valida, verranno mostrate anche informazioni derivate legate alla posizione attuale e a quella precedente: stiamo parlando di distanza, velocità ed orientamento esattamente come anticipato nel paragrafo 5.2. In questo e nei casi seguenti, vengono utilizzate delle schede personalizzate, realizzate grazie alla libreria CardsUI, che possono essere eliminate tramite uno swipe laterale oppure portate in primo piano con un semplice tocco.

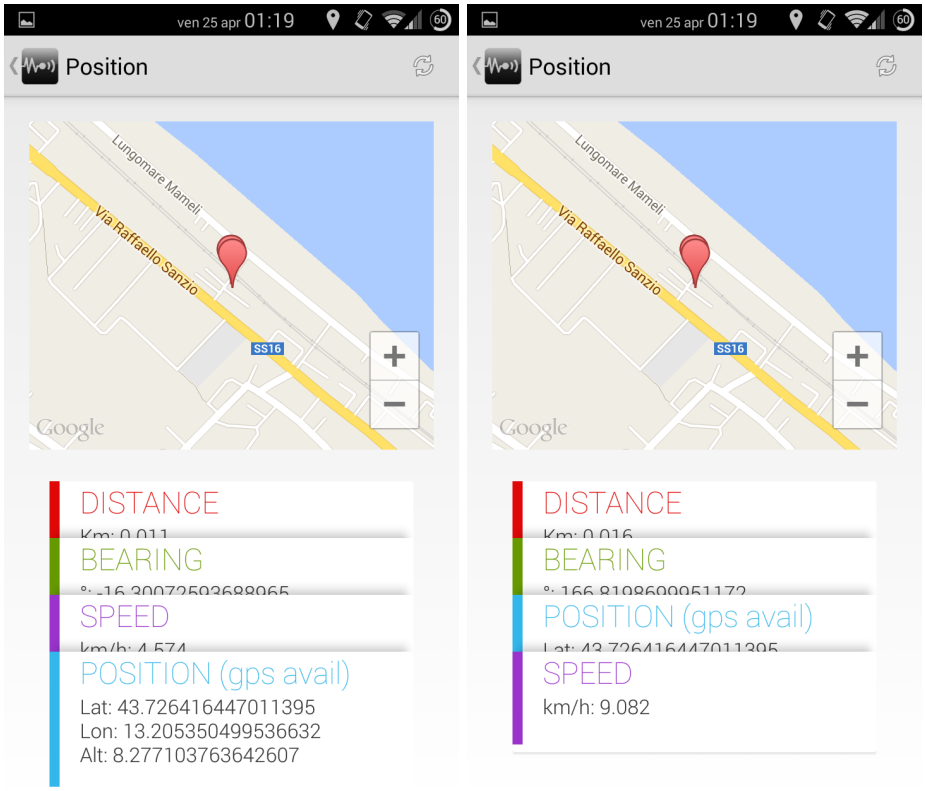

Figura 7.5: screenshot che mostra l'aspetto del tab position con le info derivate

Tab Acceleration

Il tab "Acceleration" consente di visualizzare i dati legati all'accelerometro mediante un grafico a due dimensioni, ovviamente solo nel caso in cui l'accelerometro sia disponibile e abilitato. A questo scopo abbiamo usufruito di una libreria esterna chiamata AndroidPlot che ha reso il nostro compito molto più semplice.

In questo caso l'AcceleratorFragment contiene quindi un androidplot.xy.XYPlot seguito da una card e da un'icona nel menu: il comportamento del pulsante è esatta-

