ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

IL PROGETTO TRAUMATRACKER: STUDIO E SVILUPPO PROTOTIPALE DI UN SISTEMA WEARABLE HANDS-FREE IN AMBITO HEALTHCARE

Elaborato in
PROGRAMMAZIONE DI SISTEMI EMBEDDED

Relatore
Prof. ALESSANDRO RICCI

Presentata da
MATTEO GABELLINI

Co-relatori
Ing. ANGELO CROATTI
Prof. CATIA PRANDI

Seconda Sessione di Laurea
Anno Accademico 2015 – 2016

PAROLE CHIAVE

wearable computing

eye-wear computing

hands-free

speech recognition

healthcare

smartglass

alla mia famiglia

Indice

Introduzione

In questa tesi si andrà a illustrare il progetto *TraumaTracker*, nato dalla collaborazione tra Università di Bologna e l'ospedale M. Bufalini di Cesena con l'obiettivo di creare un nuovo sistema informatico, che funga da supporto alle attività dei medici del *trauma center*. In tali attività fino ad oggi non è stato possibile introdurre dispositivi smart a causa di vari limiti tecnici. Ciò diventa oggi fattibile grazie all'introduzione di wearable devices. Per tale motivo si inizia analizzando prima il concetto wearable computing, come parte di un insieme di sistemi embedded/distribuiti basati su: agenti, intelligenza artificiale e realtà aumentata, i quali aprono la strada a modi completamente nuovi di percepire se stessi e il mondo che ci circonda. Ciò rende concreto il concetto di ubiquitous computing, cioè l'idea che la computazione diventi pervasiva e costante, in ogni nostra attività quotidiana. Si passerà poi ad analizzare una tra le più importanti applicazioni del wearable computing, cioè l'uso in ambito healthcare. In tale scenario risulta possibile identificare diversi tipi di device, tra i quali quelli a supporto del medico. Tali dispositivi possono essere visti come una vera e propria estensione di una parte del corpo dell'utilizzatore. Per raggiungere questo obiettivo però, è richiesto agli sviluppatori di rivedere alcuni approcci e metodi utilizzati fino ad oggi per lo sviluppo dei sistemi informatici, soprattutto l'ideazione e la progettazione di nuove tecniche di interazione tra uomo e macchina, che siano orientate ad un approccio hands-free. L'unione di questi sistemi wearable, a supporto del medico, e i sistemi informatici già presenti all'interno degli ospedali, permettono di creare una nuova visione di *ospedale aumentato 4.0*. All'interno di questa nuova visione è possibile collocare il progetto *TraumaTracker*, il quale costituisce un esempio concreto di quanto detto sopra. Si proseguirà quindi, analizzando il caso di studio del sistema che dovrà essere sviluppato, focalizzandone i requisiti e presentando poi le varie scelte progettuali/implementative. Si cercherà anche di far cogliere l'importanza di tali sistemi e l'utilità che possono costituire concretamente per i medici. Nonché le nuove opportunità di gestione del flusso delle informazioni relative ad un intervento, attraverso l'uso di sistemi che permettano di automatizzare i diversi processi eseguiti finora da parte dell'utente. Tra gli obiettivi principali che si pone il progetto, vi è quello di offrire un approccio di

interazione che permetta all'utente di avere le mani libere per poter eseguire altre attività, mentre si interfaccia con il sistema. Tra gli approcci possibili all'interazione hands-free, una di quelle che si sta diffondendo sempre di più nell'ultimo periodo, è l'utilizzo della voce. A tal proposito sarà introdotto come è stato integrato, all'interno del sistema, un servizio di speech recognition, cercando di capire come sia possibile adattare uno speech engine ad un caso d'uso specifico, a fronte delle limitazioni che esso presenta. Per fare in modo che un wearable device sia veramente un'estensione dell'utilizzatore, occorre che il suo uso comporti pochissimo sforzo e garantisca un'interazione veloce, quasi immediata, in modo tale che l'interfacciamento con il sistema sia una cosa pressoché naturale. Generalmente i sistemi wearable vengono utilizzati mentre un utente sta svolgendo anche altre attività. Vi è quindi la necessità, soprattutto nel caso di smartglass, di progettare opportunamente un'interfaccia grafica che garantisca quanto detto sopra e non arrechi disturbo all'utente mentre sta svolgendo altre attività. Ciò è più che mai valido all'interno dello scenario di utilizzo di *TraumaTracker*. Verrà quindi illustrata la progettazione di un'interfaccia che cerchi di rispettare quanto detto. Considerando che il sistema *TraumaTracker* è tuttora in fase di sviluppo, successivamente all'introduzione di esso, sarà poi presentato un piccolo test, che permetta di capire se le scelte prese fino a questo momento sono corrette e possano già soddisfare alcuni requisiti.

Capitolo 1

Wearable Computing in ambito Healthcare

In questo capitolo verrà introdotto il concetto di **wearable computing**, analizzando il suo utilizzo in contesti *medico ospedalieri*, per capirne l'utilità e l'importanza. Verrà fatto anche un approfondimento sui possibili approcci e paradigmi da utilizzare per lo sviluppo di applicazioni per wearable devices.

