Alma Mater Studiorum · Università di Bologna

Campus di Cesena
Scuola di Ingegneria e Architettura

Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

Ingegnerizzazione di Sistemi Software basati su Schermi Adattativi Pervasivi

Elaborato in
Fondamenti di Informatica A

Relatore:
Chiar.mo Prof.
MIRKO VIROLI

Presentata da:
SIMONE GROTTI

Sessione II
Anno Accademico 2012-2013

Alla mia famiglia

Indice

Introduzione

> *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*[13]

Con queste parole Weiser sottolinea come le tecnologie più profonde siano quelle che scompaiono, si integrano talmente tanto nella vita vita quotidiana di ognuno di noi da scomparire, diventare invisibili ai nostri occhi. Proprio la trasparenza è una delle caratteristiche principali del Pervasive (o Ubiquitous) Computing, un modello a cui si fa riferimento parlando di scenari in cui la capacità computazionale permea l'ambiente ed è integrata con esso: in questo modo l'interazione tra persone e computer risulta facilitata e più naturale, poiché può avvenire attraverso modalità note alle persone come il comune utilizzo di oggetti quotidiani.

Secondo il paradigma del Pervasive Computing la capacità computazionale assume caratteristiche di ubiquità e viene integrata persino negli oggetti di uso più comune, quali ad esempio una tazza: a questa visione si è arrivati grazie ai progressi tecnologici nel campo dell'hardware e delle comunicazioni, che hanno portato ad avere dispositivi informatici sempre più piccoli, performanti e comunicazioni sempre più veloci.

In questo contesto si inserisce il lavoro presentato in questa tesi, che ha come obiettivo principale la realizzazione di un framework per sistemi software basati su schermi adattativi pervasivi: si vuole perciò fornire al programmatore un insieme di strumenti e concetti per programmare applicazioni per uno schermo capace di riconoscere non solo la presenza di osservatori davanti ad esso, ma anche se essi gli stanno porgendo attenzione, e di adattare i propri contenuti sulla base dell'identità dell'utente e della sua interazione con lo schermo, che deve avvenire secondo le linee dettate dai principi di interazione naturale. Il framework è stato realizzato per applicazioni Java che utilizzano Kinect come sensore e driver OpenNI per interagire con esso: attraverso questi strumenti sono perciò stati realizzati dei componenti software, sotto forma di librerie, che il programmatore può utilizzare e comporre

per raggiungere il proprio scopo, soddisfacendo così i requisiti principali di un buon framework, quali la riusabilità.

A conclusione del lavoro svolto viene presentata la realizzazione di un caso applicativo, calato nel contesto accademico, che mostra uno dei possibili utilizzi di uno schermo di questo genere: attraverso questa applicazione si mostra come l'utilizzo del framework realizzato velocizzi lo sviluppo e permetta di concentrarsi esclusivamente sul progetto della logica specifica per il contesto considerato. Inoltre si dimostra come sistemi di questo tipo possano suscitare interesse non solo in ambito accademico, ma anche in contesti del tutto separati da questo: data la versatilità con cui gli schermi possono adattarsi ai contenuti da mostrare, potrebbero suscitare l'interesse in ambiti quali la sociologia, la cultura o il marketing.

Struttura dei contenuti

Primo Capitolo

Vengono fornite le informazioni necessarie a contestualizzare il lavoro svolto: una definizione e una breve introduzione al concetto di Pervasive Computing e una breve panoramica sugli strumenti utilizzati, cioè il sensore Kinect e i driver OpenNI e NITE per interfacciarsi ad esso.

Secondo Capitolo

Viene fornita una descrizione della fase di progetto del framework realizzato: a partire dall'analisi del prototipo realizzato in [4] per giungere alla descrizione della struttura proposta per il framework e delle funzionalità che le librerie in esso contenute si propongono di offrire, fornendone le principali considerazioni progettuali svolte durante il lavoro.

Terzo Capitolo

Si descrivono le librerie realizzate nei termini della loro implementazione, in modo tale da mostrare i concetti e le potenzialità che offrono ai programmatori che le utilizzeranno: per rendere la descrizione più efficace, essa viene fatta in riferimento a dei semplici casi applicativi realizzati ad hoc come una sorta di guida all'uso delle librerie. Vengono perciò presentate le classi principali di ogni componente software realizzato, scendendo nei dettagli implementativi. Inoltre vengono date le informazioni necessarie riguardo al deployment del framework, comprendenti anche tutte quelle riguardanti

gli strumenti necessari per ottenere un ambiente operativo per lavorare con questo tipo di sistema.

Quarto Capitolo

In questo capitolo viene presentato il caso applicativo realizzato utilizzando il framework presentato, uno schermo che permette l'identificazione di due utenti e che adatta i suoi contenuti a seconda degli utenti che gli porgono attenzione, sia in termini di progetto che di implementazione, sottolineando in quali fasi dello sviluppo il framework ha facilitato il lavoro. Viene inoltre presentato, tramite degli screenshot dell'applicazione in esecuzione, il funzionamento di questo prototipo.

Quinto Capitolo

Vengono presentate le conclusioni e gli sviluppi futuri per il lavoro svolto, soprattutto nell'ottica di estensione del framework realizzato.

Capitolo 1

Pervasive Computing, Kinect e OpenNI

In questo capitolo vengono fornite tutte le informazioni necessarie per contestualizzare il lavoro svolto in questa tesi: partendo da una sintetica descrizione dei concetti introdotti con il *pervasive computing* si passa alla descrizione di uno degli strumenti principali utilizzati in questo lavoro, il Kinect e driver OpenNI e NITE per farlo funzionare.

1.1 Pervasive Computing

Il termine *Ubiquitous Computing* venne utilizzato per la prima volta da Mark Weiser, direttore scientifico delle ricerche tecnologiche allo Xerox PARC, nel 1991 nel suo famoso articolo [13] in cui annunciava un cambiamento nel modo di concepire l'elaborazione automatica e descriveva scenari in cui i computer, onnipresenti, entravano sempre più a far parte della vita quotidiana delle persone. [1]

I computer ed il computing in generale, infatti, si stanno dirigendo verso nuovi paradigmi: dai grandi e costosi mainframe, i computer degli anni '60 per cui parlava di paradigma *many people per computer*, poiché molti utenti condividevano una sola macchina si è arrivati al paradigma *one person per computer*, in cui ogni persona può disporre di un proprio calcolatore, grazie al progresso tecnologico che ha consentito la realizzazione dei personal computer, avvicinando sempre più persone all'utilizzo del computer. Nell'ultimo decennio la diffusione di laptop, Personal Digital Assistant(PDA), telefoni cellulari multifunzione, dispositivi portatili dotati di microprocessori e di capacità di immagazzinare dati ha mutato ulteriormente il rapporto

[1] Questa sezione è una rielaborazione personale dei contenuti reperiti in [11], [12] e [4]

uomo-computer, aprendo le porte all'era dei *many computers per person*: tanti elaboratori per una singola persona.

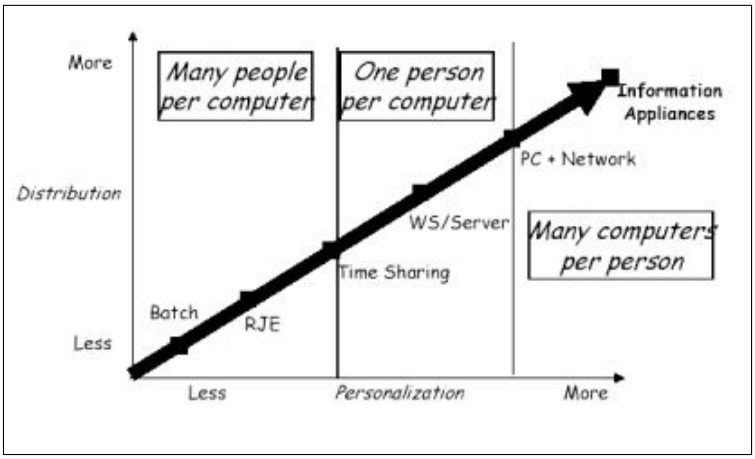

Figura 1.1: Evoluzioni dei computer

Il proliferare di questi dispositivi intelligenti, sempre più piccoli e meno costosi, insieme con i progressi delle tecnologie di comunicazione, ha indotto Mark Weiser ad immaginare un prossimo futuro nel quale i computer entrano a far parte integrante di ogni oggetto della realtà quotidiana, favorendo la comunicazione e l'elaborazione di informazioni in maniera naturale: *everywhere, all the time.*

Un sistema di Ubiquitous Computing è caratterizzato da due attributi fondamentali:

- **Ubiquità**: l'interazione con il sistema è disponibile dovunque l'utente ne abbia bisogno;
- **Trasparenza**: il sistema non è intrusivo ed è integrato negli ambienti della vita quotidiana.

In accordo con questa visione è possibile identificare due dimensioni che forniscono una più chiara definizione per i sistemi Ubiquitous ed esprimono le relazioni esistenti con le altre aree di ricerca emergenti:

- **Mobilità dell'utente**: esprime la libertà che l'utente ha di muoversi quando interagisce con il sistema;
- **Trasparenza di interfaccia**: riflette lo sforzo consapevole e l'attenzione che il sistema richiede all'utente, sia per operare su di esso che per percepirne i suoi output.

In quest'ottica l'Ubiquitous Computing mira alla realizzazione di un mondo composto da ambienti dotati di capacità computazionali e comunicative, talmente integrati con l'utente da diventare una "tecnologia che sparisce", utilizzabile in maniera trasparente ed inconscia. Non una realtà virtuale, in cui le persone sono inserite in un mondo generato dai computer, ma piuttosto una virtualità reale che porta i computer a vivere nel mondo reale, insieme con le persone.

Tuttavia, gli scenari dipinti da Weiser 13 anni fa erano anacronistici; la tecnologia hardware necessaria per la loro realizzazione semplicemente non esisteva. E così, i tentativi compiuti allo Xerox PARC fallirono. Diversi anni dopo il fallimento di questi tentativi, i recenti sviluppi tecnologici hanno dato nuovo impulso alle ricerche sull'Ubiquitous Computing, oggi chiamato anche col nome di *Pervasive Computing*, per suggerire il carattere pervasivo con cui l'intelligenza elaborativa si diffonde e si manifesta negli oggetti che ci circondano.

1.1.1 Evoluzione verso il Pervasive Computing

Il Pervasive Computing può essere visto come una nuova forma di computazione altamente dinamica e disaggregata: gli utenti sono mobili e i servizi sono forniti da una serie di componenti distribuiti che collaborano tra loro. Le applicazioni necessarie per supportare queste nuove esigenze sono costituite, da un punto di vista architetturale, da moduli allocati in numerosi nodi della rete. In tal senso, il Pervasive Computing costituisce una nuova tappa nel percorso evolutivo dell'elaborazione e del calcolo distribuito.

Il *Distributed Computing* mira a ripartire dati e capacità computazionali su componenti indipendenti, potenzialmente residenti su macchine diverse, che comunicano tra loro attraverso reti di interconnessione.

L'introduzione di vincoli e problematiche legate al concetto di mobilità ha però reso necessarie nuove soluzioni tecnologiche che hanno portato alla creazione di una nuova forma di computing chiamata *Mobile Computing*: in questi nuovi scenari di calcolo distribuito, non si hanno più nodi di rete fissi, con connessioni stabili e veloci, ma nodi costituiti da dispositivi mobili che accedono alla rete e la abbandonano continuamente ed in maniera del tutto imprevedibile, dotati di connessioni precarie, e in cui le limitate capacità di calcolo e di memoria, unite alle esigenze di risparmio energetico, rappresentano altre caratteristiche di cui tenere conto.

I sistemi di *Pervasive Computing* sono quindi a loro volta anche sistemi distribuiti e mobili: in questo paradigma vengono riprese le problematiche derivanti dai paradigmi precedenti, in qualche modo però amplificate a cau-

sa dei requisiti stringenti e dei particolari contesti definiti in questi nuovi ambienti.

1.1.2 Caratteristiche dei Sistemi Pervasivi

Nei paradigmi di ubiquitous computing, servizi ed informazioni sono virtualmente accessibili dovunque, in ogni istante attraverso qualsiasi dispositivo, ma considerazioni di carattere amministrativo, territoriale e culturale ci inducono a dividere il mondo in tanti domini, piuttosto che vederlo come un unico enorme sistema. Ognuno di questi differenti ambienti pervasivi è contraddistinto da componenti o unità software che implementano astrazioni di servizi, clienti, risorse o applicazioni ed, in generale, da un'infrastruttura fissa e da una serie di elementi mobili che, in maniera del tutto impredicibile, entrano a far parte del sistema e lo abbandonano continuamente, talvolta migrando fra i diversi domini. Da queste considerazioni emergono le caratteristiche principali che un sistema pervasivo deve possedere, descritte qui di seguito.

Context Information e run-time adaption

Una prerogativa di un sistema pervasivo è la capacità di ottenere informazioni sugli utenti e sullo stato dell'ambiente, quali la posizione e l'identità dei singoli utenti e la disponibilità delle risorse.

Le informazioni di contesto vengono ottenute dalla collezione dei dati grezzi ricavati da una moltitudine di sensori, cercando di mantenere la scalabilità e garantire la sicurezza delle informazioni e la privacy. In base alle informazioni raccolte, le applicazioni possono adattare il loro comportamento in maniera diversa a seconda dei casi: i due tipi di adattamento principali sono quello *funzionale* e quello *strutturale*.

Una percezione dell'ambiente ragionevolmente accurata risulta quindi necessaria, insieme a meccanismi per il rilevamento e la correzione di informazioni di contesto inattendibili o contrastanti.

Task Recognition e Pro-Activity

Diversamente dai paradigmi di computing convenzionali in cui il comportamento del computer è principalmente composto di risposte all'interazione con l'utente, il pervasive computing mira alla realizzazione di un modello in cui i dispositivi sono parte attiva nell'interazione con l'utente: a partire dalle informazioni ricavate sul contesto in cui è inserito, il sistema pervasivo dovrebbe essere quindi in grado di riconoscere le azioni dell'utente e

guidarlo nell'attività, possibilmente capendone anche le intenzioni. Per questo i modelli che tengono conto dell'esperienza e del passato rappresentano un importante strumento per la caratterizzazione di un sistema pervasivo, favorendo la *pro-activity* del sistema: consentono di determinare infatti, in accordo con i precedenti comportamenti dell'utente, le azioni ottimali da eseguire in determinate situazioni.

Resource Abstraction e Discovery

Le risorse di un sistema pervasivo dovrebbero essere rappresentate in maniera astratta in modo da poter essere selezionate in base a requisiti di tipo generale oltre che specifico.

Dovrebbero essere previsti anche dei meccanismi per scoprire, interrogare ed interagire con le risorse nell'ambiente, per consentire l'introduzione di nuovi componenti senza onerose operazioni di configurazione e di riconfigurazione dei componenti esistenti.

Eterogeneity e Service UI Adaption

Un ambiente pervasivo è caratterizzato da una moltitudine di dispositivi eterogenei, la cui riduzione delle dimensioni comporta una crescita del loro numero ed una intensificazione delle interazioni uomo macchina.

Gli scenari di computing pervasivo evidenziano che distribuire ed installare servizi per ogni class e famiglia, specialmente in un'area geografica molto estesa, come avveniva tradizionalmente, diventa ingestibile. Risulta quindi importante che i servizi forniscano agli utenti interfacce che possano adattarsi alle caratteristiche del dispositivo client, senza stravolgere le proprie funzionalità.

Security e Privacy

Un sistema pervasivo generalmente gestisce grosse quantità di informazioni, molte delle quali acquisite tramite sensori e riguardanti gli utenti, dal cui punto di vista è desiderabile che vengano rispettati i principi di privacy e che sia garantita la sicurezza di queste informazioni. In molte situazioni però una certa perdita di *privacy* può essere tollerata, come ad esempio in situazioni di pericolo.

Per aggiungere sicurezza alle informazioni, i servizi nell'ambiente non dovrebbero consentire accessi non autorizzati e quindi indesiderati.

Fault Tolerance e Scalability

Gli ambienti pervasivi costituiscono sistemi perennemente attivi, pertanto un componente che subisce un guasto non deve compromettere il funzionamento generale dell'intero sistema, né richiedere una complessa tecnica di gestione. I componenti che cadono in errore dovrebbero automaticamente ripartire, laddove possibile, magari adoperando, ad esempio, memorie di stato persistenti che consentano di effettuare *resume* rapidi ed efficaci.

Gli ambienti pervasivi sono inoltre caratterizzati da una forte dinamicità: dispositivi possono aggiungersi all'ambiente ed abbandonarlo in qualsiasi momento, alcuni servizi possono cadere e altrettanti nuovi possono arrivare, gli stessi utenti possono entrare ed uscire dall'ambiente secondo la propria volontà. Il sistema deve quindi garantire scalabilità, ossia essere in grado di gestire e assicurare il suo funzionamento anche in seguito all'aggiunta di componenti; allo stesso tempo, i nuovi dispositivi e servizi introdotti nell'ambiente non devono interferire con quelli esistenti.

1.2 Kinect

Microsoft Kinect (inizialmente conosciuto con il nome *Project Natal*), è un accessorio per Xbox 360 sensibile al movimento del corpo umano, che permette al giocatore di interagire con la console senza l'utilizzo di alcun controller.

Figura 1.2: Microsoft Kinect

Sebbene fosse stato inizialmente distribuito esclusivamente per Xbox 360, a partire dal 1 febbraio 2012 Microsoft ha reso disponibile una versione della periferica per i PC dotati del sistema operativo Windows 7 e Windows 8. Inoltre, nell'estate 2011 sono stati rilasciati i driver ufficiali per poter utilizzare Kinect nel proprio Personal Computer, favorendo lo sviluppo di varie applicazioni tra il mondo degli sviluppatori di software. [2]

1.2.1 Storia

Kinect è stato annunciato al pubblico il 1 giugno 2009 durante la conferenza stampa di Microsoft all'E3 2009 (Electronic Entertainment Expo) con il nome Project Natal, poi rinominato Kinect alla presentazione ufficiale all'E3 2010.

L'hardware di Kinect si basa su tecnologie della 3DV, una compagnia israeliana specializzata in tecnologie di riconoscimento dei movimenti tramite videocamere digitali che Microsoft ha prima finanziato e poi acquisito nel 2009, e sul lavoro della israeliana PrimeSense, che ha poi dato in licenza la tecnologia a Microsoft. Il software di Kinect è stato invece sviluppato internamente dai Microsoft Game Studios e, più precisamente, dai programmatori della Rare, la quale ha dovuto cancellare altri progetti per dedicarsi interamente alla periferica.

1.2.2 Campi di utilizzo

Kinect è uno strumento nato come componente aggiuntivo per la console XBOX 360, quindi il contesto principale rimane quello dei videogiochi. Alcuni esempi in commercio che utilizzano Kinect come unico controller sono Kinect Adventures, Kinect Animals ed il gioco di ballo Dance Central.

Il costo relativamente basso, insieme alle funizionalità di body-tracking che il dispositivo offre, ha fatto incuriosire molti sviluppatori software: dopo qualche mese infatti il Web si è popolato di una moltitudine di applicazioni non strettamente legate al contesto dei videogames. Tra queste si possono citare programmi di visualizzazione di immagini, di riconoscimento del volto, plugin per software già esistenti e addirittura prototipi di riproduzione di una persona attraverso l'utilizzo di due Kinect.

Grazie a questo sensore è possibile eliminare mouse, tastiere e telecomandi: persone disabili potrebbero utilizzare questi dispositivi per abbattere numerose barriere che impediscono loro l'utilizzo della tecnologia.

[2] Questa sezione è una rielaborazione personale dei contenuti reperiti in [9], [8] e [4]

Il sensore può essere utilizzato anche nell'ambito della robotica utilizzando ad esempio la visione artificiale per il movimento degli automi, oppure per far volare un elicottero o per far muovere un piccolo veicolo, evitando ostacoli mediante la creazione di una mappa 3D dell'ambiente. Il dispositivo permette anche di risparmiare enormi budget per la realizzazione un sistema di motion capture.

Infine potrebbero essere realizzate anche applicazioni anche per l'intrattenimento e nel campo della medicina.

1.2.3 Hardware

Kinect è dotato di telecamera RGB, sensore di profondità a raggi infrarossi composto da un proiettore a infrarossi e da una telecamera sensibile alla stessa banda. La telecamera RGB ha una risoluzione di 640 x 480 pixel, mentre quella a infrarossi usa una matrice di 320 x 240 pixel.

Figura 1.3: Microsoft Kinect: componenti interni

È presente anche di un insieme di microfoni utilizzato dal sistema per la calibrazione dell'ambiente in cui ci si trova mediante l'analisi della riflessione del suono sulle pareti e sull'arredamento. In tal modo il rumore di fondo e i suoni del gioco vengono eliminati ed è possibile riconoscere correttamente i comandi vocali. La barra del Kinect è motorizzata lungo l'asse verticale e segue i movimenti dei giocatori, orientandosi nella posizione migliore per il riconoscimento dei movimenti.

I componenti, escluso il piccolo motore, non sono molto robusti: tutti gli ingranaggi sono in plastica e quindi facilmente usurabili.

Dato che le immagini vengono elaborate direttamente sul Kinect, Microsoft ha inserito nella periferica due schede ed una barra di supporto metallico in parallelo, separati da quattro distanziatori metallici. Sul lato è montata una piccola ventola per il raffreddamento che evita il surriscaldamento e il danneggiamento del dispositivo. Un altro componente particolare è la cella di Peltier posta tra l'IR e la barra metallica, che svolge il ruolo di sistema di raffreddamento. Tra la telecamera RGB e il proiettore IR è inoltre situato anche un piccolo LED di stato.

Il dispositivo è dotato di un array di quattro microfoni collegato alla scheda madre con un connettore a cavo unico, che permette al Kinect di ricevere i comandi vocali. I microfoni sono tutti e quattro orientati verso il basso, tre sono sul lato destro del dispositivo ed uno sul lato sinistro: la scelta di questo orientamento è stata decisa da Microsoft in quanto ritenuto quello ottimale per la raccolta del suono.

Per alimentare la periferica, Microsoft usa ben 12 Watt mentre le porte USB sono in grado di fornire in media 2,5 Watt di potenza, pertanto Kinect necessita anche di un cavo di alimentazione.

Inoltre è da notare che il cavo proprietario USB che la periferica usa per collegarsi alle normali console potrebbe non essere riconosciuto dai sistemi operativi su PC: è possibile quindi utilizzare un adattatore, di norma venduto insieme al Kinect, che Microsoft inserisce nella confezione per questioni di compatibilità con le prime versioni delle console XBOX 360.

1.2.4 Caratteristiche tecniche

Campo visivo (in gradi)	58°H, 45°V, 70°D
Risoluzione x/y (a 2 m dal sensore)	3 mm
Risoluzione z (a 2 m dal sensore)	10 mm
Range di lavoro	0.8 m – 3.5 m
Interfaccia USB	2.0
Consumo	2.25 W
Immagine di profondità	320 x 240 pixel
Immagine a colori RGB	640 x 480 pixel
Frame-rate	30 fps
Stream audio	4 canali 16 bit (fc 16KHz)

Tabella 1.1: Caratteristiche tecniche di Microsoft Kinect

Il sistema è teoricamente in grado di misurare le distanze all'interno di un area di 2 metri con un margine di errore di 1 cm. [3]

Nella tabella 1.1 vengono riassunte le principali caratteristiche tecniche di Kinect.

1.3 OpenNI e NITE

In questa sezione vengono brevemente presentati i driver utilizzati per interagire con Kinect: OpenNI e NITE

1.3.1 OpenNI

OpenNI (Open Natural Interaction) è un framework sotto licenza GNU GPL[4] indipendente dalle piattaforme di lavoro e multi-linguaggio che definisce le API per scrivere applicazioni che usano le Natural Interaction.