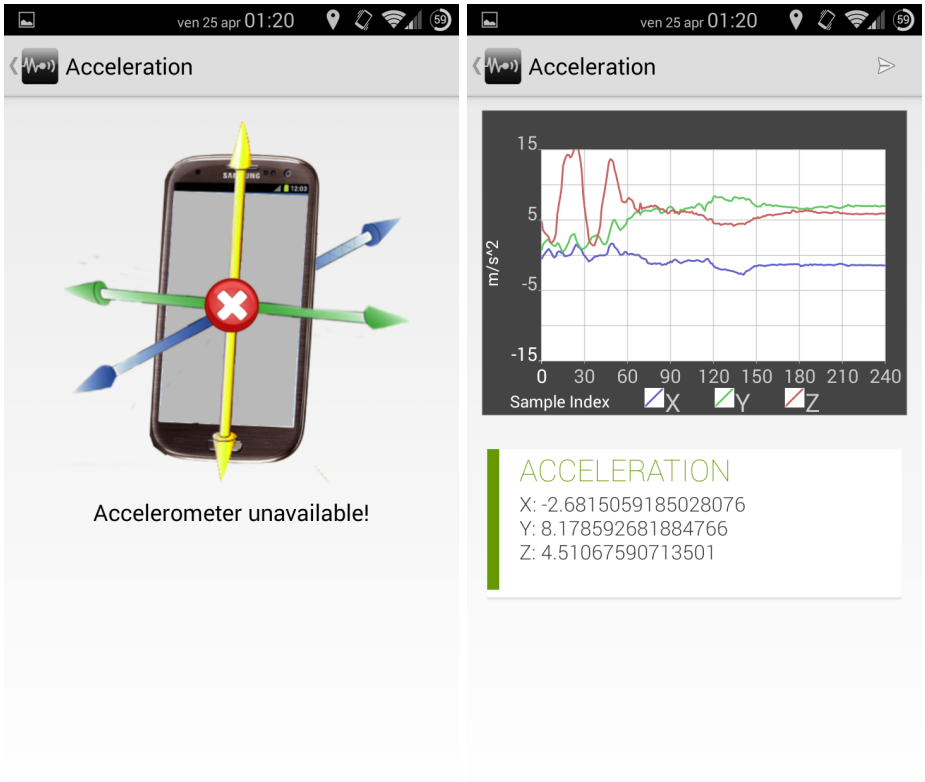

Figura 7.6: screenshot che mostra l'aspetto del tab acceleration

mente conforme a quanto visto per il tab precedente ma in questo caso ci avvarremo di una AccelerationInteraction. Nel grafico vengono mostrati con colori diversi gli andamenti dei valori nel tempo relativi alle 3 componenti dell'accelerazione: mentre l'unità di misura dell'ascissa è puramente indicativa in quanto rappresenta un indice, quella dell'ordinata sono i m/s^2.

Tab Rotation

Il tab "Rotation" è analogo in tutto e per tutto a quello dell'accelerazione, ma mostra ovviamente i dati relativi al vettore di rotazione del dispositivo. In questo caso viene utilizzata la RotationInteraction e l'unità di misura dell'ordinata del grafico è adimensionale.

Tab Network

Dopo le informazioni strettamente collegate alla posizione e al movimento del di-