1.1 Introduzione al Wearable Computing

Il continuo sviluppo dell'informatica e dell'elettronica ha portato ad una sempre maggior miniaturizzazione dei dispositivi capaci di elaborare informazioni, rendendo ormai la computazione pervasiva in tutti i contesti della vita delle persone. Un esempio concreto di ciò può essere visto attraverso il **wearable computing**. Esprimere una definizione precisa di **wearable computing** potrebbe non essere semplice. Nel mondo della ricerca ne possiamo trovare diverse, le quali forniscono spiegazioni da punti di vista differenti. Si trovano definizioni legate ad una visione prettamente a livello hardware, attraverso le quali è possibile interpretare il weareble computing come l'insieme di tutti quei *sistemi embedded* indossabili, il cui utilizzo introduce nuovi concetti e paradigmi allo sviluppo di tali sistemi rispetto a classici *sistemi embedded* [2]. D'altro canto possiamo trovare definizioni dal punto di vista dell'utente, le quali vedono il wearable computing come un'estensione dell'utilizzatore. Ciò permette di arricchire in modo significativo, attraverso informazioni digitali, la percezione e l'interazione dell'utente con il mondo circostante. Allo stesso tempo introduce nuovi approcci all'analisi e allo sviluppo di nuovi stili di interfacce [4],[5],[8].

Letteralmente **wearable computing** significa computazione indossabile, e provando ad unire varie definizioni, potremmo dire che tale termine racchiude:

- tutto quell'insieme di oggetti che possono essere indossati, capaci di elaborare informazioni (**wearable devices**).
- e tutti quei concetti legati all'utilizzo di tali dispositivi e allo sviluppo di applicazioni per essi

1.1.1 Wearable Devices

I *wearable devices* possono essere sia indumenti (giacche, t-shirt, pantaloni, ecc...) che accessori (occhiali, orologi, anelli, ecc...), i quali, come i classici sistemi embedded, integrano microprocessori/microcontrollori ed attraverso opportuni sensori ed attuatori, sono in grado di interagire sia con il mondo fisico sia con l'utente che li indossa. Sempre con il termine wearable device possiamo anche identificare tutti quei dispositivi che possono essere montati su oggetti indossabili, i quali nascono senza un sistema embedded integrato (ad esempio: Vuzix M-100 è un dispositivo che può essere agganciato ad una propria montatura, rendendo così i propri occhiali dei wearable devices).

Figura 1.1: Thad Starner che indossa un paio di occhiali che integrano un display

Come afferma Thad Starner [4] (un pioniere del wearable computing), uno degli obbiettivi di tali dispositivi (e direttamente anche della wearable computing) è quello di fornire un assistente intelligente all'utente, il quale può permettere di arricchire la propria memoria, intelletto, creatività, comunicazione, abilità e sensi. Si potrebbe pensare che alcuni di questi obiettivi siano già stati raggiunti dagli smartphone, ma una differenza importante che distingue tali sistemi dai wearable devices è che quest'ultimi vengono progettati

appositamente per un uso continuativo da parte dell'utente durante il corso della giornata, il che porta inevitabilmente a rivedere molte tecniche e paradigmi utilizzati finora nello sviluppo di sistemi mobile nel caso si voglia adoperare approcci simili.

1.1.2 Tecniche di interazione e scenari di utilizzo

Fino a pochi anni fa, a livello consumer[1], era comune utilizzare i computer solo attraverso mouse, tastiera o touchscreen, quindi impegnando sempre le mani dell'utente. Ora il **wearable computing** apre la strada anche all'evoluzione di paradigmi di interazione *hands-free* o *hands-limited*, in cui all'utente non è più richiesto totalmente o in parte l'utilizzo delle mani per controllare il sistema. Nell'uso dei classici sistemi desktop e sistemi mobile, l'utente è anche costretto a concentrarsi sull'interazione o con il sistema o con l'ambiente che lo circonda, non potendo prestare attenzione ad altre attività nello stesso momento [9]. Generalmente attraverso il wearable computing, invece, l'utente può non distogliere totalmente l'attenzione dalla sua attività principale e allo stesso tempo interagire con il sistema [fig:1.2].

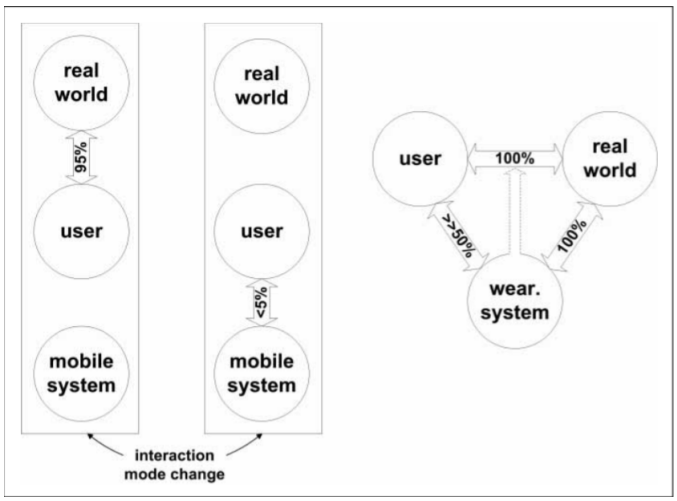


Figura 1.2: Differenza di interazione tra i comuni sistemi mobile e i wearable devices.