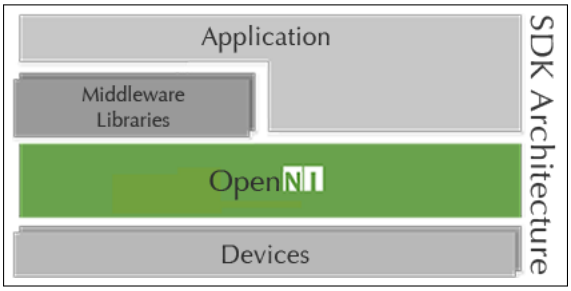

Figura 1.4: Architettura del framework OpenNI

L'intento di OpenNI è creare uno standard API per svolgere due compiti:

- comunicare con sensori visivi ed audio, per percepire figure e acquisire suoni;
- sviluppare funzionalità software (middleware) per analizzare e elaborare dati video ed audio registrati in una scena.

Il middleware permette di scrivere applicazioni che elaborano dati senza doversi preoccupare del sensore che li ha prodotti.

[3] Parametri di precisione forniti direttamente da Microsoft

[4] La GNU General Public License, comunemente indicata con l'acronimo GNU GPL o semplicemente GPL, è una licenza per software libero. `http://it.wikipedia.org/wiki/GNU_General_Public_License`

Il framework OpenNI è un livello astratto che fornisce l'interfaccia tra i dispositivi fisici e componenti middleware.

Il livello più alto rappresenta il software che fa uso di OpenNI implementando le Natural Interaction; quello centrale (middleware) rappresenta l'interfaccia OpenNI con le sue capacità di comunicare con i vari sensori e con le funzionalità disponibili (Skeleton Tracking, Hand Tracking, ecc.); mentre quello più basso è il livello hardware composto da tutti i sensori che possono inviare dati audio o visivi. Questi componenti sono indicati come moduli e quelli supportati dalle API sono, per i sensori:

- sensore 3D
- fotocamera RGB
- dispositivo audio (un microfono o un array di microfoni)

mentre per i componenti middleware:

- Analisi totale del corpo: è un componente software che elabora i dati sensoriali e genera informazioni relative al corpo come una struttura dati che descrive le articolazioni, il loro orientamento, il centro di massa del corpo e molto altro;
- Analisi della mano: è un componente software che elabora i dati sensoriali, individua la sagoma di una mano assegnando un punto alla posizione del palmo;
- Rilevamento gesto: è un componente software che identiica gesti predefiniti (Push, Wave, Circle) associandoli ad eventi.
- Analisi della scena: è un componente software che analizza l'immagine della scena al fine di produrre informazioni come individuare più persone nella scena, trovare le coordinate del piano, separare oggetti in primo piano da quelli sullo sfondo.

OpenNI ha rilasciato i driver e i propri codici sorgente per far sì che le sue librerie vengano implementate su più architetture possibili e su qualsiasi sistema operativo, in modo da accelerare l'introduzione di applicazioni di Natural Interaction sul mercato. Con l'uscita del Kinect la popolarità di OpenNI è nettamente aumentata, grazie anche alla creatività dei numerosi sviluppatori che lavorano con queste librerie.

Va sottolineato che OpenNI non implica necessariamente l'uso di Kinect, ma la facilità del framework di comunicare con qualsiasi sensore ha solo facilitato l'utilizzo del dispositivo Microsoft.

1.3.2 NITE

La libreria OpenNI restituisce dati a basso livello come mappe di profondità, mappe di colori, audio e altri tipi. NITE invece lavora ad un livello superiore: è un toolbox implementato sulle interfacce OpenNI che fornisce funzionalità aggiuntive e facilita la creazione di applicazioni di controllo basato sul movimento delle mani dell'utente e sullo scheletro. NITE contiene implementazioni per tutti i moduli della libreria OpenNI ed è formato dai seguenti livelli:

- OpenNI Modules: i moduli di OpenNI supportati da NITE sono Gesture Generator, Hands Generator e User Generator; inoltre supporta lo Skeleton Tracking;
- OpenNI Infrastructure: si veda 1.3.1.
- Control Management: riceve un insieme di punti e li indirizza verso il livello di controllo;
- Controls: ogni controllo riceve un insieme di punti e li traduce in un'azione specifica di tale controllo. Successivamente viene richiamata una funzione (*callback*) dall'applicazione che può cambiare il modulo di controllo attivo nel livello Control Management: questo definisce perciò il flusso dell'applicazione.

Un esempio di controllo è il *controllo del punto*: permette di registrare quando un punto viene creato, quando scompare e quando si muove.

I controlli sono comunque degli oggetti sempre in ascolto che attendono l'arrivo di nuovi dati ad ogni fotogramma: le funzioni di callback vengono invocate solo quando un determinato evento si verifica.

1.4 Funzionamento del sensore

Una delle funzionalità principali che Kinect offre è il riconoscimento degli utenti nella scena, prima raggiungibile solo a fronte di costi abbastanza sostenuti: l'obiettivo viene raggiunto grazie all'utilizzo di algoritmi sulla mappa di profondità. Per creare questa mappa il proiettore IR del Kinect getta un fascio di raggi infrarossi (che Microsoft ha assicurato non essere pericolosi per il corpo e per la vista): i raggi riflessi vengono catturati dalla telecamera ad infrarossi e con un algoritmo viene determinata la distanza dal sensore di tutti i punti.

Figura 1.5: Esempio di pattern di proiezione IR

Sulla base di queste informazioni è possibile assegnare una tonalità di grigio ad oggetti più o meno distanti, per rendere la mappa di profondità significativa ed intuitiva: infatti più il grigio è scuro più il punto è lontano.

L'immagine acquisita dal sensore viene fatta passare in diversi filtri in modo tale che il dispositivo possa capire cosa è una persona e cosa non lo è: l'intero sistema segue a questo proposito delle linee guida riguardanti la conformazione generale del corpo che permettono di non confondere gli oggetti con le persone in fase di calibrazione. Non tutte le persone hanno però la stessa conformazione fisica, inoltre spesso vengono utilizzati indumenti larghi o cappelli, e per questo vengono inseriti tra le linee guida degli algoritmi specifici di riconoscimento.

Quando la fase di calibrazione è terminata il dispositivo converte la parte dell'immagine relativa all'identificazione del corpo in uno scheletro che nella fase di tracking permette il movimento delle articolazioni, escluse, attualmente, quelle delle dita: l'intero sistema lavora a 30 fps ed ha 200 pose comuni per lo scheletro precaricate.

Nel caso l'utente faccia un movimento che impedisca alla telecamera di riconoscere il gesto fatto, l'algoritmo utilizza la posa tra quelle presenti che più si adatta al caso per non perdere il tracciamento dell'utente.

Un'altra funzionalità di interesse del sensore è il riconoscimento vocale, reso possibile grazie all'array di quattro microfoni inseriti nel device.

Il sottosistema composto dai microfoni ha come obbiettivo quello di essere sensibile al riconoscimento delle voci fino a 10 metri di distanza cercando di ignorare rumori ambientali. La larghezza del dispositivo Kinect è dovuta proprio al sistema di microfoni, per il quale Microsoft ha effettuato test in 250 abitazioni utilizzando 16 microfoni disposti in modo differente.

La soluzione ottimale trovata è stata l'array di quattro microfoni rivolti verso il basso, in modo da mantenere pulita la parte anteriore della periferica:

esso funziona meglio nel raccogliere le voci a distanza, ma necessita di qualche piccolo aiuto nel suo funzionamento. C'è a questo proposito un'unità di elaborazione a bordo del Kinect che toglie il rumore che si crea in prossimità dei sistemi surround 5.1, mentre un secondo sistema software Beam Forming agisce con la telecamera per capire dove si sta creando una possibile fonte di suoni intorno all'utente. Questo permette di aiutare il Kinect a capire quando non è l'utente a parlare ma altre persone intorno a lui. Il sistema di riconoscimento vocale ha un modello acustico per ogni singolo paese che comprende anche diversi dialetti regionali. I microfoni sono in ascolto in ogni momento rendendo il sistema Kinect open-mic.

L'ultima funzionalità importante qui descritta è quella offerta dal motore: il suo inserimento in questo tipo di sensore è dovuto alle necessità di calibrazione nelle diverse abitazioni europee, asiatiche ed americane: per Microsoft la telecamera doveva essere in grado di muoversi in su ed in giù per calibrare ogni singolo spazio, effettuando movimenti di circa 30 gradi. Esso fornisce anche la possibilità di interagire con lo zoom per la fotocamera, che permette di espandere lo spazio visivo: questa particolare funzionalità è stata progettata per la video chat di Kinect, in modo che se più utenti sono nella scena ed uno viene tagliato il motore gestisce in automatico lo zoom per far entrare tutti i partecipanti della conversazione sullo schermo.

Capitolo 2

Progetto del framework

Per *framework* si può intendere in generale un sistema software che cattura l'esperienza di risoluzione di problemi generali ricorrenti in forma di algoritmi riusabili, componenti software ed architetture estendibili. Esso pone a fattor comune le parti riusabili di un progetto e di una implementazione, separando la parte generica da quella specifica di un'applicazione e propone uno stile architetturale ([10]).

Uno degli scopi principali di un *framework* è quello di facilitare lo sviluppo di nuove applicazioni software fornendo una struttura su cui il programmatore può appoggiarsi, in modo da ridurre così per ogni applicazione la quantità di codice scritta e il tempo impiegato per la realizzazione. Proprio questo è stato il principio guida per il progetto del framework realizzato in questa tesi, costituito da un insieme di librerie scritte in Java che forniscono dei componenti software riusabili e componibili in modo flessibile per la programmazione di sistemi software basati su schermi pervasivi adattativi.

In questo capitolo ci si concentra sulla struttura del framework realizzato, dando una descrizione a livello concettuale delle funzionalità che ogni libreria realizzata offre, oltre alle informazioni principali riguardanti il loro progetto.

2.1 Struttura del framework

Il progetto realizzato si basa sull'analisi del lavoro svolto in [4], in cui viene presentato un prototipo di schermo adattativo pervasivo realizzato in Java con il supporto di un Kinect e dei driver OpenNI per interagire con esso.

2.1.1 Analisi del prototipo

A partire dal funzionamento e dalla struttura di questo prototipo sono stati individuati gli elementi principali che lo compongono, che ad un alto livello di astrazione possono essere riepilogati in:

- Riconoscimento delle persone di fronte allo schermo e del loro livello di attenzione rispetto ad esso;
- Identificazione degli osservatori;
- Interazione naturale tra persona e schermo.

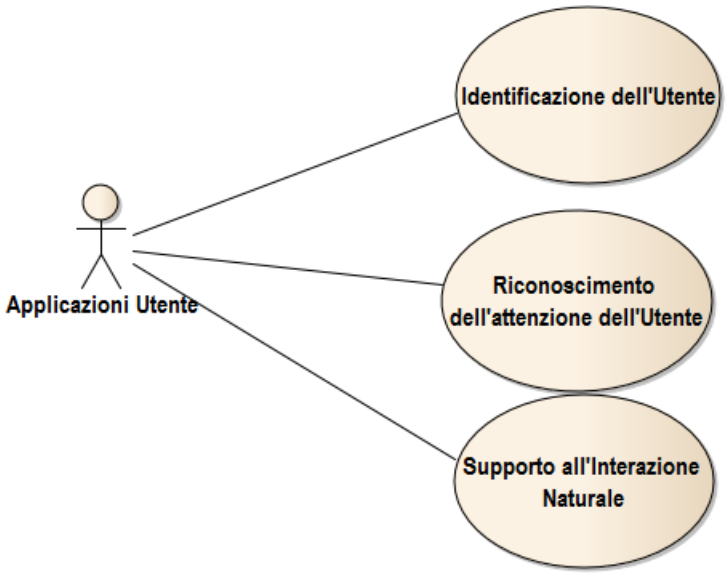

Figura 2.1: Diagramma degli Use Case per il framework

Queste possono anche essere viste come le funzionalità principali a cui una applicazione software per questo tipo di sistemi è interessata, e proprio in questo senso le modella il diagramma UML in figura 2.1.

Inoltre è risultato evidente dall'analisi del prototipo che le interazioni con i driver per comunicare con il sensore aumentano le righe di codice da scrivere, peggiorando così la leggibilità dello stesso e la sua pulizia: ad esempio anche solo inizializzare le funzionalità di OpenNI per ricavare i dati dal sensore si traduce in svariate righe di codice e nel relativo sforzo di scriverle, non facendo focalizzare pienamente il programmatore sulla logica dell'applicazione, quel che veramente conta.

Dal prototipo sono emersi anche svariati spunti per aumentare la configurabilità e la personalizzabilità di alcuni dei componenti mostrati: ad esempio la possibilità di specificare i parametri secondo cui un utente viene considerato attento, che possono variare da applicazione ad applicazione. A livello logico, il prototipo analizzato si appoggia direttamente alle librerie fornite da OpenNI, che rappresentano perciò uno strato intermedio di software che l'applicazione utilizza per ricavare i dati dal sensore: la situazione può essere meglio riassunta dalla figura 2.2.

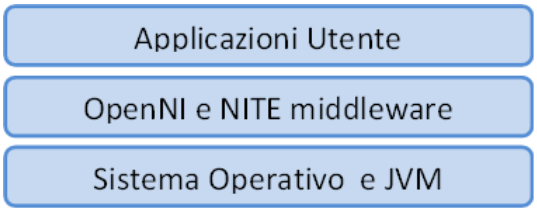

Figura 2.2: Livelli logici per il prototipo analizzato

In sintesi, dall'analisi del prototipo sono emersi diversi requisiti per il progetto del framework, che possono essere riassunti in:

- offrire una serie di componenti software riusabili e componibili a scelta del programmatore per creare svariate applicazioni nel campo considerato;
- fattorizzare le funzionalità a comuni a tutte le applicazioni (quali ad esempio quelle di inizializzazione del sensore) per diminuire sensibilmente il codice da scrivere a carico del programmatore;
- fornire al programmatore gli elementi principali necessari alla programmazione di uno schermo adattativo pervasivo;
- dare la possibilità di personalizzare e configurare, quando possibile, alcune delle funzionalità in modo da rispondere meglio alle esigenze della particolare logica dell'applicazione;
- essere facilmente estendibile, per permettere di aggiungere in futuro nuovi pezzi in modo da aumentarne le potenzialità espressive;
- essere *human readable*, ossia comprensibile in breve tempo da coloro che intendono usarlo: questo sottintende una buona documentazione delle funzionalità offerte, che riduce sensibilmente il tempo con cui un nuovo utilizzatore riesce a rendersi operativo con il framework.

Una volta realizzato il framework, le applicazioni utente non poggeranno più direttamente sulle librerie fornite da OpenNI (o, nel caso dell'identificazione su quella fornita da ZXing per i codici QR), ma sul framework, che si interpone così come nuovo livello logico tra le applicazioni e le librerie prima citate.

Figura 2.3: Livelli logici dopo la realizzazione del framework

2.1.2 Soluzione proposta

La soluzione proposta ed utilizzata per la realizzazione si articola in un insieme di quattro librerie, ognuna delle quali contiene un insieme di funzionalità strettamente legate ad uno dei componenti principali emersi durante la fase di analisi. Infatti ognuna di esse si concentra separatamente su uno degli aspetti fondamentali emersi per questo tipo di applicazioni, in modo tale da permettere al programmatore di scegliere quali librerie utilizzare a seconda delle sue esigenze.

Dopo aver deciso la struttura del framework, e quindi la sua divisione in librerie, si passa ad una descrizione ad alto livello di astrazione delle funzionalità che ognuna di esse ha il compito di offrire:

- **Libreria per l'interfacciamento con il sensore:** deve fornire le funzionalità di base con cui prelevare e manipolare i dati provenienti dal sensore, come ad esempio la visualizzazione delle immagini riprese dal sensore oppure il tracciamento dello scheletro di un utente presente nella scena. Anche le funzioni di inizializzazione del sensore, e quindi l'interfaccia con i suoi driver, fanno parte dell'insieme di funzionalità offerte da questa libreria. Rappresenta un pezzo molto importante del framework in quanto alcune delle altre librerie realizzate si affidano a questa per quanto riguarda l'interazione con il sensore.

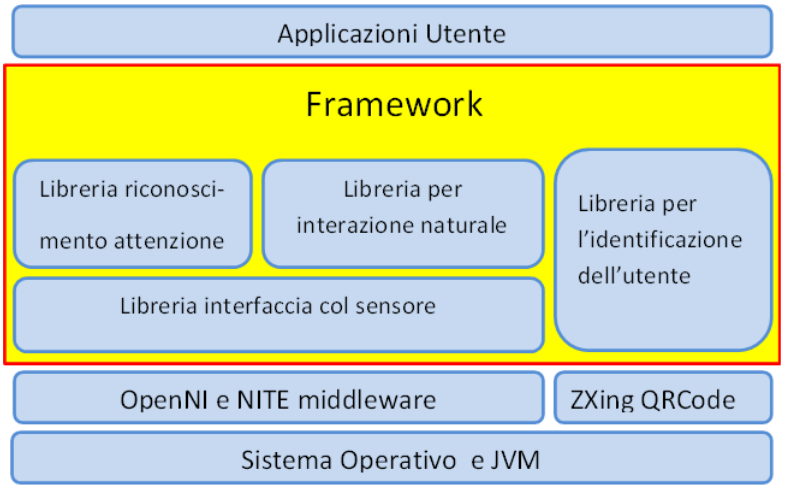

Figura 2.4: Struttura del framework e sua articolazione in librerie

- **Libreria per il riconoscimento dell'attenzione:** utilizza la libreria addetta alle funzionalità di base per ricavare i dati sulla scena ripresa dal sensore e li analizza per calcolare il livello di attenzione degli utenti presenti: deve fornire la possibilità di configurare i parametri secondo cui un utente è considerato attento e quindi la possibilità di sapere se in un determinato momento un certo utente sta prestando attenzione allo schermo o meno.

- **Libreria per l'interazione naturale:** utilizza la libreria addetta alle funzionalità di base per ricavare i dati sulla scena ripresa dal sensore e si occupa dell'interazione tra persona e schermo, resa il più naturale possibile per l'utente tramite l'utilizzo di *getures*: viene fornita al programmatore la possibilità di configurare le azioni eseguite a fronte di un certo gesto effettuato da un utente di fronte allo schermo.

- **Libreria per l'identificazione:** deve permettere l'identificazione di un utente nella scena, e quindi la possibilità di associare alla persona virtuale nell'applicazione alcune informazioni riguardanti la persona reale che si trova di fronte allo schermo (quali ad esempio il suo nome). Grazie a questa libreria lo schermo può così adattare i suoi contenuti all'identità della persona che gli sta di fronte, per esempio dandogli la possibilità di avere una sessione di interazione personalizzata.

La struttura del framework realizzato può essere meglio sintetizzata con la figura 2.4, che mostra come si articolano le varie librerie al suo interno,

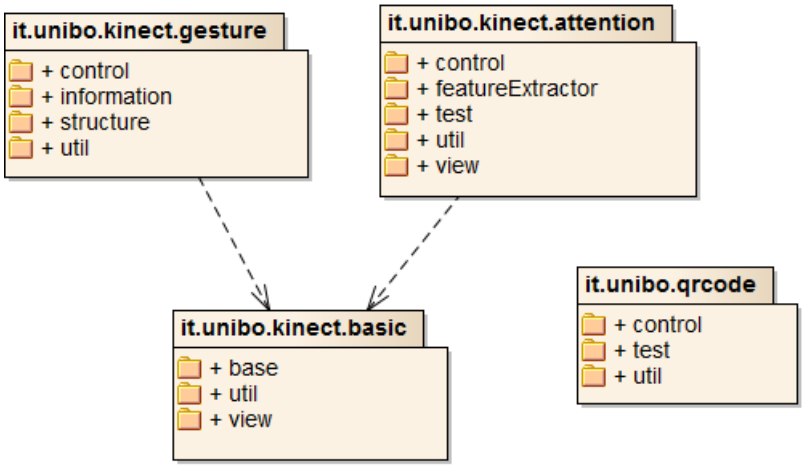

Figura 2.5: Diagramma UML per la struttura del framework

evidenziandone la struttura a livelli: si può infatti notare come le librerie progettate vadano a costituire nuovi *layer* su cui poi andranno ad appoggiarsi le future applicazioni utente.

Per una migliore esplicitazione delle dipendenze fra le librerie si fa riferimento al Package Diagram di UML (figura 2.5): si possono così facilmente notare le dipendenze esistenti tra la libreria per il riconoscimento dell'attenzione oppure quella per l'interazione naturale e quella per l'interfaccia con il sensore.

2.2 Interfaccia con il sensore

Questa libreria assume un'importanza rilevante all'interno del framework, poiché il suo scopo principale è quello di fornire le funzionalità di base, utili non solo al programmatore nella creazione di nuove applicazioni, ma anche alle librerie del framework che devono comunicare col sensore.

2.2.1 Elementi principali di progetto

Il suo nucleo fondamentale può essere riassunto con la fattorizzazione delle operazioni di base nell'utilizzo dei drivers (in questo caso OpenNI), utili a tutte le applicazioni che devono in qualche maniera utilizzare i dati provenienti dal sensore: ad esempio, inizializzare i componenti forniti dai driver per comunicare con il sensore fa parte di sicuro dell'insieme di operazioni comuni a tutte le applicazioni, così come la possibilità di mostrare le imma-

gini riprese. Anche il tracciamento dello scheletro di un utente presente nella scena viene considerata un'operazione comune a molti tipi di applicazioni e quindi inserito come funzionalità all'interno di questa libreria, poiché risulta fondamentale per avere maggiori informazioni sulle caratteristiche fisiche dell'utente di fronte allo schermo, utili per molteplici scopi, tra i quali anche l'analisi dell'utente per capire il suo livello di attenzione.

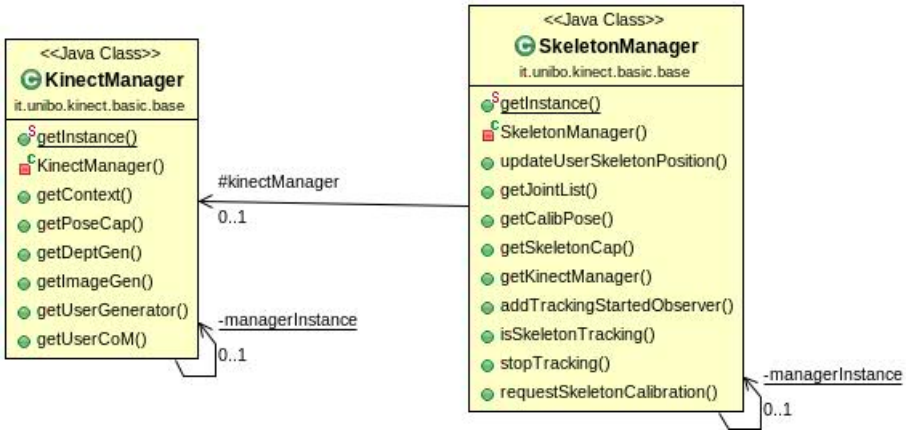

Figura 2.6: Diagramma UML per le classi manager

Per mettere a fattor comune tutte queste funzionalità di base, si è pensato al progetto di classi *Manager*, ognuna delle quali responsabile di un certo sottoinsieme di operazioni tra loro correlate. Nel diagramma UML in figura 2.6 vengono mostrate le due classi pensate per questo scopo: `KinectManager` si occupa di tutte le funzioni base di interfaccia con il sensore, e perciò è quella di maggior importanza, senza la quale non si potrebbe comunicare con il sensore, mentre `SkeletonManager` è specializzata nella gestione degli scheletri degli utenti. L'importanza di `KinectManager` è sottolineata dal fatto che è necessaria affinché `SkeletonManager` possa adempiere al suo compito, in quanto deve analizzare le informazioni ricavata tramite il sensore, come si può anche notare dal diagramma UML appena descritto.