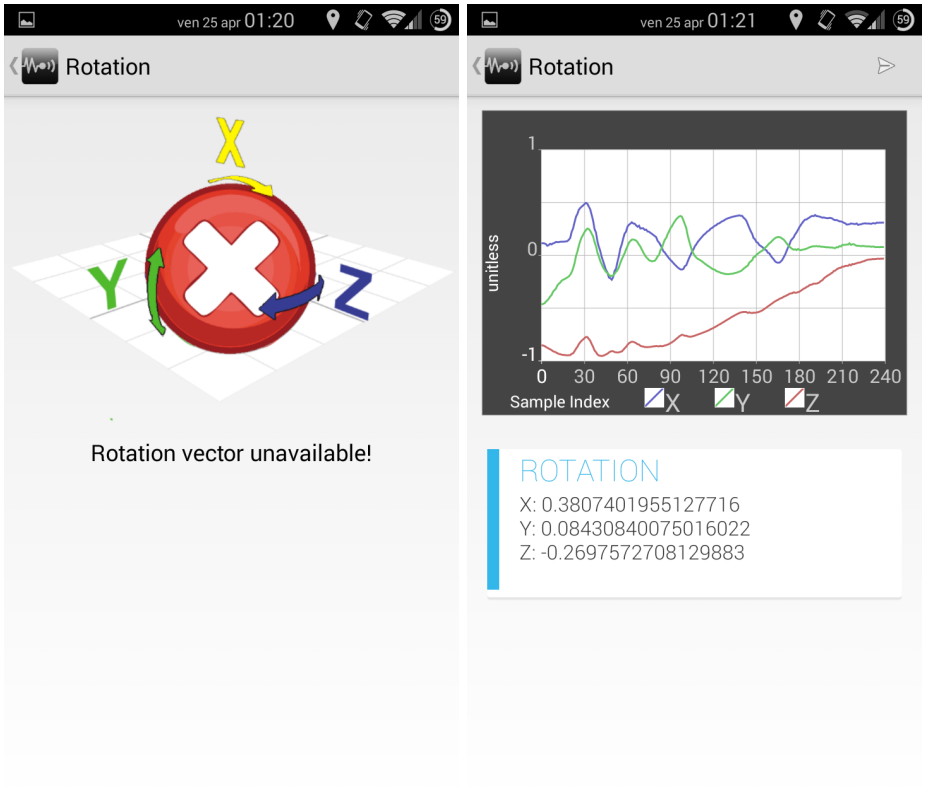

Figura 7.7: screenshot che mostra l'aspetto del tab rotation

spositivo, affrontiamo ora quello che ci viene messo a disposizione nel momento in cui effettuiamo una qualsiasi connessione alla rete. In questo caso il ConnectionFragment utilizza la ConnectionInteraction per ottenere i dati in questione, che come si vede dalle immagini seguenti variano a seconda della tipologia di connessione. Anche qui caso la pressione dell'icona del menu attiverà il meccanismo di aggiornamento della GUI e l'eventuale invio alla blackboard.