Poter mostrare le immagini riprese dal sensore è sicuramente una funzionalità importante ed utile a molteplici applicazioni e perciò inserita in questo livello: per lo scopo, è stata progettata una tassonomia di classi che rappresentano dei pannelli in cui viene mostrata la scena ripresa.

Il diagramma 2.7 è sicuramente esplicativo in questo senso, poiché mostra le relazioni di ereditarietà fra i possibili pannelli da mostrare.

Oltre a questi, anche il riconoscimento della presenza di persone nella scena ripresa dal sensore è un'operazione considerata comune a molti scenari

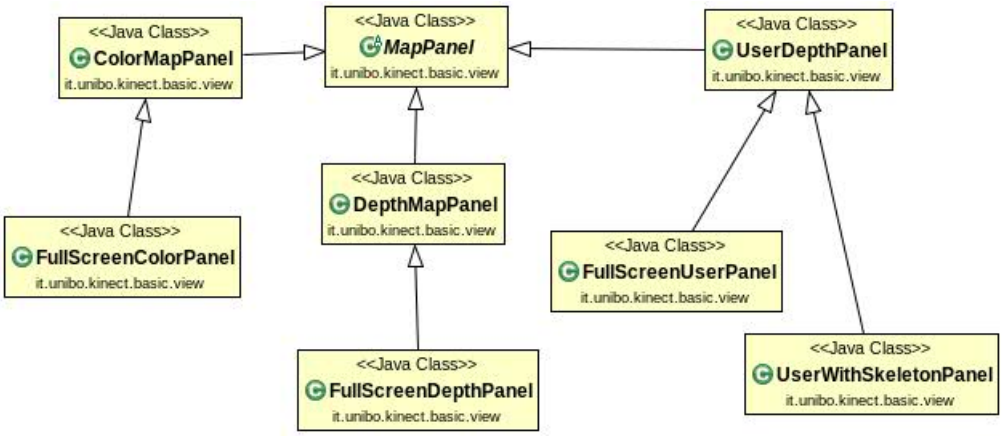

Figura 2.7: Diagramma UML per la tassonomia dei pannelli in cui mostrare la scena ripresa dal Kinect

applicativi: questo permette, ad esempio, di essere a conoscenza del numero di persone davanti allo schermo in un determinato momento, oppure di sapere se un certo utente precedentemente presente ha abbandonato la scena.

Un'altra funzionalità da offrire per semplificare l'utilizzo del sensore è una struttura predefinita per ottenere da esso i dati catturati: più nello specifico, le applicazioni sono rappresentate, in linea generale, da un ciclo in cui, una volta raccolti i dati dal sensore, se ne fa un particolare utilizzo; assume perciò un ruolo importante dare al programmatore la possibilità di configurare le azioni da eseguire con i dati ricavati facendolo concentrare pienamente su di esse e non sul recupero stesso dei dati. Già a livello di progetto risulta evidente che una soluzione di questo tipo può essere realizzata facendo uso del *template method pattern*[1] che permette, attraverso l'utilizzo di metodi astratti, di far specificare al programmatore il comportamento specifico da adottare nell'implementazione del metodo stesso, che resta perciò a suo carico per consentirgli di realizzare la propria logica applicativa con una maggiore flessibilità.

Nella figura 2.8 viene mostrato il diagramma UML della classe Kinect-ControlThread: al programmatore non resta che creare una classe che erediti da questa ed implementarne i metodi astratti per avere già pronto il flusso di controllo principale per la propria interazione, poiché in essa sono già incapsulate tutte le operazioni per una corretta gestione dell'aggiornamento dei dati provenienti dal sensore, comprendente anche il controllo degli utenti

[1]Uno dei *design patterns* più famosi ed utilizzati, descritto per la prima volta in [6]

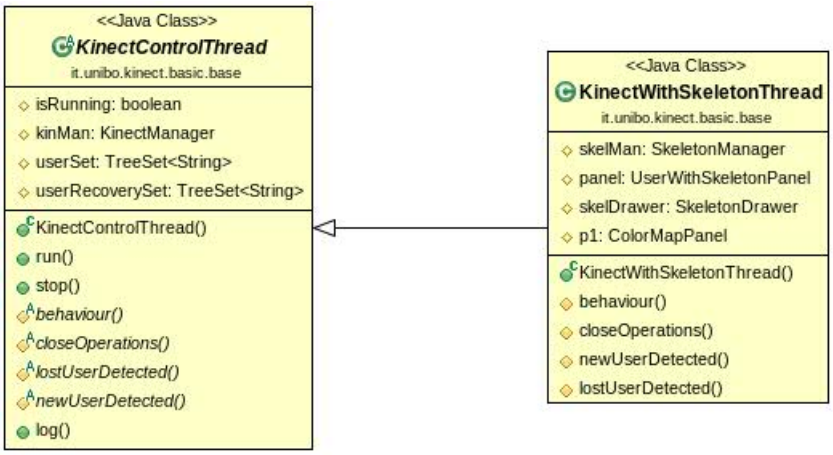

Figura 2.8: Diagramma UML per la classe KinectControlThread e una possibile subclass

presenti in scena. In questo specifico diagramma, nella subclass si utilizzano i dati per aggiornare e tracciare lo scheletro dell'utente.

Il comportamento generale della classe KinectControlThread viene invece riassunto in figura 2.9 con un diagramma UML delle Attività: questa struttura si riflette sui metodi astratti che la classe offre.

Riguardo al posizionamento di questa libreria nella struttura del framework, essa risulta essere in basso, poiché a stretto contatto con i driver per il sensore, assumendo così il ruolo di un nuovo livello logico su cui appoggiarsi per il progetto delle librerie che utilizzano il sensore: non solo quelle in seguito realizzate per questo framework, ma anche quelle che in futuro possono essere realizzate per estenderne le potenzialità e le funzionalità offerte, con l'aggiunta di nuovi strumenti per creare applicazioni in un tempo sempre minore e con meno sforzo. In questo modo si cerca di soddisfare il requisito di estensibilità del framework, di cui si è parlato nella sottosezione 2.1.1.

2.2.2 Assunzioni e criticità

Già a questo livello, bisogna comunque considerare importanti assunzioni derivanti dall'uso dei driver OpenNI per la realizzazione di questo progetto, che riguardano la segmentazione dell'utente nella scena, in particolare la sua posizione e come si muove al suo interno. Si possono verificare infatti delle imprecisioni quando l'utente è troppo vicino ad altre persone o oggetti nella scena (muri, sedie, ecc.), oppure quando due utenti si toccano mentre si muovono, e infine se ci sono occlusioni o l'utente esce di scena il suo ID

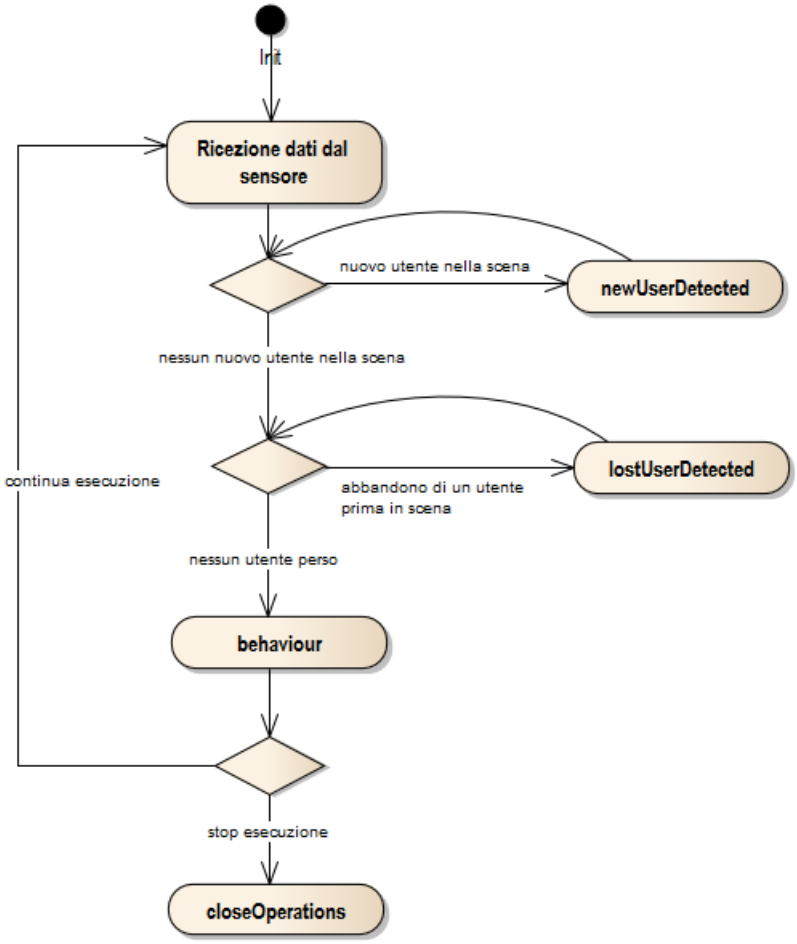

Figura 2.9: ActivityDiagram per KinectControlThread

può essere perso oppure scambiato erroneamente con un altro. Inoltre vi sono delle particolari assunzioni anche per il tracciamento dello scheletro: la parte superiore del corpo dell'utente deve trovarsi all'interno della visuale e la distanza ideale dal sensore è di 2.5 metri; inoltre, vestiti troppo larghi e capelli lunghi potrebbero degradare la qualità del tracciamento. Per quanto riguarda la calibrazione automatica (attiva di default), affinché questa possa funzionare correttamente, la maggior parte del corpo dell'utente dovrebbe essere visibile ed essere distante almeno 1 metro dal sensore, in quanto funziona per utenti già in piedi e non per quelli seduti. Per completare la calibrazione più velocemente ed avere risultati migliori sarebbe bene seguire ulteriori raccomandazioni: l'utente deve essere rivolto verso il sensore e stare in piedi

evitando movimenti veloci mentre le mani non devono nascondere l'area del torso. Per il tracciamento dello scheletro vengono usate delle euristiche che forniscono dei dati più plausibili quando il valore di confidenza è zero (ossia quando i giunti dello scheletro sono stati persi o sono ostruiti). Ad esempio se il valore di confidenza di un braccio è zero la sua posizione viene aggiustata in modo da essere a lato del corpo e puntare verso il basso, inoltre transizioni tra le posizioni tracciate e quelle calcolate tramite le euristiche avvengono in modo graduale su un certo numero di frame in modo da evitare salti.

2.3 Riconoscimento dell'attenzione

Questa libreria risulta di grande importanza perché permette di discriminare fra le persone che, all'interno della scena ripresa dal sensore, stanno veramente prestando attenzione allo schermo cui si trovano di fronte e tutte le altre, non interessate ai contenuti offerti dallo schermo. Per il progetto realizzato si è presa in larga considerazione la realizzazione utilizzata nel prototipo analizzato, facente riferimento al lavoro svolto in [5]. Prima di descrivere il progetto per questa libreria, va però dato qualche riferimento alla definizione di *attenzione* ed ai possibili modi per riconoscerla in una persona.

2.3.1 L'attenzione: definizione e modi per riconoscerla in letteratura

L'attenzione è la capacità dell'individuo di elaborare delle informazioni con la possibilità di selezionare determinati stimoli ambientali ignorandone alcuni altri. Si assume che questa capacità sia limitata per ogni individuo e che nell'eseguire qualsiasi compito venga richiesta una certa quantità della capacità totale. La misura dell'attenzione umana è un campo di ricerca tuttora attivo che coinvolge molte discipline(come le scienze cognitive, psicologia, intelligenza artificiale e data mining) e ad ora un metodo generale per misurarla o classificarla ancora non esiste, come si evince da [7].

Sempre nel lavoro [7] si cerca di stimare l'attenzione delle persone che transitano davanti a uno schermo pubblico in base alle variazioni del comportamento degli utenti e del contenuto informativo del display: viene utilizzato un Kinect per catturare i dati necessari all'analisi della scena di fronte allo schermo e un modello di attenzione simile a quello introdotto in [14]. In questo modello gli stimoli esterni possono essere filtrati in due modi: attivamente (o top-down,concentrando direttamente l'attenzione su qualcosa) o passivamente (o bottom-up, percezione degli stimoli in input). Inoltre si assume che un uomo abbia una quantità limitata di attenzione, distribuibile

a diversi processi contemporaneamente, in base a differenti priorità; si tiene infine conto solamente degli stimoli esterni, in quanto quelli interni e lo stato emotivo (top-down) non possono essere analizzati dal sistema. Per analizzare il comportamento sono state scelte determinate caratteristiche, tra le quali figurano velocità, direzione, accelerazione e distanza dallo schermo del soggetto, posizione e linearità del percorso che compie, oltre al movimento di testa e occhi e che sono state utilizzate per dividere i passanti in diverse tipologie:

- **Indifferente:** il soggetto non mostra reazioni visibili e non vi è nessun impatto sul suo comportamento, nulla di osservabile;
- **Glimpser:** si ha un minimo impatto sul movimento della testa, uno sguardo di breve durata, ma nessun impatto sul movimento;
- **Osservatore:** si ha un impatto sul movimento e sul comportamento e il soggetto guarda consapevolmente lo schermo;
- **Stopper:** il soggetto è concentrato sullo schermo, si ha un impatto visibile sul movimento e sul comportamento, il soggetto annulla precedenti attività che stava svolgendo e fa una sosta molto lunga.

In [5] viene preso in considerazione inoltre il rapporto tra postura di un soggetto e la sua attenzione: attraverso l'analisi di alcuni studi effettuati in letteratura si è giunti alla conclusione che posture più semplici (quali ad esempio una posizione seduta oppure una eretta con i piedi alla larghezza delle spalle) richiedono una minore quantità di attenzione da parte del soggetto, che è quindi maggiormente libero di focalizzare un quantitativo maggiore di attenzione ad altri scopi.

Anche la relazione dell'attenzione con il movimento degli occhi viene analizzata in base a vari studi effettuati in letteratura, anche nel campo delle scienze cognitive, giungendo alla conclusione che una persona attenta ai contenuti dello schermo presenta caratteristiche quali movimenti limitati della testa e degli occhi. Infatti durante la lettura il movimento degli occhi si concentra in una finestra piuttosto stretta, in cui avviene l'identificazione delle lettere o delle parole correnti.

Infine viene analizzata la relazione tra attenzione e tempo: l'elaborazione di informazioni è un processo che richiede tempo, e la natura delle rappresentazioni mentali costruite dall'integrazione degli stimoli e dei segnali esterni cambia continuamente al suo scorrere, diventando più sofisticata grazie all'integrazione con altre informazioni. Perciò, quando gli osservatori sono più lenti e viene speso maggior tempo per l'elaborazione visiva, l'attenzione viene guidata da una conoscenza di più alto livello.

2.3.2 Elementi principali di progetto

Per quanto riguarda il progetto di questa libreria, l'approccio utilizzato è ispirato alle analisi degli studi effettuati in letteratura, in particolare al lavoro svolto da Alois Ferscha nel progetto europeo SAPERE.

Riassumendo, considerando di essere nelle vicinanze di uno schermo, l'attenzione può essere misurata a partire dai seguenti parametri, ricavabili da un'analisi del comportamento dell'osservatore:

- velocità
- direzione
- accelerazione
- distanza
- posizione
- linearità del percorso seguito
- postura

Considerando questi parametri, un soggetto può essere considerato attento al contenuto del display se la sua velocità, accelerazione e distanza diminuiscono in direzione dello schermo, ed è orientato con la testa e il corpo verso il display, effettuando un percorso lineare nell'avvicinamento ad esso e mantenendo una postura sostanzialmente stabile.

A livello progettuale si evince quindi che, utilizzando la libreria descritta in 2.2 si devono prelevare i dati ricavati dal sensore sulla scena, in particolar modo quelli riguardanti lo scheletro, e analizzarli in modo da poter ricavare i valori dei parametri sopra elencati. In base a queste *features* si può così stimare un indice di attenzione, misurato assegnando un relativo peso ad ognuno delle feature analizzate. Inoltre deve anche definire precisamente cosa significhi, per il sistema considerato, la nozione di attenzione: si è deciso di considerare attento un utente che ha un indice di attenzione relativamente alto, è piuttosto vicino allo schermo e la sua permanenza davanti ad esso è abbastanza lunga.

Le funzionalità che questa libreria deve offrire sono quindi concentrate sulla misura dell'indice di attenzione per gli utenti presenti nella scena, più nello specifico:

- gestire l'analisi degli utenti presenti nella scena per calcolare il loro livello di attenzione;

- rendere disponibile, per ogni utente analizzato, il valore misurato dell'indice di attenzione;
- dare la possibilità di configurare i valori secondo i quali considerare attento un utente: il valore minimo di indice di attenzione, quello minimo per il tempo di permanenza di fronte allo schermo e la distanza massima
- dare la possibilità di conoscere, in un determinato momento, quali e quanti sono gli utenti considerati attenti di fronte allo schermo

Per lo scopo è stata progettata un'architettura multithreading in cui la classe (`AttentionManager`) responsabile di gestire i livelli di attenzione di ciascun utente utilizza dei thread secondari (`UserAnalyzer`) per l'analisi: ognuno di questi thread secondari è addetto all'analisi delle feature utili per capire il livello di attenzione di un solo utente, e ogni volta che ne ha stimato il valore, lo rende noto al manager dell'attenzione.

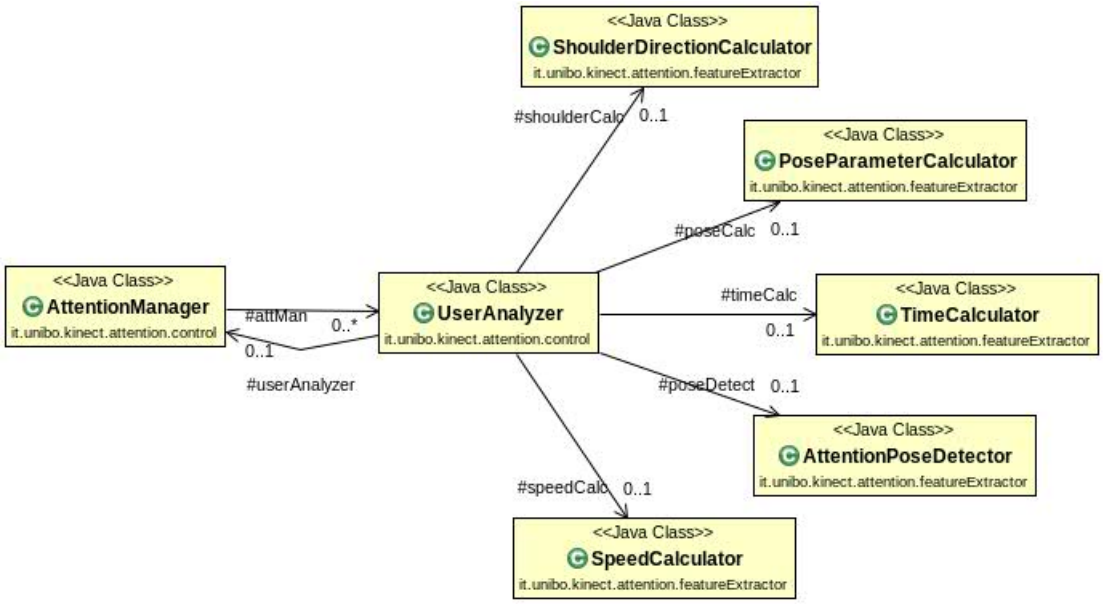

Figura 2.10: Struttura per l'analisi dell'attenzione degli utenti nella scena

La struttura di questa architettura è resa esplicita con il diagramma in figura 2.10: si può notare come la classe `UserAnalyzer` utilizzi delle classi utili ognuna all'analisi di una certa feature, mentre `AttentionManager` gestisce la vita dei thread secondari per attuare l'analisi degli utenti.

Nei confronti della struttura del framework questa libreria si pone al di sopra di quella per le funzionalità base descritta in 2.2, servendosi di essa nella comunicazione con il Kinect, costituendo un ulteriore livello logico su cui il programmatore dell'applicazione finale può appoggiarsi per utilizzare le funzioni riconoscimento del livello di attenzione degli utenti rilevati dal sensore.

2.4 Interazione naturale

In un sistema basato su uno schermo pervasivo adattativo poter interagire con esso con nient'altro che il proprio corpo è di grande importanza, ed è proprio questo lo scopo principale della libreria descritta in questa sezione, quella dedicata all'interazione tra schermo e persona, definita naturale in quanto si propone di essere il più vicina a quella comunemente e quotidianamente utilizzata dagli esseri umani, come ad esempio dei semplici gesti. Anche in questo caso, prima di descrivere il progetto di questa libreria, derivante dall'analisi del prototipo preso in considerazione e facente uso delle librerie OpenNI e NITE, e delle funzionalità che essa offre ad alto livello di astrazione, cioè senza entrare nei dettagli implementativi, si procede con una sintetica descrizione dell'interazione naturale.

2.4.1 Interazione naturale: definizione e breve introduzione

Per interazione naturale si intende un'interazione uomo-macchina basata basata su gesti comuni o quantomeno noti agli esseri umani: questo approccio mira così a fornire nuovi modi di interfacciarsi ai computer, sempre più intuitivi e naturali, anche nel senso che il loro tempo di apprendimento risulta essere davvero breve. Essa riduce il gap tra mezzi informatici e la realtà della vita quotidiana, e tra le sue caratteristiche principali vi sono certamente:

- non utilizza i principali strumenti di interazione tradizionalmente usati: il mouse e la tastiera
- utilizza schemi di interazione basati sui modelli di interazione persona-persona e persona-oggetto fisico
- è in grado tramite apposito hardware e software di comprendere azioni ed intenzioni degli utenti
- consente un'elevata mobilità all'utente per una piena naturalezza dei movimenti

- scompare il più possibile dall'ambiente e dagli oggetti potenziandone le funzionalità

- presta attenzione ad aspetti estetici e cognitivi

Il bisogno di questo tipo di interazione è stato reso necessario più che altro per ridurre i tempi di apprendimento di uso delle interfacce con la macchina e per ottenere dispositivi con i quali fosse sempre più facile interagire: a questo scopo, anche grazie al progresso tecnologico degli ultimi anni, stanno assumendo sempre maggiore importanza le NUI (Natural User Interaction).

Le NUI sono state definite come *la prossima fase evolutiva dopo il passaggio dall'interfaccia a riga di comando (CLI) all'interfaccia grafica utente (GUI)* [2]: esse sono infatti il prossimo passo nell'evoluzione delle interfacce utente, passate prima dalla CLI, in cui gli utenti dovevano utilizzare un mezzo artificiale (la tastiera) per immettere i dati e usare una serie di input in codice con una sintassi rigorosa, ricevendo i dati in uscita sotto forma di testo scritto, e poi dalla GUI, introdotta con l'utilizzo del mouse e di più facile utilizzo.

Figura 2.11: Evoluzione delle interfacce utente

L'introduzione della GUI ha segnato un significativo passo avanti nell'avvicinamento delle persone comuni, non dotate di particolari conoscenze, all'utilizzo di strumenti informatici; questo numero di persone è stato aumentato ulteriormente con l'introduzione della NUI visto che rende sensibilmente più facile, naturale ed intuitivo l'uso dei dispositivi anche ai meno esperti.

Alcuni dei più famosi esempi di NUI già usati attualmente possono essere:

[2] August de los Reyes, Predicting the Past,Sydney Convention Centre, Web Directions, 25 settembre 2008

- **Touch User Interface (TUI):** interfaccia touch-screen, che permette di interagire tramite il tocco delle dita; sicuramente è una delle più utilizzate, che si può trovare sia su dispositivi mobili come smartphone e tablet, sia su schermi di più grandi dimensioni;

Figura 2.12: Esempio di interfaccia touch

- **Gesture recognition system:** permette di interagire con il dispositivo tramite movimenti del proprio corpo; un esempio di queste interfacce è dato proprio dall'utilizzo del Kinect;

- **Speech recognition:** permette di interagire con il dispositivo tramite riconoscimento vocale; ormai di uso comune sia sugli smartphone che, ad esempio, integrato nella auto;

- **Gaze tracking:** consente di interagire con il dispositivo tramite il movimento dello sguardo.

2.4.2 Elementi principali di progetto

Per il progetto è importante sottolineare che si utilizzano le funzioni offerte dal middleware NITE che, appoggiandosi sui driver OpenNI, permette di riconoscere alcune delle più comuni gestures che un utente può effettuare. Appoggiandosi sulle funzionalità offerte dal componente software descritto in 2.2 per l'interazione di base con il sensore, questa libreria ha il compito di fornire al programmatore le seguenti possibilità:

- riconoscere una determinata gesture (tra quelle già predefinite da NITE) effettuata dall'utente, dando la possibilità di definire il comportamento che l'applicazione deve tenere una volta accertato il riconoscimento;
- fornire la possibilità di seguire i movimenti della mano di un utente, così da permettergli, ad esempio, di utilizzare la propria mano come cursore all'interno dello schermo;
- dare la possibilità di interagire con lo schermo tramite elementi visuali, quali ad esempio pulsanti, che compaiono sullo schermo, fornendo la possibilità di crearne facilmente di nuovi a seconda delle esigenze applicative, estendendo così in modo semplice il framework.

La classe `GestureManager` è quindi quella responsabile dell'interfaccia con le librerie OpenNI e NITE per quanto riguarda le gestures e, sfruttando l'architettura ad eventi già presenti in queste librerie, della configurazione degli ascoltatori per gli eventi generati quando il sensore rileva che l'utente ha compiuto una gesture. Inoltre essa ha anche la responsabilità della configurazione dell'ascoltatore per gli eventi generati dl tracciamento della mano.

Visto che il middleware offerto da NITE offre già delle primitive in grado di riconoscere delle gestures di base, la parte principale del progetto di questa libreria è costituita dalla creazione di elementi grafici da mostrare sullo schermo con cui l'utente può interagire: a questo scopo sono stati pensati dei comandi attivabili posizionandovi sopra la mano. La mano dell'utente viene tracciata tramite il sensore per ricavarne la posizione corrispondente all'interno dello spazio costituito dallo schermo, in base a cui si può stabilire qual'è oggetto di interazione in un dato momento ed in questo modo determinarne lo stato. Infatti i comandi sono stati pensati come pannelli con uno stato che cambia a seconda che essi siano o meno oggetto di interazione da parte dell'utente. Come si può notare dalla figura 2.13, i possibili stati per un oggetto di questo tipo (elencati nella enumeration `GestureState`) sono:

- `INACTIVE`, che rappresenta lo stato in cui si trova il componente quanto l'utente non sta interagendo con esso, ossia non vi ha la mano posizionata sopra;
- `ACTIVE`, quando la mano dell'utente è posizionata all'interno dello spazio occupata dal componente, e si muove entro i suoi limiti;
- `PRESSED`, rappresentante lo stato in cui il componente è stato premuto, in cui si accede tenendo ferma la mano per certo tempo all'interno dello spazio occupato dall'oggetto.

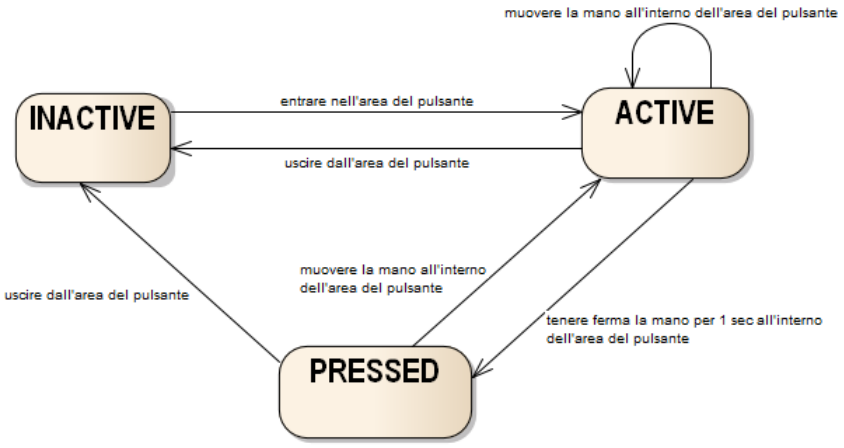

Figura 2.13: Diagramma UML a stati per GestureGUIPanel

Le transizioni da uno stato all'altro dovranno poi essere resi visibili all'utente finale, ad esempio cambiando l'immagine che caratterizza il componente sullo schermo. Il comportamento rappresentato da questa macchina a stati è condiviso da tutti i tipi di `GestureGUIPanel` grazie al meccanismo dell'ereditarietà: per il progetto qui mostrato sono stati definiti GestureGUI per bottoni, slider [3] e dial (delle manopole), ma con questa architettura risulta facile estendere la libreria per creare altri diversi tipi di comandi. Il pun-

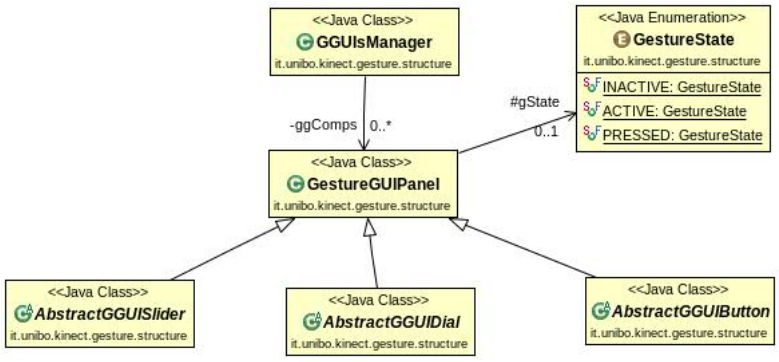

Figura 2.14: Gerarchia per elementi grafici interattivi tramite interfacce naturali

to cardine di questo progetto è perciò rappresentato dalla tassonomia degli

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

oggetti `GestureGUIPanel`, riportata in figura 2.14: in esso viene citato anche `GestureGUIManager`, che rappresenta un contenitore per questi oggetti, memorizzandone e gestendone il posizionamento nello schermo. É tramite quest'oggetto che si riesce, a partire dalle coordinate in cui si trova la mano dell'utente, a discriminare quale GestureGUI è attiva in qualsiasi momento.

Come meccanismo di notifica della avvenuta pressione di una GestureGUI si chiama il metodo `announcePress`, che deve essere implementato dalle classi che vogliono gestire questo tipo di oggetti: infatti non appena la GestureGUI entra nello stato `PRESSED` invoca questo metodo sul relativo oggetto, che, nel caso gestisca più di uno di questi elementi, dovrà discriminare in base alle informazioni passate come parametri, quale di questi è quello che ha generato la notifica. Anche le informazioni utilizzate nella notifica, visto

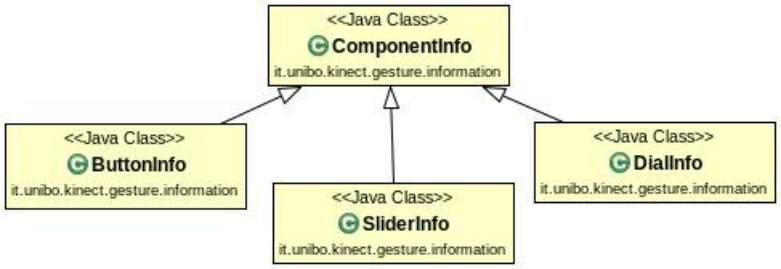

Figura 2.15: Tassonomia per le informazioni relative agli elementi grafici interattivi

che variano a seconda del tipo di elemento grafico usato, sono state modellate con una tassonomia, riportata in figura 2.15. Per esempio, rispetto alle informazioni relative alla pressione di bottone, quelle per uno slider avranno anche l'indicazione del livello voluto dall'utente. Infine viene presentato il diagramma UML in figura 2.16, che mostra come i vari `GestureGUIPanel` utilizzino diverse `ComponentInfo` a seconda del loro tipo.

Una delle principali assunzioni da considerare a questo livello è che si è deciso di considerare l'interazione con un solo utente alla volta, poiché, come è facilmente intuibile, possono insorgere delle imprecisioni nel riconoscimento di alcune gestures o nel tracciamento della mano dell'utente quando due persone all'interno della scena sono troppo vicine, oppure le loro mani si sovrappongono.

All'interno del framework questa libreria si colloca appena sopra quella descritta in 2.2, andando a costituire un ulteriore livello logico tra essa e le applicazioni che possono essere realizzate per questi sistemi: essa assume però notevole importanza dal momento che permette un'interazione intuitiva

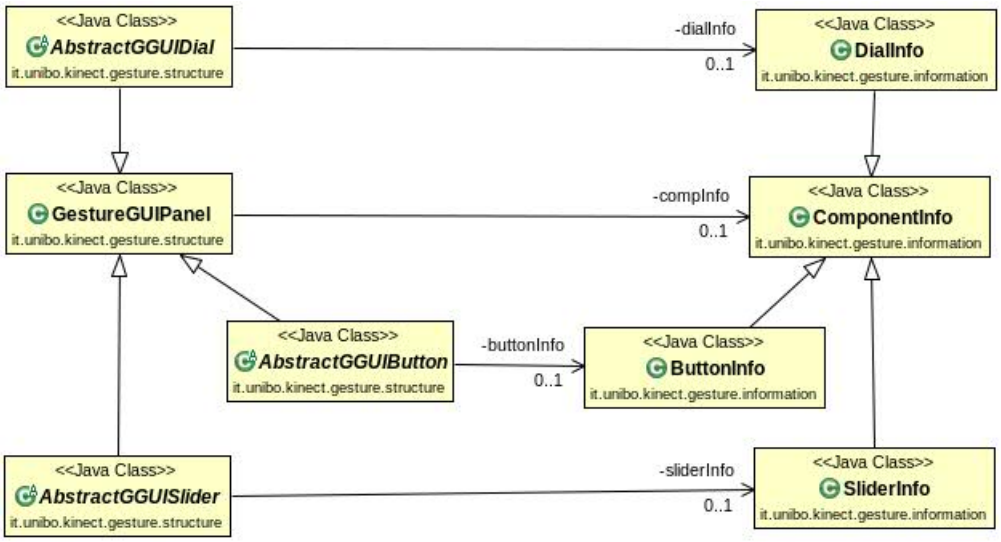

Figura 2.16: UML per la struttura degli elementi interattivi: collegamento con le relative informazioni

tra persona e schermo, aprendo molti nuovi possibili scenari di utilizzo del framework, che con questa aggiunta, aumenta di molto le potenzialità e le funzionalità offerte.

2.5 Identificazione dell'utente

L'identificazione di un utente assume molta importanza in un sistema per uno schermo pervasivo, in quanto ottenere informazioni sulla persona di fronte ad esso permette di adattarne i contenuti sulla base dell'utente stesso mantenendo un alto grado di trasparenza: per questo motivo la libreria che si occupa dell'identificazione dell'utente assume un ruolo importante all'interno del framework, aumentandone le potenzialità e i possibili utilizzi.

Per ricavare informazioni sull'utente di fronte allo schermo non deducibili tramite l'utilizzo del sensore utilizzato, una possibile soluzione è quella di sfruttare un oggetto di uso comune, quale lo smartphone, per interagire con lo schermo e comunicare ad esso le informazioni desiderate. L'obiettivo principale della libreria è infatti quello di dare un'identità agli utenti rilevati nella scena dal sensore: a questo proposito è stato pensato di utilizzare un codice QR ogni qualvolta lo schermo necessiti di accoppiare ad un utente delle informazioni riguardanti la sua identità. In sintesi, per identificarsi l'utente deve scansionare il codice QR mostrato dallo schermo tramite un'applicazio-

ne pensata ad hoc per lo scopo, che una volta terminata la scansione invia allo schermo i dati necessari per l'identificazione.

Prima di procedere con la descrizione delle funzionalità pensate per questa libreria si fornisce una breve descrizione dei codici QR e della libreria per utilizzarli.

2.5.1 Codici QR e ZXing

Un codice QR è un codice a barre bidimensionale (o 2D), ossia a matrice, composto da moduli neri disposti all'interno di uno schema di forma quadrata, generalmente impiegato per memorizzare informazioni destinate a essere lette di solito tramite un telefono cellulare o uno smartphone; in un solo crittogramma possono essere contenuti 7.089 caratteri numerici o 4.296 alfanumerici. Il nome QR è l'abbreviazione dell'inglese *quick response*, in virtù del fatto che il codice fu sviluppato per permettere una rapida decodifica del suo contenuto: esso fu infatti sviluppato nel 1994 dalla compagnia giapponese Denso Wave allo scopo di tracciare i pezzi di automobili nelle fabbriche di Toyota; vista la capacità del codice di contenere più dati di un codice a barre, venne in seguito utilizzato per la gestione delle scorte da diverse industrie.[4] In seguito al rilascio dei codici QR sotto licenza libera da parte della Denso Wave, essi si sono diffusi dapprima in Giappone e poi, più lentamente negli USA e in Europa. Dalla metà degli anni 2000 questi codici sono divenuti sempre di più comune utilizzo, ad esempio per scopi pubblicitari, poiché sono in grado di veicolare facilmente URL e indirizzi.

ZXing è invece una libreria Open Source implementata in Java: essa permette non solo la decodifica di codici QR tramite la scansione con la fotocamera di uno smartphone (tramite un'applicazione scaricabile per i modelli Android per esempio), ma anche la creazione di codici QR al cui interno sono encodificate le informazioni volute.[5] Per questi motivi si è utilizzata la libreria sopra citata per utilizzare i codici QR sia lato smartphone che lato schermo.

2.5.2 Elementi principali di progetto

Il nucleo fondamentale di questa libreria è l'interazione tra schermo e smartphone, per cui essa si deve dotare necessariamente di due parti: la prima deve dare la possibilità allo schermo di creare un codice QR e di mostrarlo, attendendo che sia scansionato per una identificazione, mentre la seconda,

[4] `http://it.wikipedia.org/wiki/Codice_QR`

[5] `https://code.google.com/p/zxing/`

Figura 2.17: Esempio di codice QR

lato smartphone, deve dare la possibilità di scansionare il codice QR inviando automaticamente le proprie informazioni allo schermo. Per il progetto trattato, si è deciso di progettare un'applicazione per smartphone Android, appoggiandosi all'applicazione sviluppata da ZXing per la decodifica di codici QR [6].

La libreria dovrà quindi fornire al programmatore la possibilità di inserire l'identificazione dell'utente in modo flessibile, permettendogli di decidere cosa fare sia per mostrare il codice sia una volta avvenuta l'identificazione: a questo scopo si può già notare, a livello di progetto, come il template method pattern può risultare utile nel fornire le linee guida per la realizzazione.

Il diagramma in figura 2.18 mostra infatti il comportamento generale che un server di richieste di identificazione basate su decodifica di codici QR deve avere: tra le operazioni citate nel diagramma, quelle per mostrare il codice QR e per fare utilizzo delle informazioni aggiuntive sull'utente ricevute sono quelle che meglio si prestano ad essere modellate come astratte, poiché possono differire abbastanza sensibilmente a seconda della logica applicativa da soddisfare (ad esempio mostrare queste informazioni oppure utilizzarle per una query su di un Database). Si può inoltre notare come sia già chiaro a livello progettuale che questo server debba avere un proprio flusso di controllo separato per rispondere tempestivamente alle richieste che gli giungono, e per questo dovrà essere eseguito in un thread separato da quello principale dell'applicazione.

[6] `https://play.google.com/store/apps/details?id=com.google.zxing.client.android`

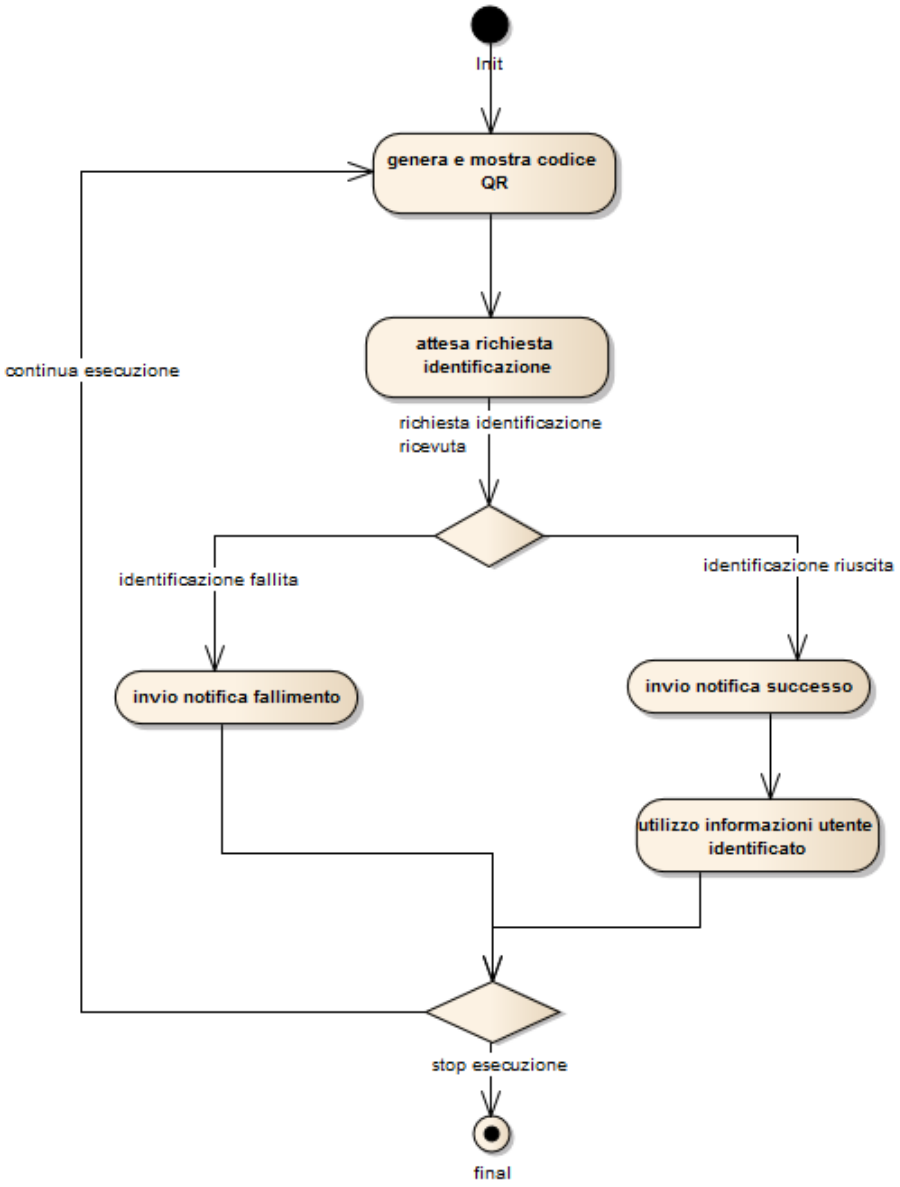

Figura 2.18: ActivityDiagram per il server di richieste di identificazione

Viene proposto infine anche il diagramma per l'applicazione per smartphone, che nell'architettura considerata assume il ruolo di client: in esso si mostra la sequenza delle operazione principali da eseguire ogni qualvolta che si vuole essere identificati.

Per la comunicazione tra le due entità client-server in gioco viene utilizza-

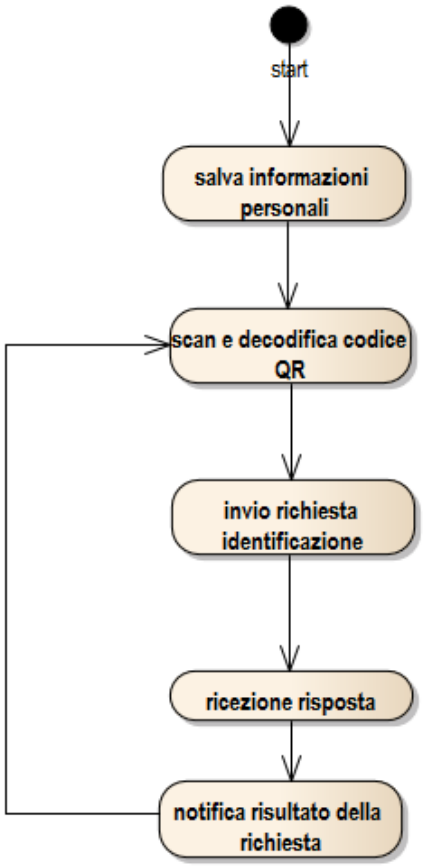

Figura 2.19: ActivityDiagram per l'applicazione Android per l'identificazione

ta una connessione TCP, in cui il client assume il ruolo di initiator: è infatti l'utente che, attivando l'applicazione sul suo smartphone e decodificando il codice QR invia una richiesta al server.

La decodifica del codice QR è necessaria dal momento che in esso sono encodificate, in forma di stringa ipaddress:port?sessionID, le informazioni necessarie per raggiungere il server (indirizzo IP, numero di porta e codice segreto di sessione): tramite questo espediente ci si assicura che solamente le persone che vedono il codice, e quindi si trovano di fronte allo schermo nelle sue vicinanze, possono identificarsi.

Per gestire l'identificazione è stato progettato anche un semplicissimo protocollo di richiesta-risposta tra client e server, che prevede quattro semplici fasi:

- il client invia al server il sessionID ricavato grazie alla decodifica del

codice QR

- il server controlla se il sessionID ricevuto è corretto e, a seconda che lo sia, invia un messaggio di errore o di conferma al client
- se il client riceve una notifica di successo della prima fase di identificazione, invia le sue informazioni encodificate in forma di file XML
- il server prova a ricavare dal file XML ricevuto le informazioni riguardo all'utente, ed invia al client un messaggio di conferma o di errore a seconda della buona riuscita di questa operazione

É importante sottolineare che per il progetto di questa libreria non si è fatto uso del sensore Kinect, ma solamente della libreria di ZXing: per questo motivo all'interno del framework questa libreria assume una posizione intermedia tra le applicazioni utente e la libreria appena citata, fornendo un ulteriore livello di astrazione per dare al programmatore uno strumento di più facile utilizzo per l'identificazione dell'utente.

Capitolo 3

Implementazione del framework e deployment

In questo capitolo vengono descritte le informazioni principali e i punti salienti riguardanti l'implementazione delle librerie sviluppate per il framework, in modo tale da fornire anche una guida utile a coloro che lo utilizzeranno. Ove possibile, per una maggiore concretezza ed efficacia della descrizione, si fa riferimento a semplici casi applicativi realizzati ad hoc per mostrare le potenzialità e alcuni possibili utilizzi delle librerie.

Bisogna ricordare che è stato deciso di implementare il framework con il linguaggio Java, utilizzando il Kinect come sensore e i driver OpenNI 1.5.4.0 e NITE 1.5.2.23. L'implementazione fa quindi necessariamente riferimento ai concetti introdotti dagli strumenti utilizzati.

Lo scopo di questa descrizione è quello di fornire i dettagli di maggior rilievo della fase di implementazione, quindi per un livello di dettaglio maggiore si rimanda alla documentazione del codice stesso.