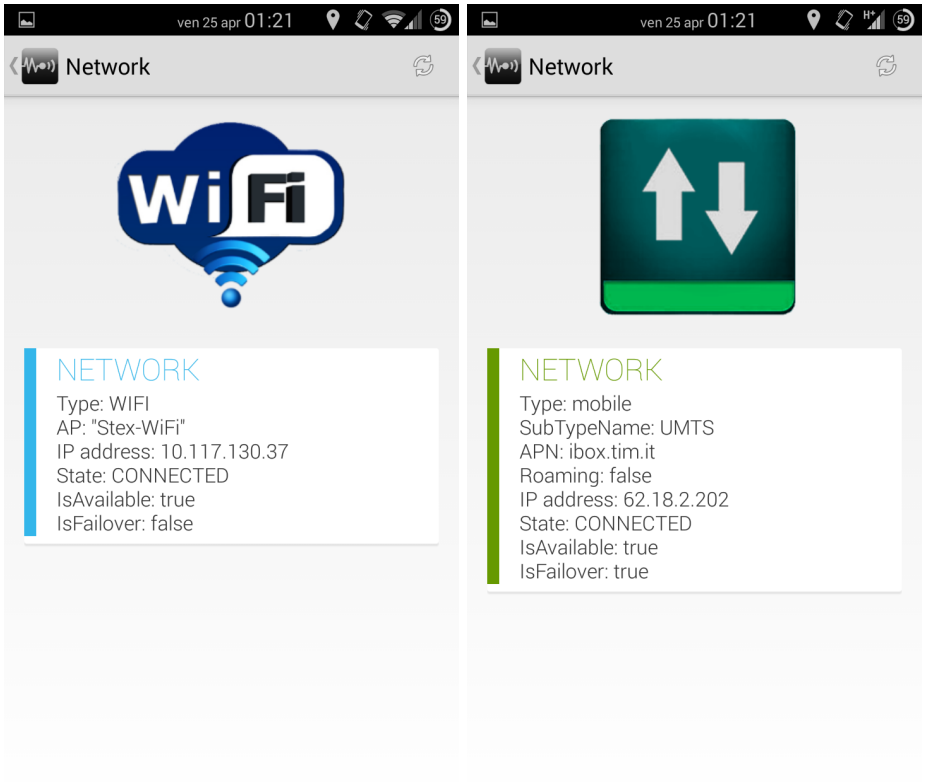

Figura 7.8: screenshot che mostra l'aspetto del tab network

Tab Activity Recognition

In questo tab cerchiamo di sfruttare al meglio una caratteristica messaci a disposizione dai Google Play Services ovvero l'Activity Recognition: essa cerca di individuare l'attività fisica corrente che il proprietario del dispositivo sta effettuando. In base all'intervallo di tempo settato, vengono inviate informazioni contenenti una lista di possibili attività abbinate alle rispettive percentuali di probabilità, per questo motivo abbiamo deciso di mostrare per ogni aggiornamento una scheda contenente le prime due tipologie riconosciute con la relativa percentuale.

Per avviare o terminare le richieste di aggiornamento abbiamo a disposizione due voci nel menu, "start" e "stop", il cui comportamento rispecchia ovviamente ciò che i due termini evocano. Con il passare del tempo si formerà una lista di schede che possiamo eliminare tramite uno swipe laterale oppure possiamo scorrere scrollando verticalmente.

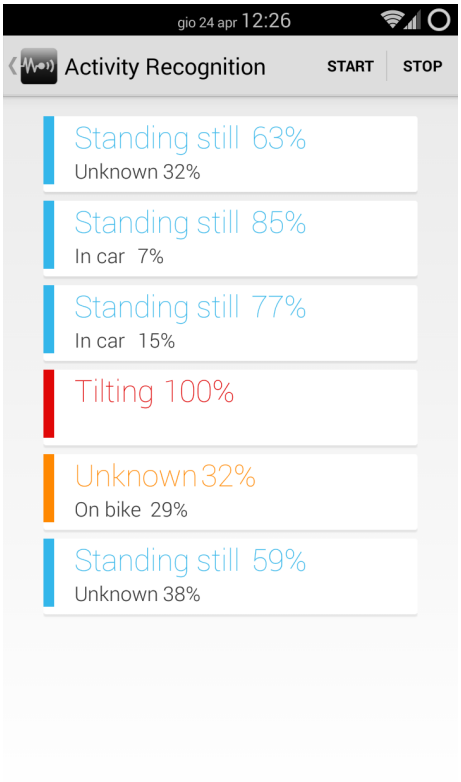

Figura 7.9: screenshot che mostra l'aspetto del tab activity recognition

Tab Game

In precedenza abbiamo mostrato in maniera piuttosto spartana quelli che sono i dati provenienti dall'accelerometro e quindi derivanti dai movimenti del dispositivo, gli smartphone vengono però ormai utilizzati anche come consolle e quindi ci sembrava una buona idea realizzare un semplice gioco che sfruttasse direttamente i dati a disposizione. Si tratta di una banale pallina che si muove all'interno dello schermo rimbalzando sui bordi e definendo i suoi spostamenti in base a quelli che l'utente impone al dispositivo attraverso movimenti o tocchi dello schermo.