3.1 Interfaccia con il sensore

Lo scopo principale di questa libreria è quello di fornire le operazioni più comuni per i sistemi software considerati in questo lavoro, quali l'inizializzazione del sensore e le operazioni per ricavare da esso i dati.

3.1.1 Struttura

La libreria è strutturata in 3 package:

- `it.unibo.kinect.basic.base` in cui sono contenuti le classe che implementano la logica della libreria

- `it.unibo.kinect.basic.util` in cui é contenuta una classe di utilità
- `it.unibo.kinect.basic.view` in cui sono presenti le classi utili a visualizzare le immagini rilevate dal sensore sullo schermo

Il package it.unibo.kinect.basic.base

Risulta immediatamente evidente come il *core* della libreria sia rappresentato dal package *base*, in cui sono implementate le classi che racchiudono la logica delle funzionalità offerte della libreria. Esse sono:

- `KinectManager`: rappresenta una classe di gestione di base della periferica usata come sensore attraverso i driver OpenNI. Questo oggetto si occupa quindi, alla sua creazione, di inizializzare il sensore per la raccolta dei dati e delle entità messe a disposizione dai driver per raccoglierli: esse sono chiamate, in questa versione di OpenNI, *generators*. Più nello specifico, ci si occupa di istanziare un generator per ogni tipo di dato che si può voler raccogliere dal sensore (ad esempio quello ricavato dal sensore di profondità (`DepthGenerator`) o da quello a colori RGB (`ImageGenerator`), oltre a quello utile per riconoscere gli utenti nella scena(`UserGenerator`)). Si offrono perciò dei metodi per ottenere i riferimenti a questi generator (comunemente chiamati getter). Questo oggetto riesce anche, sfruttando le delegazione al generator degli utenti nella scena, a restituire la posizione, in coordinate 3D, del centro di massa di un utente presente nella scena.

- `SkeletonManager`: è la classe che permette la gestione del tracciamento dello scheletro di un utente all'interno della scena grazie all'utilizzo di `SkeletonCapability`. Essa ha il compito di inizializzare le componenti fornite dai driver per tracciare le scheletro, e permette, oltre all'aggiornamento dei *joint*[1], anche di aggiungere delle azioni da effettuare al completamento della fase di calibrazione dell'utente, che precede il vero e proprio tracciamento. Infatti in questo caso è stata ampliata l'architettura ad eventi proposta da OpenNI, utilizzando il pattern *Observer*: la fine della calibrazione viene riconosciuta dalla classe `CalibrationCompleteObs`, implementata come observer a sua volta osservabile, poiché notifica l'avvenuta compilazione a tutte le entità precedentemente registratesi come ascoltatori. É in questa classe che viene tenuta una struttura dati, più precisamente una HashTable,

[1] I joint rappresentano in OpenNI quello che, intuitivamente rappresentano per gli esseri umani le articolazioni, ossia punti di raccordo tra le varie componenti dello scheletro.

contenente le informazioni relative ad ogni giunto dello scheletro di ogni utente correntemente tracciato: per il progetto realizzato i giunti presi in considerazione sono quelli relativi a testa, collo, spalle, gomiti, mani, torso, anche, ginocchia e piedi, poiché supportati da OpenNI.

- `KinectControlThread`: rappresenta il template base per una classe che intende prelevare ed interpretare i dati provenienti dal sensore. Era stato infatti notato, durante la fase di progetto, che tutte le applicazioni di questo tipo sono costituite da cicli in cui prima si raccolgono i dati dal sensore e poi se ne fa un particolare uso, iterativamente. Per fornire al programmatore un componente software che permettesse di concentrarsi solo sulla logica applicativa e non sull'interazione col sensore, questa classe è stata dichiarata astratta: attraverso l'uso del Template Method pattern infatti essa si occupa di ricavare i dati dal sensore (ovviamente tramite l'utilizzo di un oggetto di classe `KinectManager`), lasciando nelle mani del programmatore della particolare applicazione utente l'implementazione dei metodi riguardanti l'interpretazione dei dati. Oltre alle azioni riguardanti l'utilizzo dei dati(metodo `behaviour`), viene offerta la possibilità di specificare le operazioni da compiere sia quando un nuovo utente (metodo `newUserDetected`) viene rilevato sia quando viene perso uno precedentemente presente (metodo `lostUserDetected`), oltre a quelle da compiere alla chiusura dell'applicazione (metodo `closeOperations`).

La criticità maggiore riguardante l'implementazione di questo package è sicuramente rappresentata dal riconoscimento della perdita di un utente precedentemente presente nella scena: infatti dalle specifiche di OpenNI, la segnalazione che un utente è uscito di scena avviene circa 10 secondi dopo che l'utente è effettivamente fuori dal campo di vista e per ora non vi è modo di modificare questo valore. Questa bassa reattività non è funzionale ai sistemi considerati: l'obiettivo infatti è che lo schermo reagisca in tempo reale all'uscita di un osservatore, sia per non continuare ad offrire contenuto informativo ad un utente che in realtà non sta più prestando attenzione allo schermo, sia per permettere una maggiore reattività nell'individuazione degli utenti realmente presenti all'interno della scena. Per raggiungere questo obiettivo si è preferito implementare un modo alternativo all'uso delle primitive di OpenNI: si ottiene infatti tramite il generator addetto al riconoscimento degli utenti una struttura dati contenente degli interi, ognuno dei quali rappresenta un determinato pixel dell'immagine. Quando il valore dell'intero è diverso da zero, significa che il pixel da esso rappresentato è costituente di una determinata persona, di cui il valore stesso ne rappresenta l'identificatore. Controllando questa struttura ad ogni raccolta dei dati

provenienti dal sensore, si è in grado di ottenere l'informazione cercata sulla presenza o assenza degli utenti nella scena senza latenze, a discapito, come si può facilmente intuire, di un piccolo costo computazionale.

Altre note importanti riguardano le due classi *manager*: poiché esse hanno la responsabilità di gestire la comunicazione con il sensore, sono state implementate attraverso l'uso del pattern Singleton[2], in modo tale che, all'interno di un'applicazione, esse possano essere istanziate una sola volta così da prevenire errori che potrebbero sorgere cercando, ad esempio, di inizializzare più generator dello stesso tipo per ricavare dati da un unico sensore.

Il package it.unibo.kinect.basic.util

L'unica classe implementata in questo package è `PosAndTime`: che rappresenta la posizione e il momento di acquisizione di un giunto dello scheletro; questa informazione viene inserita nella struttura dati contenente le informazioni riguardanti lo scheletro di un particolare utente.

Il package it.unibo.kinect.basic.view

Le classi implementate in questo package rispondono alle esigenze di mostrare sullo schermo le immagini riprese: si precisa che, per il progetto del framework, è stato deciso di utilizzare la libreria Swing di Java, perciò in questa implementazione si fa riferimento ai suoi concetti.

La parte più importante è sicuramente costituita dalla tassonomia di classi creata per fornire all'utente le immagini raccolte dai vari tipo di sensori offerti dal Kinect: ad esempio è presente sia una classe per le immagini a colori sia una per quelle di profondità. Per l'implementazione, si è scelto di fornire le immagini in un pannello (classe Swing `JPanel`) che deve essere ridisegnato ogni volta che le informazioni provenienti dal sensore vengono aggiornate (quindi con una frequenza di circa 30 HZ). La tassonomia è stata implementata grazie alla classe astratta `MapPanel`, che incapsula tutte le operazioni comuni ai vari tipi di pannelli, poi estesa dalle altre classi che ne hanno implementato (o anche sovrascritto) i metodi a seconda delle particolari esigenze:

- `ColorMapPanel` mostra le immagini a colori ricavate dal sensore RGB
- `DepthMapPanel` mostra le immagini di profondità
- `UserDepthPanel` estende il pannello della mappa di profondità evidenziando gli utenti presenti nella scena con vari colori

[2]Uno dei design pattern descritti in [6]

- `UserWithSkeletonPanel` estende `UserDepthPanel` aggiungendovi anche il tracciamento dello scheletro per ogni utente, e per questo necessita anche della classe addetta alla gestione degli scheletri e della classe `SkeletonDrawer` che, a partire dalle informazioni riguardo ai giunti, traccia sull'utente lo scheletro.

In particolare il metodo di maggior rilievo è `updateImage`, tramite il quale si richiede l'aggiornamento dell'immagine mostrata dal pannello: esso è perciò dichiarato come astratto nella radice della tassonomia, poiché implementato in maniera differente a seconda del particolare pannello implementato. Dei pannelli a colori e di profondità sono state fornite anche le versioni a schermo pieno, realizzate adattando l'immagine (di standard 640x480) all'ampiezza dello schermo. Inoltre va ricordato che è presente anche la classe `KinectMapFactory`, una factory che permette al programmatore la creazione dei vari pannelli in modo più intuitivo.

3.1.2 Indicazioni di utilizzo

Per utilizzare le funzionalità messe a disposizione, basta istanziare la classe che si occupa di interagire con il sensore e, se necessario, quella addetta alla gestione dello scheletro; il nucleo dell'applicazione verrà comunque realizzato estendendo la classe `KinectControlThread` che, implementando l'interfaccia *Runnable*, può essere eseguita come un Thread. É in questa classe che, una volta raccolti i dati dal sensore, vanno aggiornate tutte le entità che lo necessitano: supponendo, ad esempio, di voler mostrare le immagini riprese tramite uno dei pannelli forniti dalla libreria, è nel metodo `behaviour` che bisogna richiamare l'aggiornamento del pannello; oppure, supponendo di voler tracciare gli scheletri degli utenti, bisogna fermare il tracciamento ogniqualvolta un utente scompare dalla scena e iniziarlo per ogni nuovo utente che arriva. Proprio sul tracciamento degli scheletri è basato l'esempio di `KinectWithSkeletonThread`, che mostra come utilizzare il template astratto per un thread che deve ricavare dati dal Kinect: esso può essere funzionale anche come esempio per capire come aggiornare correttamente tutte le strutture dati e gli oggetti necessari ad una corretta gestione degli scheletri degli utenti.

3.2 Riconoscimento dell'attenzione

Lo scopo di questa libreria è quello di fornire un componente che, interagendo con il sensore, riesca a riconoscere se gli osservatori che transitano

davanti allo schermo gli stanno porgendo attenzione, stimando un valore che ne quantifichi il livello.

3.2.1 Struttura

La libreria è stata strutturata nei seguenti quattro package:

- `it.unibo.kinect.attention.control` in cui sono contenute le classi che implementano la logica della libreria
- `it.unibo.kinect.attention.featureExtractor` in cui vi sono le classi utili ad estrarre le informazioni per ricavare il livello di attenzione a partire dai dati ricavati dal Kinect
- `it.unibo.kinect.attention.util` che contiene le classi di utilità
- `it.unibo.kinect.attention.view` che fornisce un semplice pannello in cui visualizzare i livelli attenzione per gli utenti

Il package it.unibo.kinect.attention.control

Le classi implementate in questo package sono tutte importanti ai fini della comprensione della libreria:

- `AttentionManager` è la classe responsabile di tenere traccia dello stato del livello di attenzione egli utenti. Essa permette infatti di iniziare e fermare l'analisi, inizializzando tutte le risorse e gli oggetti necessari: è a questa classe che le applicazioni devono rivolgersi per sapere il livello di attenzione di un certo utente oppure se può essere considerato attento. Essa si occupa inoltre di caricare i valori per i parametri di configurazione dal file AppConfig.xml, che possono essere modificati in modo da soddisfare svariate esigenze applicative. É importante notare che è richiesta l'analisi dello scheletro degli utenti per calcolare il loro livello di attenzione: è infatti da essa che si possono ricavare le informazioni più rilevanti, e per questo motivo è richiesto un oggetto di classe `SkeletonManager` come parametro del costruttore di questa classe.
- `TrackingStartedObserver` estende `Observer`, ed è pensato per utilizzare l'architettura ad eventi proposta nella libreria 3.1.1, visto che l'analisi dell'attenzione richiede l'analisi dello scheletro che può essere effettuata, in OpenNI, solo dopo una fase di calibrazione, il cui completamento è proprio l'evento che questa classe si propone di ascoltare: alla notifica dell'inizio del tracciamento dello scheletro per un utente

è perciò compito di questa classe notificare al manager dell'attenzione che può far partire l'analisi per quell'utente.

- `UserAnalyzer` rappresenta un pezzo fondamentale dell'architettura di questa libreria, in quanto è la classe responsabile dell'analisi dell'utente per calcolarne l'indice di attenzione. La sua funzione risiede sinteticamente in un ciclo in cui, quando i dati dell'utente sono disponibili, utilizza gli oggetti preposti all'analisi delle varie features utili per quantificare il livello di attenzione: una volta stimato il valore di questo indice, notifica al manager dell'attenzione i risultati, per poi ricominciare da capo, perlomeno fino a quando non viene fermato. É importante ricordare che i valori secondo i quali `UserAnalyzer` considera un utente attento possono essere cambiati nel file di configurazione, permettendo un uso più flessibile e personalizzabile di questa classe.

L'indice di attenzione

Prima di passare alla descrizione degli altri package risulta necessaria una sintetica descrizione riguardante il calcolo dell'indice di attenzione.

Per la stima dell'indice dell'attenzione ad ogni feature misurata viene associato un valore che indica quanto un determinato vincolo è stato rispettato. Prendendo spunto dagli studi sull'attenzione già presenti in letteratura si può dire che l'attenzione di un utente aumenta se:

- l'utente è seduto;
- non è sbilanciato ma ha una posizione centrata rispetto all'asse verticale del corpo;
- la base di sostegno è circa pari alla larghezza delle spalle;
- è in double support phase piuttosto che in single support phase;
- il corpo dell'utente è rivolto verso il sensore;
- l'utente è fermo o si muove molto lentamente;
- resta un certo tempo davanti al display per leggerne e capirne il contenuto.

Nella tabella 3.1 sono indicati i valori assegnati di default a ciascun aspetto precedentemente elencato.

In essa t è il tempo totale che l'utente rimane davanti allo schermo con un angolo di torsione delle spalle ridotto e tempoNecessario deve essere deciso in

Postura	**+28**: seduto
	+14: in piedi
	+14: double support phase
	+7: single support phase
	+7: sbilanciamento verso destra o sinistra
	+14: nessuno sbilanciamento
	+14: base di sostegno normale
	+7: base di sostegno stretta o larga
Direzione delle spalle	**+14**: angolo minore di 15°
	+2: altrimenti
Velocità	**+14**: fermo
	+3: lento
	+1: veloce
Tempo	**+t*14/temponecessario**
Posizione pensante	**+2**, altrimenti **0**

Tabella 3.1: Tabella per il calcolo dell'indice di attenzione

base al numero di parole da leggere: 175ms è il tempo minimo per capire una parola e spostarsi a quella successiva. In questo modo l'utente che rispetta tutti i vincoli ha un indice di attenzione vicino a 100, mentre decresce se i vincoli non vengono rispettati.

Un utente si può considerare attento se il suo indice di attenzione è maggiore di 80, mentre per valori inferiori a 60 l'utente si può valutare come non attento al contenuto dello schermo.

Il package it.unibo.kinect.attention.featureExtractor

In questo package vi sono le implementazioni delle classi utilizzate per calcolare la parte di indice di attenzione dovuta ad ogni singola feature analizzata:

- `AttentionPoseDetector` guarda se l'utente ha assunto una postura attenta
- `PoseParameterCalculator` analizza la postura dell'utente (se è in piedi o seduto, in equilibrio o meno)
- `ShoulderDirectionCalculator` analizza la posizione del torso per calcolare la direzione dell'utente rispetto al sensore
- `SpeedCalculator` analizza la velocità dell'utente

- `TimeCalculator` tiene conto del tempo passato dall'utente di fronte al sensore

Il package it.unibo.kinect.attention.util

In questo package vi sono le strutture dati utili per l'implementazione, tra le quali la più importante è di sicuro `UserData`: infatti in essa vengono memorizzati i dati relativi ad un singolo utente nella scena. É proprio questa la struttura dati che si controlla con `UserAnalyzer`.

Il package it.unibo.kinect.attention.view

Viene fornita la sola classe `AttentionPanel`, molto utile perché implementa un semplice pannello predisposto per mostrare tutti i valori per i livelli di attenzione attualmente misurati per gli utenti in scena: basta così aggiornarlo invocando il metodo `updateAttentionLevels()` per avere una visione immediata dell'indice di attenzione calcolato per gli utenti.

3.2.2 Esempio di riferimento

L'applicazione a cui si fa riferimento per rendere più efficace la descrizione è stata realizzata per mostrare un possibile utilizzo della libreria: in sintesi, in essa si mostrano le immagini rilevate dal sensore di profondità evidenziando gli utenti presenti rispetto all'ambiente loro circostante, sui quali viene poi disegnata una bolla di un colore diverso a seconda dello stato di avanzamento nell'analisi della loro attenzione. Sono inoltre presenti due pannelli contenenti informazioni sulla scena: in quello in alto vengono indicati il valore del livello di attenzione di ciascun utente analizzato, mentre in basso si mostrano informazioni di sintesi quali il numero di osservatori considerati attenti presenti in quel momento all'interno della scena. Bisogna inoltre riportare che per questa applicazione vengono considerati attenti gli utenti che sono rimasti di fronte allo schermo per più di 1,5s, ad una distanza massima di 2m e con un livello di attenzione superiore a 60: questi parametri sono stati configurati grazie al file di configurazione. In particolare:

- una bolla di colore rosso viene disegnata per utenti per cui ancora non si sta analizzando il livello di attenzione, in attesa di cominciarne il tracking.

- una bolla di colore azzurro per utenti per cui si sta analizzando il livello di attenzione (reso perciò visibile nel pannello in alto) ma che non sono

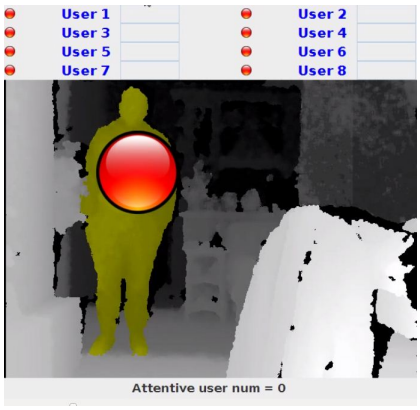

Figura 3.1: Un utente per cui ancora non è cominciata l'analisi dell'attenzione

ancora considerati realmente attenti, in quanto non soddisfano i criteri decisi in precedenza

- una bolla di colore verde per gli utenti considerati attenti secondo i parametri definiti: una volta guadagnato lo status di utente attento, lo si può perdere solamente lasciando la scena.

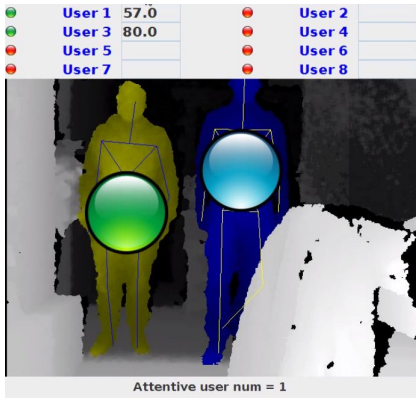

Figura 3.2: Due utenti di cui uno è attento e l'altro no

3.2.3 Elementi principali di implementazione

Questo esempio, seppur semplice, risulta importante poiché mostra come possono essere sfruttati i principali strumenti offerti dalla libreria. In particolare vengono qui proposti gli stralci di codice più significativi.

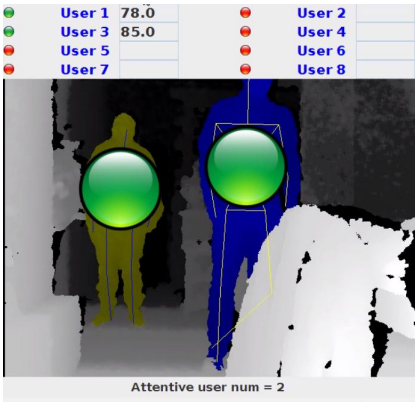

Figura 3.3: Due utenti entrambi considerati attenti

```
KinectManager k = KinectManager.getInstance();
SkeletonManager skelMan = SkeletonManager.getInstance(k);
AttentionManager attMan = new AttentionManager(skelMan);

//starts the tracking when the skeleton calibration is completed
skelMan.addTrackingStartedObserver(
                  new TrackingStartedObserver(attMan));

//initialize the core thread
final KinectAttentionTracker t1=
         new KinectAttentionTracker(k, skelMan, attMan);
```

In questo primo snippet di codice si mostra come istanziare i vari manager e come impostare l'osservatore per far partire l'analisi dell'attenzione non appena è terminata la fase di calibrazione per l'utente.

```
Object[] skelUsers = actualUserSet.toArray();
for(Object user : skelUsers){//update for every user
    int userId =(Integer.parseInt((String)user));

    if(skelMan.isSkeletonTracking(userId)){
    //if the skeleton is in tracking, update it
    skelMan.updateUserSkeletonPosition(userId);

     if(attMan.isTrackingStarted(userId)){
             attMan.userDataReady(userId,
             skelMan.getJointList().get(userId),
                    kinMan.getUserCoM(userId));
```

```
            }
        }else{
            continue;
        }
    }
```

In quest'altro esempio, tratto dal metodo `behaviour()` viene mostrato come occuparsi degli scheletri ad ogni ciclo di ricezione dei dati dal sensore.

```
    protected void lostUserDetected(int userId){
        //remove the skeleton from the hashmap
        try {
            skelMan.getJointList().remove(userId);
            if(skelMan.isSkeletonTracking(userId)){
                skelMan.stopTracking(userId);
            }
            log(" LOST USER ID is "+ userId);

            attMan.lostUser(userId);
            attPanel.lostUser(userId-1);
        } catch (StatusException e) {
            e.printStackTrace();
        }
    }

    protected void newUserDetected(int userId){
        try {
            skelMan.requestSkeletonCalibration(userId, true);
            log(" NEW USER ID is "+ userId);
        } catch (StatusException e) {
            e.printStackTrace();
        }
    }
```

In quest'ultimo esempio vengono invece mostrate le due implementazioni adottate per `newUserDetected()` e `lostUserDetected()`: in questo caso ogni volta che viene notificato l'arrivo di un nuovo utente se ne fa partire il tracking, mentre quando se ne perde uno bisogna fermare il thread addetto all'analisi per l'utente con il dato ID.

Per quanto riguarda la GUI dell'applicazione, è stata estesa la classe `UserWithSkeletonPanel` per averne una in grado di mostrare sia lo scheletro degli utenti sia le bolle di vario colore: per fare ciò è stato sovrascritto il metodo `drawAtSingleUser`, che permette di disegnare quel che si vuole nel centro di massa di un dato utente.

É da ricordare inoltre che l'esempio è stato realizzato tramite la creazione di 3 sole classi, di cui una rappresenta il pannello da mostrare nella GUI, mentre un'altra ha la sola funzione di main() in cui si configurano i vari elementi e si fa partire l'applicazione: il core di questa applicazione è quindi sicuramente `KinectAttentionTracker`, che si occupa di gestire l'aggiornamento sia del pannello, sia degli scheletri che dei manager.

3.3 Interazione naturale

Lo scopo di questa libreria è quello di fornire dei meccanismi flessibili e riusabili per permettere un'interazione naturale tra schermo e utente. Visto che nell'implementazione sono stati utilizzati in maniera importante i meccanismi introdotti con il middleware NITE per il riconoscimento di gestures, vale la pena darne una sintetica descrizione.

3.3.1 NITE 1.5.2.23

NITE è un middleware che opera sulla libreria OpenNI che mette a disposizione degli algoritmi per facilitare i programmatori non solo nell'individuazione di un punto che identifichi il palmo della mano, ma anche nell'elaborazione dei dati provenienti dal sensore in maniera tale da trarre delle informazioni più accurate sugli spostamenti effettuati dalla mano rispetto a quelle offerte dal tracciamento di OpenNI. Per elaborare ed analizzare i dati inerenti alla mano e al suo spostamento al fine di individuare specifici movimenti vengono messi a disposizione dei particolari oggetti, chiamati controlli, che nello specifico sono:

- **Push Detector** rileva l'evento legato al gesto Push, che consiste nel muovere la mano verso il sensore;
- **Swipe Detector** gestisce gli eventi legati agli Swipe (verso l'alto, il basso, destra o sinistra), brevi movimenti della mano in una direzione dopo i quali essa si ferma, ad esempio utile per sfogliare le pagine di un documento;
- **Steady Detector** rileva il gesto Steady, che consiste nel tenere la mano praticamente ferma per un determinato lasso di tempo;
- **Wave Detector** si occupa della gesture Wave, che consiste in un movimento ondulatorio della mano, quale muoverla cambiando velocemente direzione (in genere sono necessari 4 cambiamenti di direzione);

- **Circle Detector** riconosce movimenti circolari (completi) della mano sia in senso orario (positivo) che in senso antiorario (negativo);
- **SelectableSlider1D** riconosce uno scorrimento della mano in una delle tre direzioni degli assi X,Y,Z, cioè sinistra-destra, sopra-sotto, vicino-lontano.
- **SelectableSlider2D** riconosce uno scorrimento della mano nel piano X-Y

Tutti questi oggetti estendono dall'oggetto Point Control il quale riceve gli hand point attivi ad ogni frame, tenendo in questo modo traccia di tutti i punti (rappresentati in coordinate rapportate allo schermo) dello spostamento della mano. Ogni sottoclasse analizza questi punti per verificare la rilevazione del movimento per cui è specializzata, e nel caso lo notifica grazie all'architettura ad eventi in cui questi controlli sono inseriti.

3.3.2 Struttura

La libreria è organizzata nei seguenti package:

- `it.unibo.kinect.gesture.control` che contiene la classe manager per le gestures
- `it.unibo.kinect.gesture.structure` che contiene le classi per creare delle GestureGUI, gli elementi grafici con i quali è possibile interagire tramite lo schermo
- `it.unibo.kinect.gesture.information` che contiene il modello delle informazioni riguardanti i GestureGUI
- `it.unibo.kinect.gesture.util` contiene le classi di utilità, tra le quali un pannello per mostrare la mano-cursore

Il package it.unibo.kinect.gesture.control

La classe implementata in questo package, `GestureManager`, è di sicuro quella di maggior importanza di tutta la libreria, in quanto si assume la responsabilità di interagire con i driver OpenNI e NITE per riconoscere i gesti significativi effettuati dall'utente. Scendendo più nei dettagli, si inizializzano `HandGenerator`, `GestureGenerator` e in particolare `SessionManager`, a cui si può specificare con quale gesture si vuole che l'utente inizi una sessione di

interazione: nel contesto di schermi pervasivi considerato, per rendere all'utente il più trasparente possibile questa interazione è stato scelto di iniziare la sessione con il gesto `RaiseHand`, in modo tale che essa cominci piuttosto naturalmente, visto che alzare la mano è un movimento che quasi sicuramente l'utente farà di fronte allo schermo per interagire con esso. Questa classe permette di specificare il comportamento da tenere a fronte di un particolare gesto dell'utente grazie alla delegazione: infatti si devono passare ai metodi per la configurazione del riconoscimento delle varie gestures gli ascoltatori che verranno poi risvegliati al momento della notifica, secondo il pattern Observer, i quali vengono poi configurati sui relativi oggetti *detector* (di cui si è descritto in 3.3.1), addetti ognuno ad un particolare gesto. Così vengono offerti metodi come `setWaveGestureHandler()` che permette di impostare il comportamento da tenere quando l'utente compie la gesture Wave, oppure `setRightSwipeGestureHandler()` che permette di specificare quello da tenere a fronte di uno swipe verso destra, e analogamente per tutti gli altri controlli implementati.

Da evidenziare anche il metodo `setPointControlHandler()`, che imposta gli observer per gli eventi generati dall'oggetto `PointControl`: può ricevere fino a 4 parametri in ingresso per gestire la creazione, l'aggiornamento, la cancellazione e la perdita del punto che rappresenta la mano presa in considerazione.

Infine, vi è anche la dichiarazione dell'interfaccia `IGestureGuiManager`, che deve essere implementata dagli oggetti che devono gestire GestureGUI: infatti in essa è presente il solo metodo `announcePress()`, invocato dagli elementi grafici sul loro gestore una volta che sono stati premuti.

Il package it.unibo.kinect.gesture.structure

In questo package sono state implementate le classi per la creazione degli elementi grafici di interazione con l'utente: a partire dalle considerazioni fatte nel progetto, è risultato piuttosto naturale implementare queste classi con una gerarchia, alla base della quale si trova la classe `GestureGUIPanel`, da cui poi le altre ereditano.

Più nel dettaglio, le classi maggiormente importanti sono:

- `GestureGUIPanel` che rappresenta il template astratto per un elemento grafico con cui interagire, e perciò ne incapsula gli elementi fondamentali del comportamento, descritti tramite il diagramma UML in figura 2.13. Per l'implementazione si è pensato di rendere evidenti le transizioni da uno stato dall'altro all'utente grazie al cambiamento dell'immagine che rappresenta sullo schermo il particolare controllo: il metodo

`updateState()` costituisce il core di questa classe in quanto controlla la posizione della mano in relazione allo schermo e in base a questa aggiorna lo stato del componente. Ad esempio, se il componente risulta essere attivo viene invocato il metodo `drawActive()`, che disegna sullo schermo l'immagine corrispondente all'oggetto nel suo stato attivo. Di default, sono previste una immagine per lo stato inattivo (`inactiveIm`) ed una per lo stato attivo (`activeIm`), mentre per lo stato in cui il controllo è premuto si disegna un piccolo cerchio rosso al centro dell'elemento: ereditando da questa classe è però facile cambiare questo comportamento, sovrascrivendo il metodo `drawPressed(Graphics g)`, che implementa le operazioni per mostrare che il controllo è stato premuto. Viene inoltre prevista la possibilità di disegnare un'etichetta sull'elemento grazie al metodo `drawLabel(Graphics g)`.

- `GGUIsManager` implementa invece una classe responsabile della gestione di gruppi di controlli grafici GestureGUI: ne gestisce infatti il posizionamento (permettendo di posizionarle ai lati o al centro dello schermo) memorizzandone i confini per determinare, di volta in volta, quale controllo è attivo. I metodi `locateComponents()`, che permette di memorizzare le coordinate nello schermo in cui sono stati posizionati gli elementi grafici, e `updateGGUIs(Point scrPt)`, che controlla con quale oggetto sta interagendo l'utente e lo aggiorna per renderlo attivo, risultano perciò essere fondamentali per questa classe.

Per questo framework sono stati implementati i controlli per bottoni, slider e dial, fornendone anche qualche esempio concreto direttamente utilizzabile nelle applicazioni: per questo è stata predisposta una classe factory (`StandardGGUIFactory`) che ne facilita la creazione tramite il metodo `getGGUIInstance()`. Ognuno di questo tipo di controlli ha perciò una relativa classe astratta che ne determina il comportamento e le informazioni che devono scambiare con il loro gestore: l'implementazione risulta spesso molto simile per cui, per motivi di sintesi, viene descritta solamente l'implementazione dei bottoni, da cui si possono evincere gli elementi principali che permettono poi di capire anche come sono stati realizzati gli altri controlli.

Un caso di GestureGUI: i bottoni

Gli elementi grafici di interazione per dei semplici bottoni sono stati realizzati secondo la struttura mostrata nel diagramma UML 3.4, da cui è possibile notare che i tipi di bottoni implementati sono stati 3: HomeButton utile per tornare ad una schermata principale, BackButton utile ad esempio

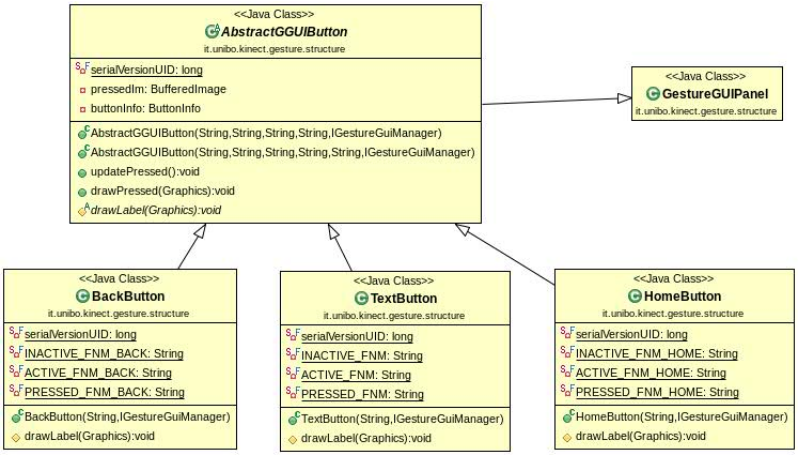

Figura 3.4: UML per l'implementazione di pulsanti che possono essere premuti tramite interfaccia naturale

per annullare l'azione precedente o tornare indietro nella navigazione, e un generico TextButton che rappresenta un pulsante con un testo.

Tutti questi bottoni ereditano dalla classe `AbstractGGUIButton` che incapsula perciò il comportamento generale di un pulsante: prevede un'immagine diversa da mostrare per ogni stato in modo da far capire in ogni momento all'utente se l'interazione sta andando a buon fine. Al costruttore vanno quindi indicati i path delle immagini da utilizzare (perciò il nome del file e se necessario anche della cartella che lo contiene).

Per quanto riguarda la specifica implementazione dei singoli bottoni, essa è risultata molto semplice:

- `TextButton` non fa altro che sovrascrivere il metodo che disegna l'etichetta per mostrare il testo voluto all'interno di un bottone di forma rettangolare grigio se inattivo, giallo se attivo e verde se premuto
- `HomeButton` mostra un bottone di forma squadrata con il simbolo Home al centro; di colore grigio se inattivo, giallo se attivo e verde se premuto
- `BackButton` mostra un bottone di forma squadrata con una freccia che sembra tornare indietro al centro; di colore grigio se inattivo, giallo se attivo e verde se premuto

La cosa importante da ricordare è che per la creazione di questi bottoni, visto che le immagini sono inserite come risorse all'interno del framework, il programmatore non deve neppure specificare i path delle immagini, di cui si occupa il costruttore dato che ne conosce le precise locazioni.

Il package it.unibo.kinect.gesture.information

In questo package vengono implementate le classi riguardanti le informazioni riguardanti le GestureGUI secondo la gerarchia mostrata in figura 2.15.

Le informazioni più rilevanti sono che le classi specializzate per le informazioni di uno specifico controllo possono fornire delle informazioni aggiuntive su di esso, ad esempio:

- `DialInfo` specifica l'angolo con cui l'utente ha ruotato la manopola, utile ad esempio per ruotare un'immagine
- `SliderInfo` specifica il valore a cui l'utente ha impostato lo slider, utile per esempio per determinare lo zoom per un'immagine

Il package it.unibo.kinect.gesture.util

Tra le classi di utilità implementate per questa libreria vi è sicuramente `CursorPanel`, costituita da un pannello Swing trasparente in cui viene disegnata una piccola icona rappresentate una mano: esso può essere utilizzato per mostrare all'utente il tracciamento della sua mano e renderlo consapevole della sua posizione all'interno dello schermo, in modo da rendere naturale ed intuitiva l'interazione con GestureGUI.

3.3.3 Esempio di riferimento

Figura 3.5: Prima schermata dell'applicazione GestureTutorial

L'applicazione a cui si fa riferimento per la descrizione dell'implementazione di questa libreria mostra uno dei tantissimi modi di utilizzare gli

strumenti da essa offerti: in sintesi essa è composta da due schermate con cui l'utente può interagire tramite gesti e GestureGUI. Nella prima schermata viene mostrata a tutto schermo l'immagine a colori ripresa dal Kinect, sopra la quale sono disegnati dei bottoni che permettono all'utente di interagire con lo schermo, come mostrato in figura 3.5: la mano dell'utente è rappresentata con un cursore all'interno dello schermo e, posizionandola all'interno del pulsante tondo rosso si passa alla seconda schermata, mentre con il bottone contrassegnato dall'etichetta EXIT si chiude l'applicazione. L'utente può passare alla seconda schermata anche effettuando una Push gesture. Nella seconda schermata di interazione, come si nota dalla figura

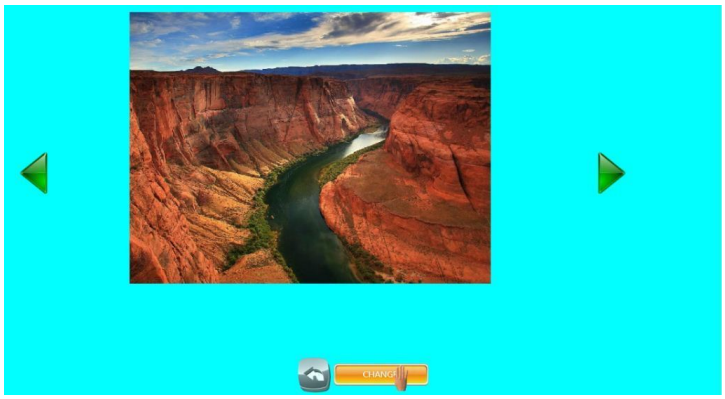

Figura 3.6: Seconda schermata dell'applicazione GestureTutorial

3.6, invece l'utente può sfruttare le gesture Swipe a sinistra e a destra per cambiare l'immagine visualizzata all'interno di uno slideshow di fotografie predefinite, inoltre tramite il pulsante CHANGE può cambiare il colore dello sfondo della schermata, mentre vi è un pulsante BACK per tornare alla schermata precedente. Tramite la Circle gesture l'utente può inoltre chiudere l'applicazione direttamente dalla seconda schermata.

3.3.4 Elementi principali di implementazione

L'applicazione mostrata combina tutti gli elementi fondamentali offerti dalla libreria, di cui si dà una sintetica descrizione proponendo gli stralci di codice più significativi.

Il main per l'applicazione è realizzato con pochissime righe di codice, che però mostrano come configurare i manager con cui si ha che a fare.

```
public static void main(String[] args) {
    KinectManager k = KinectManager.getInstance();
```

```
        GestureManager gestureMan =
                        GestureManager.getInstance(k);
        KinectGesturesTracker t =
             new KinectGesturesTracker(k, gestureMan);
        new Thread(t).start();
    }
```

La classe principale dell'applicazione è `KinectGesturesTracker`, che oltre a rilevare i dati dal Kinect, configura la grafica dell'applicazione.

Le operazioni da eseguire sui dati provenienti dal sensore sono implementate, come al solito, nel metodo `behaviour()`:

```
    protected void behaviour(TreeSet<String> actualUserSet) {
        try {
            sessionMan.update(context);
            if(state == State.HOME){
                colorPan.updateImage();
                layeredHomePan.repaint();
                colorPan.repaint();
            } else if(state == State.MAIN){
                layeredMainPan.repaint();
            }
        } catch (StatusException e1) {
            e1.printStackTrace();
        }
    }
```

Si può notare come tramite la chiamata `sessionMan.update(context)` si aggiorna il manager di sessione NITE per le gestures e come vengono aggiornati i pannelli della grafica a seconda dello stato in cui ci si trova.
Per quanto riguarda la grafica dell'applicazione, sono state utilizzate le API messe a disposizione dalla libreria Swing, tra le quali spiccano per importanza nell'utilizzo:

- `CardLayout`, che permette di far condividere lo stesso spazio sullo schermo a due o più `JPanel` differenti e di visualizzarne solo uno alla volta tramite appositi metodi, è stato utilizzato per passare da una schermata all'altra;

- `JLayeredPane` invece permette di sovrapporre più pannelli, quasi come infilandoli in una pila, è stata utilizzata per disegnare sopra alle schermate i controlli grafici e la mano-cursore che segue quella dell'utente.

Per mostrare la mano dell'utente è stata utilizzata la classe `CursorPanel`, mentre gli elementi grafici sono stati creati utilizzando la factory, come si può notare nello stralcio di codice proposto qui di seguito, che mostra come è stata creata la parte grafica della prima schermata, in cui al `FullScreenColorPanel` che contiene l'immagine a colori ripresa dal Kinect vengono sovrapposti prima i pulsanti, e poi il pannello con il cursore.

```
handPanel = new CursorPanel(width, width);
structPan = new JPanel();
structPan.setLayout(new CardLayout());
c.add(structPan);

layeredHomePan = new JLayeredPane();
layeredHomePan.setBounds(0, 0, width, height);
layeredHomePan.setPreferredSize(screenD);
structPan.add(layeredHomePan, HOME_PANEL);

colorPan = new FullScreenColorPanel(kinMan);
layeredHomePan.add(colorPan, 0, 0);

homeButtons = new GGUIsManager(width, height);
layeredHomePan.add(homeButtons.getPanel(), 1, 0);
GestureGUIPanel button =
    StandardGGUIFactory.getGGUIInstance(
       StandardGGUIFactory.BUTTON_TEXT, this, "EXIT");
homeButtons.add(button);
GestureGUIPanel button2 = new PressButton("app", this);
homeButtons.add(button2);

homeButtons.setEdge(Compass.CENTER);

layeredHomePan.add(handPanel, 2, 0);
```

Per l'interazione con le gesture, viene riportato solo il codice riguardante la configurazione della gesture Push, visto che le altre sono molto simili:

```
//push configuration --> to pass from home to main
gestureMan.setPushGestureHandler(
              new IObserver<VelocityAngleEventArgs>() {
    @Override
    public void update(IObservable<VelocityAngleEventArgs> arg0,
            VelocityAngleEventArgs arg1) {
        if(state == State.HOME){
            mainState();
```

```
            }
        }
    });
```

Si può quindi notare come una volta rilevata la gesture si passa, tramite il metodo `mainState()`, dalla prima alla seconda schermata.

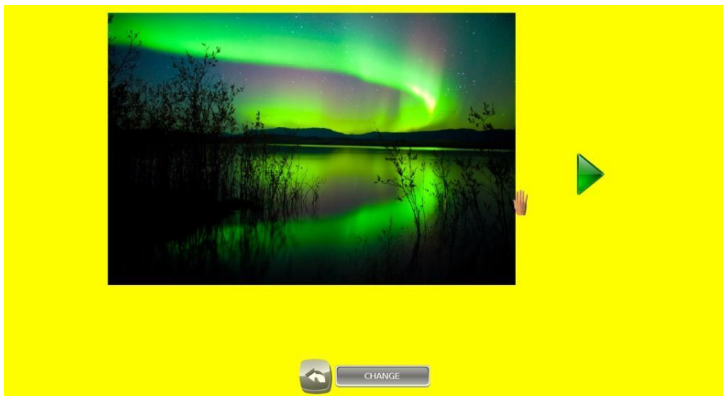

Figura 3.7: Il colore dello sfondo è cambiato, così come la fotografia mostrata

Infine viene riportato il codice del metodo `announcePress()`, in cui prima si capisce quale controllo ha generato la notifica tramite un controllo sulle sue informazioni, e poi si esegue la relativa operazione: nel caso del pulsante CHANGE ad esempio si invoca sul pannello della seconda schermata il metodo `changeBackgroundColor()` solo se l'utente sta effettivamente interagendo con quella schermata, e si ha come effetto il cambio del colore di sfondo (figura 3.7).

```
public void announcePress(ComponentInfo ci) {
    String name = ci.getName();
    if(state == State.HOME){
        switch (name) {
        case "EXIT":
            me.stop();
            break;
        case "app":
            mainState();
            break;
        default:
            break;
        }
    } else if(state == State.MAIN){
```

```
            switch (name) {
            case "back":
                homeState();
                break;
            case "CHANGE":
                mainPanel.changeBackgroundColor();
                break;
            default:
                break;
            }
        }
    }
```

Per mostrare invece quanto sia facile creare degli elementi grafici personalizzati utilizzando l'architettura messa a disposizione, è stato implementato un bottone (`PressButton`) che da rosso passa a verde quando attivo o premuto: per la realizzazione è stato sufficiente realizzare una classe con i riferimenti alle immagini del bottone. Come si vede dallo screenshot in figura

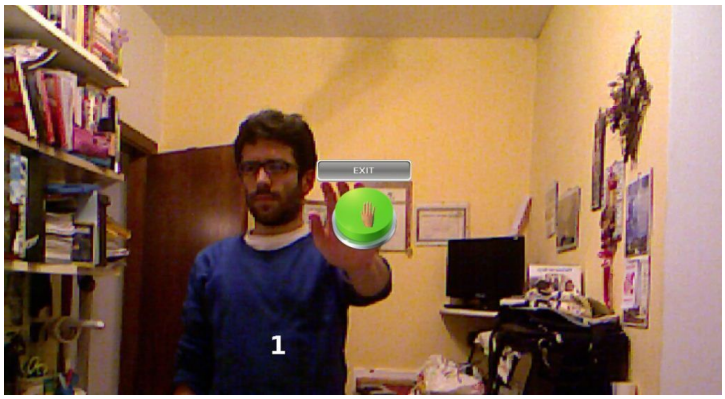

Figura 3.8: Pressione di un pulsante personalizzato

3.8 infatti il pulsante, prima rosso (figura 3.5), diventa di colore verde quando la mano dell'utente (rappresentata con l'icona di una mano sullo schermo) vi si posizione sopra.

L'esempio è stato realizzato con sole 4 classi, il cui codice serve in buona parte a configurare la parte grafica dell'applicazione facente riferimento a Swing: essa ha rappresentato lo scoglio più difficile da superare per questo esempio, visto che utilizzando il framework l'interazione con il sensore nella configurazione degli handler per le gestures è risultata molto semplificata, permettendo di spendere maggiori risorse, anche temporali, allo sviluppo

della logica dell'applicazione e alla sua resa grafica, che in un sistema per schermi pervasivi non ha un peso banale.

3.4 Identificazione dell'utente

Lo scopo da raggiungere con l'implementazione di questa libreria è quello di fornire un componente software riusabile per identificare un osservatore che sta di fronte allo schermo: come descritto in 2.5 si può raggiungere tale obiettivo sfruttando le potenzialità offerte dai codici QR e dagli smartphone che perciò diventano i componenti di base di questo layer del framework.

La struttura è differente dalle altre librerie presentate poiché non si deve comunicare con il sensore, ma con un altra macchina dotata di capacità computazionale tramite la rete, più precisamente con una connessione TCP. Viene quindi descritta sia la struttura lato server dell'applicazione, che rappresenta il componente software interno al framework, sia quella dell'implementazione di una applicazione per smartphone Android sviluppata ad hoc, lato client.

3.4.1 Struttura lato server

La libreria è organizzata in due soli package: `it.unibo.qrcode.control`, in cui viene implementata la classe che fornisce un template per il server di richieste di identificazione, e `it.unibo.qrcode.util`, in cui sono riposte le classi di utilità generali.

La classe `RecognizerServer` del package `it.unibo.qrcode.control` implementa il comportamento generale di un server di richieste di identificazione tramite lo scan di un codice QR, e quindi rappresenta la classe principale di questa libreria: il suo nucleo risiede nel metodo `run()`, la cui implementazione è stata realizzata seguendo le specifiche progettuali viste nel diagramma 2.18. Terminata una identificazione, il codice di sessione e perciò anche il codice QR vengono rigenerati per servirne una nuova, perlomeno fin quando all'oggetto non viene comunicato di fermarsi tramite l'invocazione del metodo `stop()`. Esso offre diversi costruttori per dare facoltà al programmatore di decidere i parametri con cui costruire il server quali indirizzo IP, numero di porta e grandezza del codice da generare. Le comunicazioni in rete sono state realizzate con il supporto alle `Socket` fornito da Java, mentre il file XML contenente le informazione dell'utente viene ricevuto in forma di stringa.