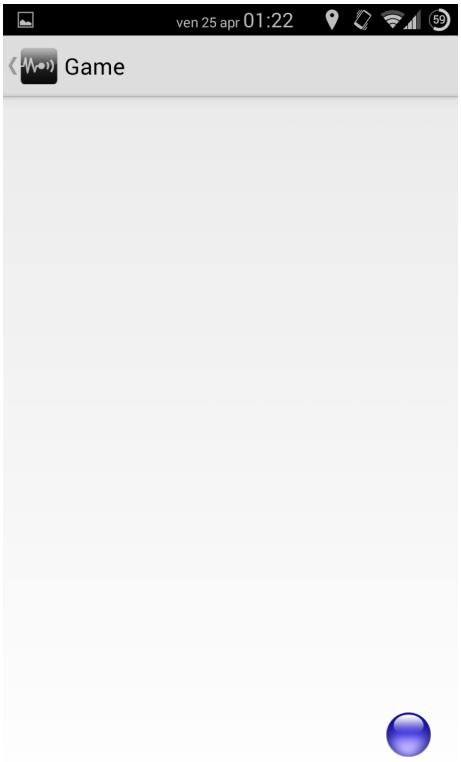

Figura 7.10: screenshot che mostra l'aspetto del tab game

L'ultimo tab ha una struttura differente dai precedenti, i quali avevano comunque uno scheletro affine, in questo caso il layout non viene definito attraverso dei file xml ma viene utilizzata una view descritta tramite normale codice Android: stiamo parlando dell'AnimationView che provvede alla creazione di un elemento Ball opportunamente definito nella rispettiva classe. Le due classi sopracitate sono responsabili del rendering della pallina all'interno dello schermo tenendo in considerazione diversi fattori che definiscono la velocità anche in base agli urti con le pareti definite dai bordi.

Oltre alla velocità definita dai valori proveniente dall'accelerometro, abbiamo la possibilità di impostare la posizione della pallina nel punto in cui tocchiamo lo schermo, o in alternativa,trascinarla attraverso uno swipe assegnandogli una velocità direttamente proporzionale alla distanza dei punti in cui si posato e rialzato il dito dallo schermo.

7.2.4 DeviceAgent, Interaction, Activity Recognition Service

Dopo aver presentato tutta l'applicazione non ci resta che approfondire ciò che avviene fra l'altro agente non ancora citato e quelle che abbiamo in precedenza definito misteriosamente come Interaction. Ricordando che il DeviceAgent viene attivato solo nel caso della versione "online", ne consegue che ci sarà la necessità di individuare un destinatario per le informazioni: a tal proposito in fase di setup dell'agente in questione rimarrà in attesa di individuare l'indirizzo nel sistema dell'agente che mantiene la blackboard. In seguito i relativi fragment invocheranno la generazione di determinate informazioni passando come parametro l'Interaction specifica e, dopo aver controllato la disponibilità del componente hardware relativo, invieranno o meno quanto ottenuto dalle Interaction.

Con il termine Interaction indichiamo quella parte del sistema a cui abbiamo demandato l'interazione con il livello fisico del dispositivo e in cui vengono quindi sfruttate le API specifiche: abbiamo preferito creare l'oggetto base Interaction e poi far estendere tale classe a ciascuna classe specifica implementando solo i metodi di interesse. Sfuggono a questi meccanismi solamente i tab game e recognition activity, il primo per ovvi motivi, il secondo per ragioni dettate dal funzionamento del servizio offerto da Android. Tale servizio è incluso nei Location Services appartenenti ai Play Services, come primo passo dobbiamo quindi controllare che l'APK corrispondente sia stato installato, dobbiamo poi considerare che il meccanismo in questione si basa sull'invio di una richiesta proveniente da un client, il quale riceverà le informazioni tramite un PendingIntent. Il meccanismo di richiesta è un processo asincrono che si avvia nel momento in cui il client richiede una connessione, a sua volta il Location Service invocherà la onConnected() in cui abbiamo inserito la richiesta di aggiornamento (sincrona). Per gestire l'Intent che il LocationService invia ad ogni intervallo di aggiornamento, dobbiamo quindi definire un IntentService e il corrispettivo metodo OnHandleIntent(). Nel nostro caso il client è implementato all'interno della classe RecognitionInteraction, nella quale si verifica anche la presenza dei Play Services e si effettuano le connessioni; l'IntentService viene definito dal RecognitionIntent e la gestione dei risultati viene effettuata nel RecognitionFragment grazie ad un broadcastReceiver.