Per aumentare la riusabilità del codice scritto e la sua leggibilità sono state realizzate delle librerie contenenti delle funzioni utili, raccolte nel package `it.unibo.qrcode.util`:

- `NetUtilities` contiene funzionalità utili come quelle per ricavare il proprio indirizzo IP dalle interfacce di rete oppure quella di generare una stringa casuale, utile per essere utilizzata come codice di sessione;

- `QRCodeUtil` si interfaccia alla libreria per utilizzare codici QR per offrire funzionalità quale l'ottenimento del codice come immagine a partire da una stringa, il suo salvataggio in un file oppure direttamente un pannello contenente il codice QR generato.

- `XmlUtilities` permette di ottenere un file XML a partire dalla stringa in cui è stato encodificato, e risulta utile nel contesto del framework in quanto il file contenente le informazioni dell'utente che arriva dalla comunicazione in rete è proprio in questa forma.

Per utilizzare questo componente software è sufficiente implementare una classe che, ereditando da `RecognizerServer` funga da server nella particolare applicazione per cui è stata realizzata: infatti il programmatore deve implementare i metodi astratti `showQRCode()` e `showUserInfo()` secondo la particolare logica applicativa.

3.4.2 Struttura lato client

L'applicazione pensata per fornire lato client un'interfaccia semplice per consentire all'utente l'identificazione tramite l'interazione con un codice QR, da eseguire su dispositivo mobile Android, non ha una struttura complicata in quanto formata da 3 sole Activity:

- `MainActivity`, che costituisce la schermata principale in cui l'utente potrà far partire l'identificazione;

- `SaveActivity`, che consente all'utente di modificare e salvare le proprie informazioni personali;

- `ConnectActivity`, che fa da 'background' alle operazioni di connessione e comunicazione, esplicitando all'utente che si è in fase di scambio dei dati, chiusa non appena le operazioni di riconoscimento finiscono, sia positivamente che non; da notare che le operazioni che coinvolgono scambio di dati in rete vengono eseguite in un worker Thread diverso da quello principale della App, in modo da non bloccarlo in caso qualcosa vada storto nella comunicazione.

Figura 3.9: Screenshot di MainActivity

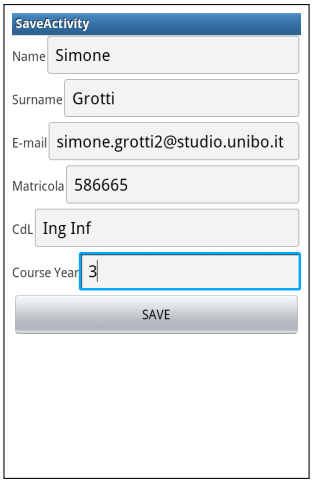

Figura 3.10: Screenshot di SaveActivity

Funzionamento

Per rendere l'interazione il più veloce possibile, una volta configurati i propri dati personali, per eseguire l'identificazione l'utente non deve fare altro che avviare l'applicazione e premere il pulsante che permette l'avvio della app Barcode Scanner: una volta inquadarato il codice QR, essa provvede a decodificare la stringa in esso contenuto e a passarla alla applicazione che l'ha richiesta. In questo modo l'applicazione sviluppata non deve occuparsi della decodifica del codice, già fornita dalle funzionalità di Barcode

Scanner, ma si concentra sul trattamento di questo informazioni: infatti nel codice QR analizzato vi deve essere encodificata una stringa del tipo IPaddress:port?sessionID, da cui si ricavano le informazioni che permettono al device di raggiungere tramite TCP il server e quindi di potergli inviare i propri dati, previo riconoscimento tramite il sessionID che può conoscere solamente chi è davanti allo schermo e ha analizzato il codice, in quanto generato casualmente. Dal punto di vista dell'utente basta per cui effettuare lo scanning tramite l'applicazione per autenticarsi: l'operazione risulta anche piuttosto veloce, poiché l'analisi del codice una volta centrato con la fotocamera è molto rapida. Durante le operazioni di riconoscimento, all'utente vengono notificati gli eventi rilevanti, quali ad esempio il successo o il fallimento con le relative cause, tramite degli appositi Dialog. Inoltre, visto che i propri dati personali vengono visualizzati nella schermata principale dell'applicazione, ad suo avvio l'utente può immediatamente controllare se i suoi dati sono quelli giusti e, nel caso non lo fossero, provvedere ad una modifica aprendo il Menu con l'apposito tasto.

Integrazione con ZXing BarcodeScanner

Integrare le funzionalità di scan e decodifica di codici QR tramite la fotocamera del device, offerte dall'applicazione BarcodeScanner realizzata da ZXing e scaricabile dal market Android, è stato semplice grazie all'uso delle funzioni contenute nel jar di integrazione[3]: infatti esse permettono con pochissime istruzioni di far partire Barcode Scanner da un'altra activity, come si può vedere nel codice riportato qui di seguito

```
IntentIntegrator integrator =
            new IntentIntegrator(MainActivity.this);
integrator.initiateScan();
```

Queste sono infatti le operazioni eseguite alla pressione del pulsante che permette di avviare lo scan del codice. Anche il trattamento dei dati viene gestito grazie a questa libreria: infatti quando Barcode Scanner termina, il risultato dell'analisi del codice può essere gestito tramite il metodo `onActivityResult()`, in cui si utilizzano i metodi contenuti nella libreria fornita da ZXing per recuperare la stringa ottenuta, passata poi a `ConnectActivity` insieme ai dati personali dell'utente (recuperati dal file XML all'avvio dell'applicazione per mostrarli nella schermata principale) tramite il meccanismo Intent di Android, con cui si riesce a far partire

[3]android-integration-2.3-20130514.113451-1.jar, scaricabile dal sito `https://code.google.com/p/zxing/`

una Activity 'figlia'. L'utente può inoltre annullare lo scan anche durante l'esecuzione di Barcode Scanner semplicemente premendo il tasto Indietro: in questo modo Barcode Scanner viene terminata in modo da poter gestire correttamente questo caso dalla Activity 'chiamante'.

Comunicazioni in rete

La Activity addetta alle comunicazioni di rete fornisce un'interfaccia tra l'utente e il thread che si occupa delle comunicazioni: grazie all'utilizzo di un Handler esso può interagire col flusso di controllo principale dell'applicazione, chiamato *UI thread*, lasciato libero dall'onere delle comunicazioni per evitare che la grafica si blocchi in caso di un'operazione di rete particolarmente lunga, evitando il dialog di errore application not responding. Questo approccio permette inoltre di poter cambiare in futuro anche la modalità di comunicazione con il server: se per esempio si volesse utilizzare la tecnologia bluetooth, basterebbe implementare un nuovo thread che utilizzi le funzionalità bluetooth per la comunicazione, fornire all'applicazione i permessi necessari e cambiare l'invocazione al thread in questa Activity per ottenere lo stesso risultato in modo trasparente all'utente.

Per utilizzare le comunicazioni di rete come quella in TCP l'applicazione necessita dei permessi per accedere alla connessione internet del device, che significa aggiungere al file Manifest dell'applicazione la riga:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

É importante ricordare inoltre che tutte le operazioni di comunicazione in rete risultano trasparenti all'utente, che viene notificato solo in caso di successo o di fallimento tramite appositi dialog.

Salvataggio dei dati

Il salvataggio dei dati personali avviene tramite l'utilizzo di SaveActivity, che preleva idati specificati dall'utente e li salva usando le API Android in un file XML interno alla memoria del device con la sintassi mostrata qui di seguito, ottenuta grazie all'utilizzo di strumenti messi a disposizione dalla piattaforma di sviluppo Android, quali XmlSerializer:

```
<student>
    <name>Simone</name>
    <surname>Grotti</surname>
    <email>simone.grotti2@studio.unibo.it</email>
    <matr>586665</matr>
    <cdl>Ing. Informatica</cdl>
```

```
    <year>3</year>
</student>
```

3.4.3 Esempio di riferimento

Figura 3.11: Screenshot iniziale dell'esempio

L'applicazione utilizzata come riferimento per la descrizione di questa libreria permette l'identificazione simultanea di due utenti tramite due diversi codici QR (figura 3.11): per ognuno dei codici mostrati infatti viene eseguito un thread che svolge le funzioni di server per le richieste in arrivo.

Figura 3.12: Screenshot dopo l'identificazione di un solo codice QR

Avvenuta l'identificazione, il relativo codice scompare per lasciare posto alle informazioni dell'utente che ha eseguito l'identificazione, come si può notare dalla figura 3.12, in cui un solo codice è stato decodificato: lato

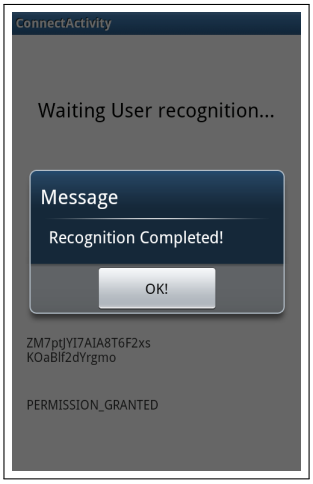

Figura 3.13: Schermata di un riconoscimento completato con successo

smartphone, completata l'autenticazione, l'utente viene notificato tramite un dialog che lo informa del successo dell'operazione (figura 3.13).

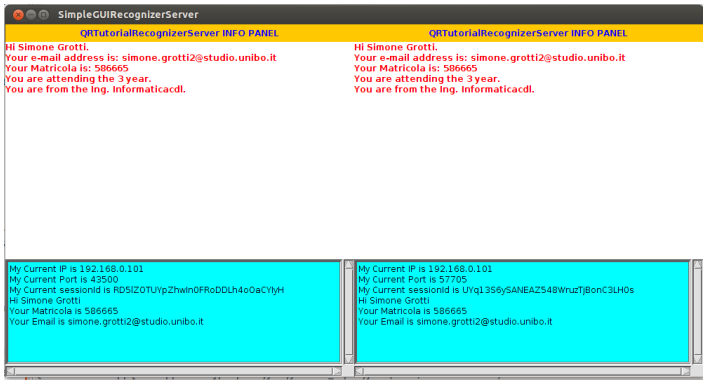

Figura 3.14: Screenshot in cui entrambi i codici sono stati decodificati

In questo esempio, realizzato per mostrare come funziona l'autenticazione, le informazioni rimangono visibili per qualche secondo prima di far spazio al nuovo codice QR generato dal server, che dimostra così di essere pronto per ricevere una nuova richiesta di identificazione.

3.4.4 Elementi principali di implementazione

L'applicazione realizzata è composta da due sole classi:

- `QRCodeTutorialApp` che contiene il main per l'applicazione, in cui si inizializzano i server e la GUI (un `JFrame`) riportato qui di seguito.

```
public static void main(String[] s){
    JFrame frame = new JFrame("SimpleGUIRecognizerServer");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel eastPanel = new JPanel(new BorderLayout());
    JPanel westPanel = new JPanel(new BorderLayout());

    Container c = frame.getContentPane();
    c.add(eastPanel, BorderLayout.EAST);
    c.add(westPanel, BorderLayout.WEST);
    TutorialRecognizerServer t1 =
            new TutorialRecognizerServer(frame, eastPanel);
    TutorialRecognizerServer t2 =
            new TutorialRecognizerServer(frame, westPanel);

    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);

    new Thread(t1).start();
    new Thread(t2).start();
}
```

- `TutorialRecognizerServer` che rappresenta lo specifico server per questa applicazione. Essa, oltre ad inizializzare la parte grafica aggiungendo anche un'area in cui vengono notificate all'utente delle informazioni riguardanti l'autenticazione in forma testuale, implementa anche i metodi astratti offerti dal template `RecognizerServer`. Il metodo `showQRCode` fa in modo che venga visualizzato nel pannello il codice QR generato, come si può notare dall'estratto riportato qui di seguito.

```
protected void showQRCode() {
    qrPanel = QRCodeUtil.getQrCodeInAPanel(myCode);
    switchPanel.remove(qrPanel);
    switchPanel.add(qrPanel, QRPANEL);
    switchPanel.validate();
    cl.show(switchPanel, QRPANEL);

    frame.pack();
    frame.validate();
```

```
        frame.setLocationRelativeTo(null);
    }
```