7.3 PC Host

Come abbiamo già detto, la connessione e quindi il passaggio alla seconda Activity dell'app Android in modalità online, è subordinato alla presenza di una piattaforma Jade già esistente ma non è affatto vincolato dalla presenza di agenti di nessun tipo. Per il corretto svolgimento delle interazioni è però necessario che almeno in seguito vengano avviati sia un BlackboardAgent che un OperatorAgent, ottenendo così una disposizione di questo tipo:

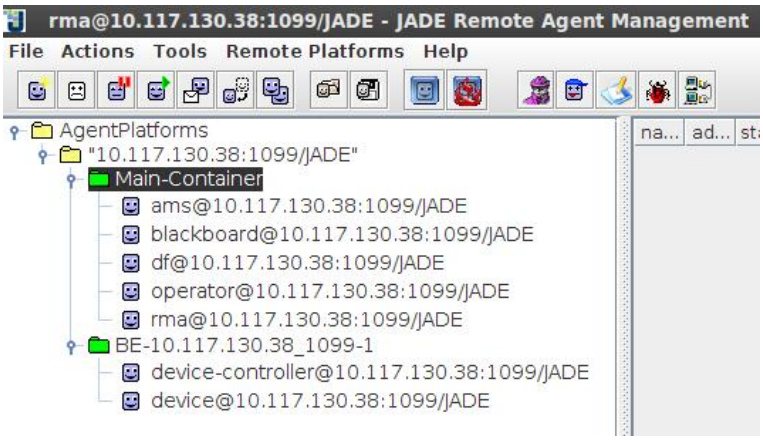

Figura 7.11: screenshot dell'rma che mostra gli agenti attivi

In particolare, un Blackboard agent è responsabile del mantenimento di quella struttura che abbiamo definito in sostituzione dei meccanismi a spazio di tuple, purtroppo assenti fra i costrutti offerti da Jade: tale struttura presenta una serie di HashMap che hanno come chiave il nome del dispositivo a cui sono associati i dati e come valore i diversi oggetti che rappresentano le informazioni spaziali generate dal device. La struttura, così come è pensata al momento, non prevede la storicizzazione delle informazioni ed ogni aggiornamento sovrascrive quindi il precedente. L'agente in questione, in fase di inizializzazione, provvede a pubblicizzare grazie al DF il servizio che mette a disposizione e in questo modo, sia il DeviceAgent che l'OperatorAgent sanno a chi devono rivolgersi per rendere complete le loro attività. In seguito a questa prima fase, esso provvederà a gestire la ricezione dei messaggi di aggiornamento dal dispositivo e risponderà alle richieste specifiche provenienti dall'OperatorAgent. L'operatore rappresenta una sorta di osservatore esterno che si occupa di richiedere e visualizzare i dati di interesse relativi ad uno specifico dispositivo: la volontà di non

appesantire il comportamente del DeviceAgent e il significato intrinseco della balckboard, fanno sì che le richieste vengano effettuate ricercando in tale struttura e non andando a interpellare il dispositivo in questione. Nonostante il lancio dell'agente, finchè non verrà registrato dal BlackboardAgent un servizio "blackboard", non verrà mostrata la seguente GUI di interazione.