Invece il metodo `showUserInfo()` fa in modo che ad essere visualizzate siano le informazioni dell'utente ricavate dall'identificazione, e aspetta per 10 secondi prima di procedere con la generazione di un nuovo codice QR e quindi essere pronto per una nuova potenziale identificazione.

```
protected void showUserInfo(String name,
        String surname, String email,String mat,
        String cdl, String year) {
    log("Hi "+name+" "+surname);
    log("Your Matricola is "+mat);
    log("Your Email is "+email);

    nameLabel.setText("Hi "+name+" "+surname+".");
    emLabel.setText("Your e-mail address is: "+email);
    matLabel.setText("Your Matricola is: "+mat);
    yearLabel.setText("You are attending the "+year+" year.");
    cdlLabel.setText("You are from the "+cdl+"cdl.");

    bkgrPanel.validate();

    cl.show(switchPanel, USERPANEL);
    frame.pack();
    frame.validate();
    frame.setLocationRelativeTo(null);

    try {
        Thread.sleep(10000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Quest'esempio, realizzato implementando due sole classi di poche righe di codice, mostra la semplicità con cui è possibile utilizzare questo componente e la sua versatilità, caratteristiche che lo rendono riusabile in vari contesti.

3.5 Deployment del framework

Vengono qui di seguito descritte le principali informazioni riguardanti il deployment delle librerie: oltre ad indicare le fonti a cui è possibile reperire

il codice sviluppato si descrive brevemente l'ambiente di lavoro in cui lo sviluppo è stato effettuato.

3.5.1 Ambiente di lavoro

Il progetto è stato realizzato interamente in Java, su una macchina il cui sistema operativo è Linux Ubuntu 12.10. Per ottenere un ambiente di lavoro che permetta di interagire con il Kinect, è necessario scaricare i driver OpenNI e NITE ed installarli sulla macchina. Qui di seguito vengono perciò riportate le informazioni principali riguardanti l'installazione:

- tramite una shell e il comando `sudo apt-get install` bisogna assicurarsi che la propria macchina abbia installati `libusb-1.0.0-dev`, `freeglut3-dev`, `python`, `build-essential`

- scaricare i driver OpenNI (versione 1.5.4.0) e NITE (versione 1.5.2.23) per la propria macchina (32 o 64 bit) dal sito ufficiale OpenNI `http://www.openni.org/openni-sdk/openni-sdk-history-2/`

- scaricare i driver per il sensore (SensorKinect) dalla pagina `https://github.com/avin2/SensorKinect`: i driver da installare si trovano all'interno del file .zip nella cartella Bin/

- installare i driver rigorosamente nell'ordine OpenNI - SensorKinect - NITE: per l'installazione utilizzare la shell ed avviare gli script di installazione presenti nelle cartelle dei driver (comando `sudo ./install.sh` dopo essersi posizionati all'interno della relativa cartella)

- per testare se l'installazione è andata a buon fine, provare uno degli esempi forniti con i driver di OpenNI e NITE, dopo aver collegato il Kinect alla corrente e alla macchina

Per sviluppare nuove applicazioni utilizzando, ad esempio, un IDE quale Eclipse bisogna fornire nelle indicazioni di BuilPath per il progetto, oltre alle librerie di OpenNI e NITE, anche le native libraries da utilizzare per interagire con il sensore: visto che questi file (.jni) sono sparsi per le cartelle di installazione dei driver, si è pensato di raggrupparle in un'unica cartella, per evitare rischi di errori nell'impostare i giusti riferimenti. La raccolta dei file per le native libraries è quindi scaricabile dal repository del prototipo realizzato, di cui si parla nel prossimo capitolo.

3.5.2 Deployment delle librerie

Tutto il codice scritto per questa tesi è stato reso disponibile online in vari repository:

- **kinectBasic**, libreria di base per l'utilizzo di Kinect all'indirizzo `https://bitbucket.org/smgrotti/kinectbasiclibrary`
- **kinectAttention**, libreria per il riconoscimento dell'attenzione all'indirizzo `https://bitbucket.org/smgrotti/kinectattentionlibrary`, mentre l'applicazione tutorial per questa libreria è disponibile all'indirizzo `https://bitbucket.org/smgrotti/attentionkinecttutorial`
- **kinectGestures**, libreria per il riconoscimento delle gestures all'indirizzo `https://bitbucket.org/smgrotti/kinectgestureslibrary`, mentre l'applicazione tutorial per questa libreria è disponibile all'indirizzo `https://bitbucket.org/smgrotti/gestureskinecttutorial`
- **QRCodeRecognizer**, libreria per l'identificazione tramite codice QR all'indirizzo `https://bitbucket.org/smgrotti/qrcodelibrary`, mentre l'applicazione tutorial per questa libreria è disponibile all'indirizzo `https://bitbucket.org/smgrotti/qrcodetutorial` e l'applicazione Android da installare sullo device mobile è reperibile all'indirizzo `https://bitbucket.org/smgrotti/userrecognizer`

In questi repository è presente sia il codice che la relativa documentazione Javadoc e, nel caso delle librerie, anche il file `.jar` da importare per creare nuove applicazioni che sfruttino le potenzialità offerte da questo framework.

Capitolo 4

Caso di Studio: uno schermo riconoscitore

In questo capitolo viene presentato il prototipo di schermo pervasivo adattativo realizzato sfruttando le funzionalità messe a disposizione dal framework descritto nei capitoli 2 e 3.

4.1 Scenario applicativo

Il prototipo da realizzare rappresenta una possibile applicazione di uno schermo pervasivo adattativo calato nel contesto accademico (nel dettaglio quello di Alma Mater Studiorum - Università di Bologna, per la Seconda Facoltà di Ingegneria con sede a Cesena): più precisamente si simula uno schermo che permetta l'interazione con due o più studenti offrendo delle informazioni di loro interesse, quali per esempio i prossimi appelli d'esame a cui uno studente può iscriversi, le ultime variazioni all'orario delle lezioni e l'orario della settimana personalizzato in base all'anno di corso frequentato. Questo progetto si ispira agli schermi già presenti in facoltà che però mostrano solamente l'orario, senza adattare i propri contenuti sulla base delle informazioni sull'ambiente in cui sono immersi. Infatti è proprio questa la caratteristica chiave della programmazione pervasiva: la capacità computazionale (in questo caso quella posseduta dallo schermo) permea l'ambiente e si integra con esso, ricavandone dati ed informazioni utili per il proprio funzionamento.

Il prototipo di schermo da realizzare deve soddisfare i requisiti richiesti dai seguenti scenari:

- offrire la possibilità ad un utente interessato di identificarsi per ottenere delle informazioni personalizzate

- permettere ad un secondo utente oltre a quello già identificato di autenticarsi ed accedere anch'egli a contenuti di suo interesse
- nel caso in cui si rilevasse l'interesse verso lo schermo da parte di altre persone rispetto alle due sopra citate, offrire dei contenuti ritenuti di interesse generale

Il sistema deve inoltre supportare modalità di interazione naturale con l'utente anche al fine di massimizzare la trasparenza nel suo utilizzo.

4.2 Elementi principali di progetto

Risulta evidente dall'analisi dei requisiti del prototipo che le funzionalità offerte dal framework realizzato risultano fondamentali:

- per captare l'interesse degli utenti nei confronti dei suoi contenuti, si possono aggiungere le funzionalità offerte dalla libreria per il riconoscimento dell'attenzione
- la libreria per l'interazione naturale permette di utilizzare le gestures come meccanismo di interazione tra utente e schermo
- l'identificazione dell'utente e la raccolta di sue informazioni aggiuntive può invece essere implementata grazie alla libreria che sfrutta i codici QR

Si intuisce come l'integrazione di questi componenti software permette di arrivare alla realizzazione del sistema considerato, il cui sviluppo dovrebbe risultare semplificato per l'utilizzo dei concetti e degli strumenti offerti dal framework.

4.2.1 Diagramma a stati

Per il sistema in questione è stato progettato due diagrammi a stati che ne catturano le dinamiche di interazione con gli utenti. Nel primo, in figura 4.1, si rendono espliciti i possibili stati in cui lo schermo si può trovare.

Inizialmente lo schermo si trova in `IDLE`, ossia uno stato di riposo in cui viene mostrata una schermata contenente il logo della facoltà (figura 4.3) e si aspetta l'attenzione di un utente per proporgli l'identificazione ed offrirgli così dei contenuti. Non appena lo schermo rileva che una persona gli sta porgendo attenzione passa nello stato `REC WAIT`, in cui viene proposto all'utente, che da qui in poi viene considerato come utente principale, di

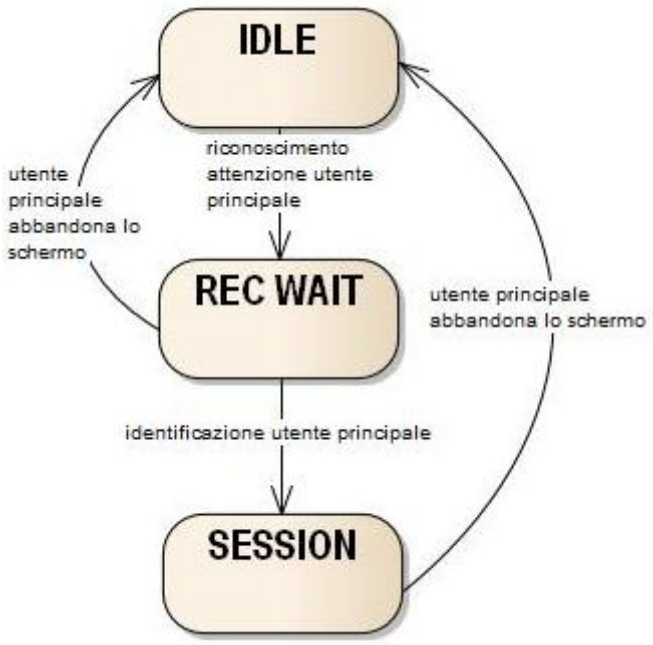

Figura 4.1: Diagramma degli stati per lo schermo

identificarsi per accedere ai contenuti mostrando il codice QR da decodificare con la applicazione smartphone (figura 4.4).

Una volta avvenuta l'identificazione lo schermo entra nello stato `SESSION`, che rappresenta una sessione di interazione: in questo stato si possono avere diversi tipi di sessione in base al numero degli osservatori che porgono attenzione allo schermo.

Più nel dettaglio, come mostrato dalla figura 4.2, i possibili tipi di sessione sono:

- `NO SESSION`, cioè non si è in sessione;
- `SESSION MAIN` rappresenta la sessione in cui è il solo utente principale ad interagire con lo schermo e ad essere interessato a suoi contenuti;
- `SESSION MAIN SIDE WAIT` è la sessione in cui oltre all'utente principale, è stata rilevata l'attenzione di un altro utente, da qui chiamato utente secondario, a cui viene perciò proposta l'autenticazione per accedere ai contenuti;

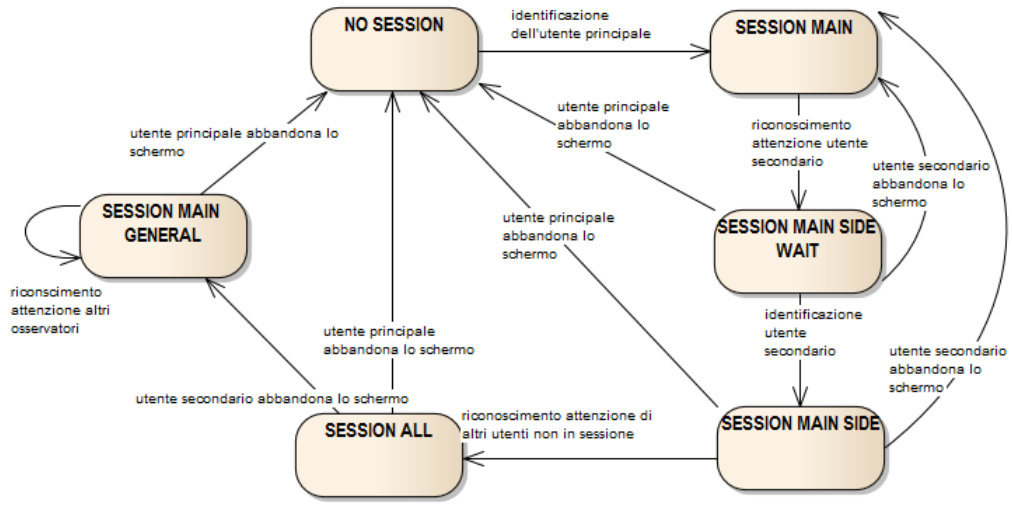

Figura 4.2: Tipi di sessione e possibili transizioni per lo schermo

- `SESSION MAIN SIDE` rappresenta il tipo di sessione che si ha quando anche l'utente secondario si è identificato e quindi lo schermo si divide mostrando ai due utenti i loro contenuti personalizzati;
- `SESSION MAIN GENERAL` si ha quando, oltre all'utente principale, altre persone che non siano l'utente secondario concentrano la loro attenzione sullo schermo, e a loro viene offerto un contenuto di interesse generale;
- `SESSION ALL` rappresenta una sessione completa, in cui oltre ai due utenti in sessione, anche altre persone hanno mostrato interesse verso lo schermo, che perciò mostra anche contenuti di interesse generale.

L'identificazione dell'utente gioca perciò un ruolo fondamentale in questo sistema, che però potrebbe essere facilmente esteso ad uno che mostri dei contenuti di interesse generale nel caso in cui l'osservatore non si identifichi, ed altri personalizzati in caso di autenticazione.

Risulta inoltre piuttosto chiaro già a livello di progetto che l'interazione tra utente principale e schermo può essere gestita con delle gesture di tipo Swipe, con le quali egli può cambiare i contenuti che lo schermo offre per lui a suo piacimento. Anche i contenuti per l'utente secondario sono dinamici, ma egli non ne ha il controllo, così come accade per quelli di carattere generale. Dal punto di vista estetico, si è pensato di mettere le informazioni di carattere generale nella parte alta, poiché più visibile anche da lontano, mentre i contenuti personalizzati (quali ad esempio i codici QR, destinati ad utenti vicini allo schermo) vengono posti nella parte bassa.

4.3 Elementi principali di implementazione

Per la realizzazione del prototipo sono stati molto utili i diagrammi degli stati mostrati in precedenza, poiché ogni stato viene mappato in un metodo da richiamare ogni volta che si vuole transitare proprio in quel particolare stato.

Le classi principali per questo prototipo sono rappresentate da

- `RecognizerScreen`, che implementa la macchina a stati dello schermo e inizializza anche la parte grafica: crea i pannelli delle schermate e li inizializza, utilizzando un `CardLayout` per mostrare ogni volta solo il pannello relativo allo stato in cui ci si trova. É proprio questa classe che infatti tiene traccia dello stato e del tipo di sessione (implementati entrambi come Enumeration Java) dello schermo, offrendo dei metodi per cambiarli. Per identificare la transizione che si sta compiendo è però necessario sapere anche lo stato in cui si era precedentemente, dopodiché si può passare allo svolgimento delle operazioni indicate nel particolare passaggio da uno stato all'altro;

- `KinectAttentionTracker` implementa il flusso di controllo atto a ricevere i dati dal sensore per poi utilizzarli soprattutto per riconoscere l'attenzione degli utenti presenti nella scena, in base al cui numero bisogna cambiare lo stato e, se necessario, il tipo di sessione dello schermo.

- `MainUserRecognizer` e `SideUserRecognizer`, che implementano i thread per i server di richieste di autenticazione rispettivamente per l'utente principale e poi quello secondario. É importante dire che sono questi thread che, una volta completata l'identificazione, notificano allo schermo di entrare in un nuovo stato: essi rivestono perciò importanza soprattutto nelle transizioni dallo stato `REC WAIT` allo stato `SESSION` e da `SESSION MAIN SIDE WAIT` a `SESSION MAIN SIDE`.

- nel package `gui` sono inoltre presenti le implementazioni di 3 pannelli che rappresentano ognuno una schermata relativa ad uno stato: il più complesso risulta sicuramente `SessionPanel`, poiché deve permettere di aggiungere contenuti anche dopo essere stato creato in quanto il tipo di sessione evolve con l'evoluzione dell'ambiente intorno allo schermo.

L'interazione naturale viene gestita dalla classe `KinectAttentionTracker` che, nel caso lo schermo sia in stato `SESSION`, provvede a rilevare le gestures Swipe configurando gli handler secondo i metodi visti in 3.3.2. Inoltre essa gestisce le transizioni di stato dovute al riconoscimento dell'attenzione di un nuovo utente oppure alla perdita dalla scena di un utente prima considerato

attento. In particolare le transizioni più significative gestite in questo modo sono quelle in cui si perde l'utente principale in sessione, che hanno come effetto quello di far tornare lo schermo nello stato `IDLE` in modo tale da dare la possibilità agli altri utenti di accedere ai contenuti personalizzati principali; e quelle in cui si perde l'utente secondario, che causa la scomparsa delle informazioni a lui dedicate, sia nel caso che si fosse in `SESSION MAIN SIDE` che in `SESSION ALL`.

4.4 Funzionamento

Viene ora proposta la descrizione del funzionamento dello schermo attraverso degli screenshot dei contenuti mostrati dallo schermo in uno dei possibili scenari a cui deve far fronte.

Inizialmente in stato di `IDLE` (figura 4.3), lo schermo attende di riconoscere l'attenzione di almeno un utente.

Figura 4.3: Schermata di idle

Una volta riconosciuto che un osservatore è interessato ai suoi contenuti, lo schermo passa nello stato `REC WAIT` (figura 4.4), in cui attende che l'utente si identifichi.

Una volta completata l'autenticazione dell'utente, lo schermo entra in stato `SESSION` con una sessione di tipo `SESSION MAIN`, in cui l'utente può decidere quali contenuti a lui destinati visionare grazie all'utilizzo delle gestures (figure 4.5 e 4.6).

Nel caso in cui lo schermo rilevi la presenza di un altro utente interessato, passa alla sessione `SESSION MAIN SIDE WAIT` che gli offre la possibilità di

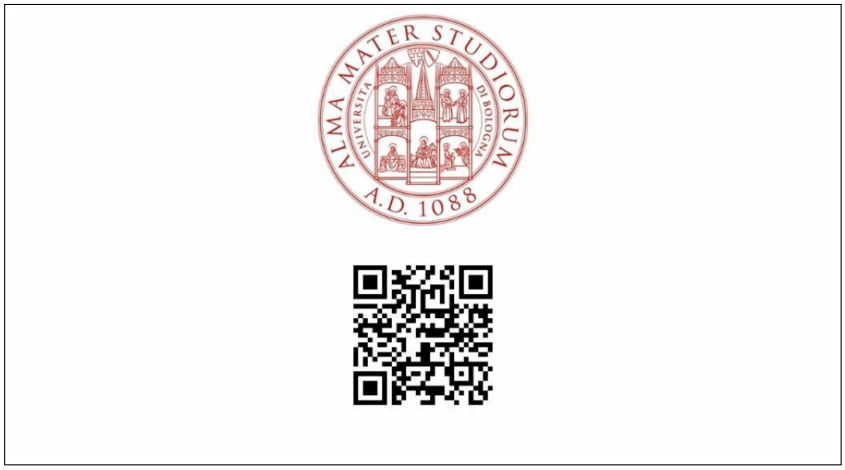

Figura 4.4: Schermata di wait

Figura 4.5: Schermata per SESSION MAIN: informazioni dell'utente

identificarsi mostrando un codice QR nella stessa schermata in cui l'utente principale sta interagendo(figura 4.7), separando le due aree con un linea di divisione: è importante ricordare che, grazie alle informazioni ricavate dal sensore, il codice viene mostrato sul lato destro o sinistro dello schermo a seconda della posizione dell'utente secondario in relazione a quello dell'utente principale.

Una volta identificato anche l'utente secondario, si passa ad una sessione `SESSION MAIN SIDE` e si mostrano i contenuti personalizzati per quest'ultimo al posto del codice QR utilizzato per l'identificazione: in questo caso viene

Figura 4.6: Schermata per SESSION MAIN: lista dei prossimi appelli d'esame

Figura 4.7: Schermata per SESSION MAIN SIDE WAIT

mostrato l'orario delle lezioni riguardante l'anno di corso dello studente in modo che il giorno visualizzato cambi automaticamente ogni 4 secondi (figura 4.8).

Se giungono altre persone interessate ai contenuti dello schermo, esso passa ad una sessione `SESSION ALL` mostrando un banner scorrevole in alto contenente delle informazioni di carattere generale (figure 4.9 e 4.10), come una variazione provvisoria all'orario delle lezioni.

Figura 4.8: Schermata per SESSION MAIN SIDE

Figura 4.9: Schermata per SESSION ALL

4.5 Deployment

Il progetto è stato realizzato su una macchina del tutto simile a quella descritta in 3.5: il codice sorgente è reperibile dal repository online all'indirizzo `https://bitbucket.org/smgrotti/recognizerscreen`. Questo sistema necessita delle librerie offerte dal framework realizzato, che devono essere così importate: per ulteriori riferimenti, si rimanda a 3.5.1.

Figura 4.10: Schermata per SESSION ALL

Capitolo 5

Conclusioni e sviluppi futuri

In questo capitolo vengono presentate le conclusioni, tirando le fila del lavoro svolto e ponendo in evidenza alcuni possibili sviluppi che esso suggerisce.

5.1 Conclusioni

Obiettivo di questa tesi era l'ingegnerizzazione di sistemi software per schermi pervasivi, raggiunto tramite lo sviluppo del framework descritto nei precedenti capitoli: esso ha messo in luce non solo i requisiti e le caratteristiche fondamentali di questo tipo di sistemi, ma anche le problematiche nel loro soddisfacimento, indicando delle tecniche e dei pattern utili per la loro risoluzione.

Infatti il framework presentato mostra, tramite la sua struttura, un metodo generale di organizzazione del software per questi sistemi: i componenti principali sono stati divisi e resi, per quanto possibile, indipendenti gli uni dagli altri in modo da permettere una maggior flessibilità e riusabilità nel loro utilizzo.

Inoltre risultano evidenti al termine di questo lavoro sia le potenzialità che anche i limiti delle librerie e degli strumenti utilizzati: data l'attenzione che questi temi riscuotono attualmente è praticamente certo che le mancanze riscontrate sia dal punto di vista hardware (precisione del sensore) che software (driver e middleware) siano colmate in breve tempo, donando agli sviluppatori sempre più funzionalità e maggior precisione ed affidabilità, spronando la loro creatività e vena realizzativa.

La realizzazione del prototipo è stata funzionale sia per la dimostrazione dei vantaggi che l'utilizzo dei componenti software prodotti porta durante lo sviluppo delle applicazioni e la loro implementazione, quale ad esempio

la netta diminuzione della quantità di codice da scrivere, sia per mostrare quanto vasti siano i campi di utilizzo di questi sistemi: applicazioni basate su schermi pervasivi potrebbero, in un futuro anche molto prossimo, interessare non solo in ambiente accademico ma anche in contesti aziendali, per scopi pubblicitari o di intrattenimento, oppure anche in strutture pubbliche.

Il sistema realizzato infatti consente di adattare i contenuti dello schermo in base al numero di persone e alle informazioni derivanti da una loro autenticazione: calando queste caratteristiche in un contesto pubblicitario, è facile immaginare schermi pervasivi in un centro commerciale che, a seconda della profilazione del cliente con cui stanno interagendo, mostrano le offerte per lui più interessanti e vantaggiose in modo da dargli maggiore visibilità e, in questo modo, aumentare le probabilità che quel cliente compri il determinato prodotto.

Infine, è bene ricordare il forte carattere progettuale e sperimentale di questo lavoro, che va interpretato come un punto di partenza, un insieme di funzionalità base da estendere e rendere più potenti ed affidabili per costruire sistemi software sempre più avanzati ed espressivi.

5.2 Sviluppi futuri

I possibili sviluppi suggeriti da questa tesi sono molteplici e qui di seguito si cerca di darne una sintetica ma significativa descrizione per ognuno di quelli analizzati.

5.2.1 Estensione del framework

Il framework realizzato rappresenta di sicuro un buona base, ma le funzionalità che offre possono essere potenziate ed estese in più di un verso:

- anziché riconoscere la sola attenzione dell'utente, si potrebbe realizzare un manager dello stato di un utente, di cui il suo livello di attenzione è solo un componente. Ad esempio, nelle informazioni riguardanti lo stato dell'osservatore potrebbero rientrare anche la sua distanza, la sua postura, eventuali caratteristiche biometriche deducibili tramite il sensore, tempo di permanenza davanti allo schermo: si potrebbe perciò progettare un manager per ognuna di queste feature, per poi aggregare le informazioni raccolte in una nozione di stato che, potendo variare a seconda delle esigenze applicative, deve essere configurabile. Un'estensione di questo tipo aumenterebbe di molto l'espressività offerta dal framework, aprendo nuove possibilità a svariati utilizzi.

- la parte relativa all'interfaccia naturale tra utente e schermo potrebbe essere migliorata rendendo più configurabile da parte del programmatore il riconoscimento delle gestures. Più nello specifico, si potrebbe consentire di specificare dei parametri aggiuntivi sul gesto che l'utente deve eseguire affinché venga riconosciuto come tale dal sistema: portando l'esempio di uno Swipe, potrebbe risultare molto utile impostare la velocità minima del movimento da rilevare e la sua ampiezza; similmente si potrebbe poi fare per tutte le altre gestures offerte da OpenNI. La difficoltà principale nella realizzazione di questa estensione sta nella documentazione per le librerie OpenNI Java: essa è scarsa e spesso assente, per cui non è ben chiaro il significato dei parametri con cui configurare i detector delle gestures, e per questo motivo una soluzione potrebbe essere quella di fare degli esperimenti provando ad eseguire i gesti con varie configurazioni di valori, per poi tenere traccia dei risultati ottenuti nella documentazione.

- riguardo all'identificazione, potrebbe essere realizzato un componente software che riconosce quando l'utente avvicina lo smartphone allo schermo per l'identificazione, in modo da avere la certezza che è proprio quello l'utente che si sta identificando. Questa informazione è altrimenti impossibile da desumere con certezza: l'approccio utilizzato attualmente infatti non permette di sapere quale, tra due utenti che sono contemporaneamente davanti allo schermo durante la fase di autenticazione, sia quello che veramente ha compiuto la sua identificazione; ad esempio, nel prototipo realizzato si assume che questo utente sia il primo arrivato. Questa estensione potrebbe essere realizzata prendendo spunto dall'architettura ad eventi già presente nelle librerie OpenNI per il riconoscimento delle gestures: realizzare un componente di questo tipo, che riconosca il movimento di avvicinamento del braccio dell'utente, porterebbe quindi al framework benefici in termini di precisione ed affidabilità.

5.2.2 Porting

Il sistema potrebbe essere migliorato permettendone il funzionamento in ambienti di lavoro diversi da quello (Linux Ubuntu 12.10) utilizzato per questo lavoro: in questo caso, visto che il software è stato scritto in Java (il cui motto è *write once, run everywhere*), si tratta di rendersi funzionanti ed operativi con i driver OpenNI per il sensore negli altri sistemi operativi. Questa operazione non è da sottovalutare, poiché sono molte le guide reperi-

bili in rete riguardo l'installazione di questi drivers, ma davvero poche quelle significative ed esaurienti.

Inoltre bisogna ricordare che il presente lavoro è stato realizzato utilizzando la libreria OpenNI 1.5, ma attualmente la versione più recente di questa libreria è la 2.2: potrebbe quindi risultare utile effettuare il porting delle librerie scritte alla versione più recente di OpenNI. Questa operazione, se da un lato può sicuramente portare dei miglioramenti in quanto la nuova versione risulterà di sicuro migliorata non solo qualitativamente ma anche dal punto di vista della quantità di funzionalità offerte, dall'altro può anche essere rischiosa, nel senso che potrebbe essere molto dispendiosa in quanto i concetti tra le due versioni della libreria potrebbero essere variati sensibilmente, e a quel punto effettuare il porting vorrebbe dire reimplementare, sfruttando i concetti e i pattern estratti dal lavoro svolto con la precedente versione, le librerie per il framework.

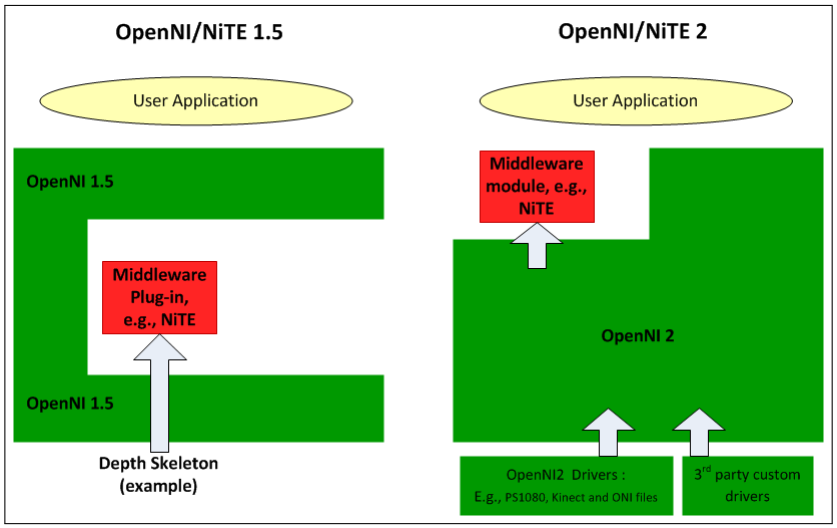

Figura 5.1: Confronto tra le le versioni 1.5 e 2.0 di OpenNI: dalla guida ufficiale OpenNI

5.2.3 Identificazione dell'utente

L'approccio utilizzato per l'identificazione dell'utente potrebbe essere migliorato per rendere il sistema ancora più trasparente rispetto al suo utilizzo da parte delle persone: ad esempio, visto che gli ultimi smartphone ne sono provvisti, si potrebbe pensare ad un'interazione basata su NFC, e quindi sul

concetto di localizzazione, per capire l'identità dell'utente che sta prestando attenzione allo schermo ed adattarne quindi i contenuti.

Oppure si potrebbe pensare all'utilizzo di altre tecnologie, quali il segnale bluetooth o le etichette RFID, sempre al fine di capire con maggiore precisione e trasparenza quale sia l'utente veramente interessato allo schermo e trarne quindi le informazioni personali.

Supponendo che l'utente abbia il sensore bluetooth acceso nel suo smartphone e sia davanti allo schermo, è facile che lo schermo, dotato anch'esso di tale connettività, recepisca la presenza dell'utente tramite la presenza del suo device: in questo caso la difficoltà maggiore è quella che riguarda la portata del segnale. Bluetooth infatti ha un range che si aggira, per i sensori più comuni (di classe 2), attorno ai 10 m[1]: il concetto di località offerto ha quindi questa precisione, che può rivelarsi insufficiente per sistemi che devono pilotare uno schermo, poiché le persone interessate sono molto vicine ad essa.

Inoltre bisogna ricordare che non sono state trattate, nella realizzazione, le problematiche di privacy e di sicurezza: per garantire un corretto utilizzo delle informazioni dell'utente bisogna sicuramente aggiungere queste funzionalità al framework.

5.2.4 Interfaccia naturale

Per quanto riguarda l'interazione con lo schermo, si potrebbe prevedere una nuova funzionalità basata sul riconoscimento vocale, che permetterebbe di rendere ancora più intuitivo il suo utilizzo e aprirebbe ancora più possibilità di utilizzo per questi sistemi.

Inoltre potrebbero essere pensati dei riconoscitori per movimenti non solo per le mani, ma anche per altre parti del corpo quali ad esempio la testa: si pensi a quanto sarebbe intuitivo rispondere, ad esempio, ad una interazione con lo schermo tramite un cenno della testa, sia positivo che negativo.

Infine, si potrebbe pensare all'utilizzo di sensori più avanzati e precisi per diminuire le assunzioni che caratterizzano il sistema: infatti, si potrebbe pensare ad una maggior robustezza agli errori nei casi in cui, per esempio, due utenti si incrociano davanti al sensore o sono troppo vicini.

[1]Informazione presa dal sito ufficiale di Bluetooth: `http://www.bluetooth.com/Pages/Basics.aspx`

Bibliografia

[1] Documentazione NITE. `http://www.primesense.com/solutions/nite-middleware/`.

[2] Documentazione OpenNI. `http://www.openni.org/reference-guide/?t=index.html`.

[3] Wikipedia. `http://www.wikipedia.org/`.

[4] S. Costanzi. Realizzazione di un prototipo di schermo pervasivo adattativo. Master's thesis, Università di Bologna, 2013.

[5] S. Franci. Relazione progetto lmc, 2013.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[7] B. Gollan, B. Wally, and A. Ferscha. Automatic human attention estimation in an interactive system based on behavior analysis. *Proceedings of the 15th Portuguese Conference on Artificial Intelligence(EPIA2011)*, 2011.

[8] S. Kean, J. Hall, and P. Perry. *Meet the Kinect: An Introduction to Programming Natural User Interfaces*. Apress, 2011.

[9] J. Kramer, N. Burrus, F. Etcher, D. Herrera, and M. Parker. *Hacking the Kinect*. Apress, 2012.

[10] A. Natali and A. Molesini. *Costruire sistemi software: dai modelli al codice*. Esculapio, 2009.

[11] D. Saha and A. Mukherjee. Pervasive computing: A paradigm for the 21st century. *IEEE Computer Society*, 2003.

[12] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 2001.

[13] M. Weiser. The computer for the 21st century. *Scientific American Special Issue on Communications, Computers, and Networks*, September 1991.

[14] C. D. Wickens and J. S. McCarley. *Applied Attention Theory.* CRC Press, 2007.

Ringraziamenti

Desidero ringraziare tutti coloro che mi hanno in qualche modo aiutato e supportato nel raggiungimento di questo importante traguardo che ha richiesto impegno e sacrificio, ma che mi ha anche portato delle belle soddisfazioni.

Vorrei ringraziare per primo il prof. Mirko Viroli, per la disponibilità e la cortesia con cui mi ha aiutato e spronato nell'attività sperimentale che questa tesi richiedeva e nella redazione dell'elaborato.

Voglio inoltre ringraziare sinceramente tutti i miei familiari per avermi sostenuto nei momenti di difficoltà: in particolare Ombretta e Luciano, i miei genitori, senza la cui fiducia e il cui sostegno, non solo economico ma in primo luogo morale, non avrei potuto pensare di raggiungere questo obiettivo, e mio fratello Matteo.

Grazie inoltre a tutti i miei amici con cui ho condiviso il tempo fuori dalla facoltà per la loro fiducia e il loro supporto nei miei confronti: in particolare mi rivolgo a Filippo, Matthew, Andrea e Luca.

Infine, vorrei sinceramente ringraziare anche i tanti compagni di questi anni di studio, e in particolar modo Simone, con cui ho condiviso svariati progetti durante l'arduo percorso universitario.