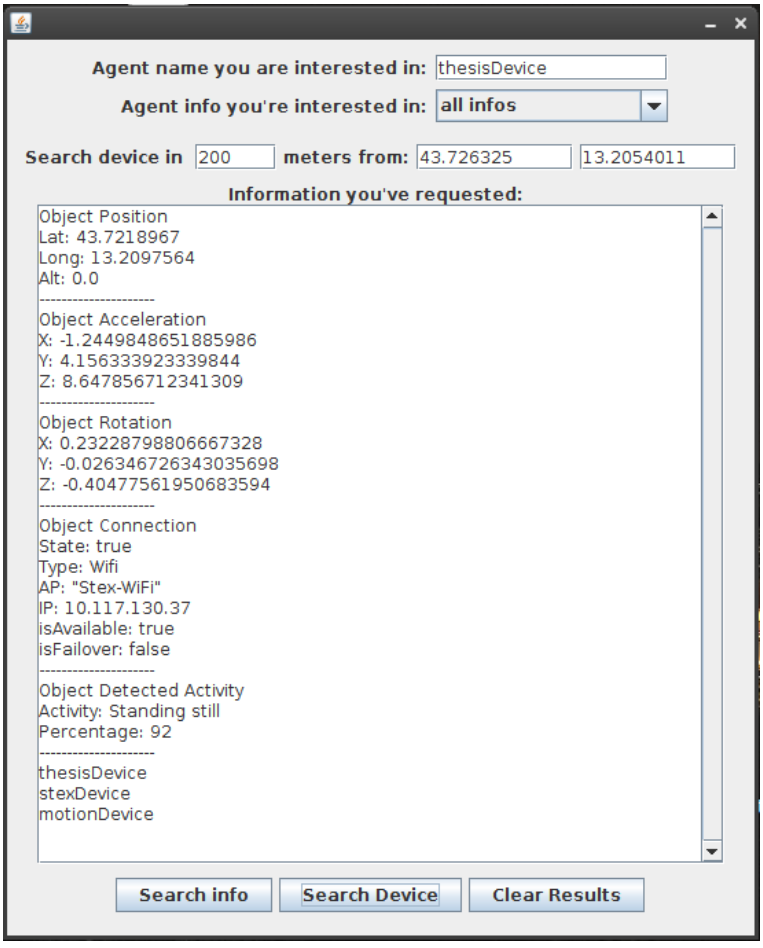

Figura 7.12: screenshot che mostra la gui dell'agente operator

Come si può osservare dall'immagine, attraverso dei campi di testo modificabili abbiamo la possibilità di inserire il nome del device a cui siamo interessati (lo stesso inserito nella prima Activity dell'app Android) e il tipo di informazione che necessitiamo (oppure tutte le info disponibili). Il pulsante "Search info" provvederà all'invio della richiesta specificata mentre "Clear results" ripulirà la textArea che mostra i risultati, in caso di richiesta su un device non presente verrà stampato il messaggio "no

info available". Abbiamo inoltre inserito una feature legata alla programmazione aggregata: ci viene infatti data la possibilità di ricercare tutti i dispositivi che si trovano ad una distanza massima da un punto specificato tramite coordinate geografiche, in questo caso sarà il pulsante "Search Device" a far partire la richiesta in questione.

Conclusioni e sviluppi futuri

Nel corso di questo testo abbiamo definito puntualmente i requisiti minimi che un dispositivo mobile deve avere per abilitare virtualmente qualunque tipo di computazione context-aware basandoci su quanto al momento disponibile sul mercato e su cosa viene effettivamente richiesto dagli scenari applicativi che coinvolgono lo spatial computing. Abbiamo in seguito analizzato il mondo della programmazione aggregata e visto cosa ci viene offerto dai linguaggi di computazione spaziale al momento presenti; dopo aver quindi ideato un'architettura astratta per un middleware con determinate caratteristiche, l'abbiamo mappata su un vero middleware già esistente e abbiamo sfruttato tutto ciò concretizzando il nostro lavoro attraverso un'applicazione multi-agente e distribuita. Ciò che è stato presentato è senza dubbio un primo prototipo abbastanza acerbo e limitato, che tiene conto ancora solo superficialmente dei dati spaziali e li utilizza come un qualsiasi altro tipo di dato senza sfruttarli a livelli non applicativi.

A questo punto si potrebbero quindi inserire determinati concetti menzionati soprattutto nei paragrafi 6.3.4 e 6.4, aumentando la specializzazione del nostro middleware soprattutto a livello infrastrutturale e limitando al minimo i dettagli implementativi in fase di sviluppo applicativo, possibilmente con l'introduzione di un framework Java/Jade-based per sfruttare API legate a determinati servizi. In particolare ci si potrebbe concentrare:

- sull'utilizzo dei dati spaziali a fini di identificazione e indirizzamento degli agenti;

- sull'inserimento di un vero e proprio spazio o addirittura centro di tuple che definisca una reale memoria condivisa data o control driven;

- sulla creazione concreta di strutture gerarchiche orizzontali e verticali legate alla topologia del sistema.

Rimanendo invece su quanto già realizzato, si potrebbero implementare alcuni operatori dei linguaggi visti al capitolo 3 per capire quale di essi può essere supportato dal nostro middleware ed eventualmente effettuare delle modifiche che permettano la loro integrazione. Potrebbe risultate inoltre significativo intraprendere una strada parallela che consenta di sviluppare concetti simili su piattaforme diverse, ad esempio analizzando un eventuale porting su IOS e capendo quali potrebbero essere le modifiche da intraprendere o le limitazioni imposte.

Bibliografia

INTRODUZIONE E INQUADRAMENTO DEL PROBLEMA

[1] Space-aware Coordination in ReSpecT, S. Mariani, A. Omicini

[2] Coordination for Situated MAS: Towards an Event-driven Architecture, S. Mariani, A. Omicini

[3] Event-Driven Programming for Situated MAS with ReSpecT Tuple Centres, S. Mariani, A. Omicini

STRUMENTI A DISPOSIZIONE PER LA COMPUTAZIONE

APPLE

[4] Inside the iPhone 5s

[5] How Apple's M7 Chip Makes the iPhone 5S the Ultimate Tracking Device

[6] The iPhone 5s Review

[7] pagina Wikipedia Apple M7

[8] iBeacon, la risposta di Apple all'NFC

[9] With iBeacon, Apple is going to dump on NFC and embrace the internet of things

[10] Core Location Framework Reference

[11] Core Motion Framework Reference

GOOGLE

[12] iPhone 5-Like Motion Processor for Any Mobile Device

[13] PNI Sentral

[14] pagina Wikipedia Bluetooth low energy

[15] Android Location API

[16] Android Sensors Overview

[17] Android Motion Sensors

[18] Android Position Sensors

ARCHITETTURA DI PROGRAMMAZIONE AGGREGATA

[19] Organizing the Aggregate: Languages for Spatial Computing, J. Beal, S. Dulman, K. Usbeck, M Viroli, N. Correll, April 2nd 2012

[20] Zephyr Introduces Revolutionary New PSM Training ECHO Solution

[21] GLM 100 C Professional

LINGUAGGI DI COMPUTAZIONE SPAZIALE

[22] Organizing the Aggregate: Languages for Spatial Computing, J. Beal, S. Dulman, K. Usbeck, M Viroli, N. Correll, April 2nd 2012

SCENARI APPLICATIVI

[23] Engineering Adaptive Service Ecosystems

[24] Pervasive Ecosystems: a Coordination Model based on Semantic Chemistry, M. Viroli, D. Pianini, S. Montagna

DEFINIZIONE MODELLO ARCHITETTURALE ASTRATTO

[25] Spatial Computing: the TOTA Approach, M. Mamei, F. Zambonelliranco

[26] Abstract software migration architecture towards agent middleware interoperability, J. Cucurull, B. J. Overeinder, M. A. Oey, J. Borrell, F. M.T. Brazier

[27] Agilla: A Mobile Agent Middleware for Self-Adaptive Wireless Sensor Networks, Chien-Liang Fok, Gruia-Catalin Roman, Chenyang Lu

[28] Agent middleware and standards: the keys for the success of agents, A. Poggi

MAPPING DELL'ARCHITETTURA ASTRATTA SU UN MIDDLEWARE GIÀ ESISTENTE

[29] Jade Home Page

[30] Developing Multi-Agent Systems with JADE

[31] Jade Programming for Beginners

[32] Jade Programmer's Guide

[33] Jade Programming for Android

[34] Jade Android Add-On Guide

[35] A General-purpose Programming Model & Technology for Developing Working Environments in MAS, A. Ricci, M. Viroli, A. Omicini

[36] Programming MAS with Artifacts, A. Ricci, M. Viroli, A. Omicini

[37] From Agents to Artifacts Back and Forth Operational and Doxastic Use of Artifacts in MAS, M.Piunti, A. Ricci

[38] Coordination as a service in Jade, Nicola Dellarocca