Questo automaticamente, permette di inserire nuovi dispositivi in tutti quei contesti che spaziano dal fitness alla moda o dall'intrattenimento al lavoro [7],

[1]Nonostante nell'ambito della ricerca, lo studio dei *wearable devices* sia già presente da diversi decenni, l'uso di tali sistemi a livello consumer sta facendo il suo ingresso solo negli ultimi anni

in cui l'utente ha generalmente le mani impegnate o che comunque non può usarle per interagire direttamente con i dispositivi. Si provi a pensare, ad esempio, ad un tecnico, che si trova a dover riparare un macchinario. Fino ad oggi egli era costretto o ad operare sul sistema oppure a consultare un manuale, non potendo fare le due cose contemporaneamente. Oggi invece, grazie all'uso di wearable devices, come gli occhiali dotati di display (detti anche smartglasses), un tecnico può operare direttamente su un macchinario e allo stesso tempo, consultare il manuale attraverso il display integrato sugli occhiali. Addirittura, se il wearable device possiede le caratteristiche adatte può, utilizzando opportune applicazioni di realtà aumentata, vedere passo a passo le manovre che deve compiere, attraverso delle animazioni olografiche [fig:1.3].

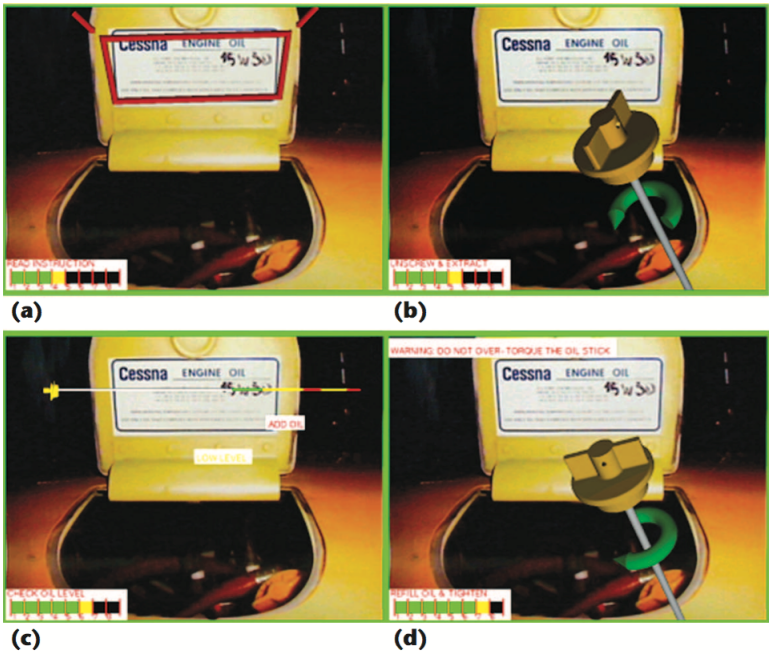


Figura 1.3: Esempio di realtà aumentata a supporto dei tecnici per attività di manutenzione. Le varie operazioni che l'operatore deve eseguire sono rappresentate da ologrammi [6].

Un'altra caratteristica molto importante è che i wearable devices possono essere dotati di diversi sensori, i quali oltre a poter essere usati per comandare il sistema, permettono di monitorare determinati parametri relativi all'utente, oppure all'ambiente in cui egli si trova.

1.2 Utilizzo in ambito HealthCare

Considerando le caratteristiche descritte in precedenza. Si può facilmente intuire che uno degli scenari di maggior utilità per il wearable computing è indubbiamente l'ambito medico ospedaliero. In tale scenario si possono trovare diversi tipi di sistemi wearable che, a seconda del loro scopo, potrebbero essere divisi in due macro famiglie.

- dispositivi per il monitoring dei pazienti
- dispositivi a supporto dei medici

1.2.1 Wearable devices per il monitoring del paziente

Grazie al fatto di venir progettati per essere indossati quasi costantemente e alla possibilità di sfruttare diversi sensori, i wearable devices permettono di superare molti limiti della medicina ambulatoriale, incentivando nuove tecniche di monitoraggio a lungo termine e relativi trattamenti real-time [10]. Sotto il punto di vista del mero monitoring, non è difficile già trovare sul mercato diversi dispositivi indossabili in grado di misurare: ECG, pressione sanguigna, ecc... [9]. Tutti questi sistemi permettono di ottenere dati continuativi relativi a lunghi periodi (giorni, settimane, ecc...), di conseguenza offrono anche la possibilità di ottenere informazioni relative a stati fisici differenti del paziente monitorato (stato a riposo, sotto sforzo ecc..). Sfruttando poi wearable devices dotati di connettività, si potrebbero implementare nuove tecniche di telemedicina dove, per esempio, il dispositivo automaticamente invii i dati raccolti al medico oppure, in caso di emergenza, chiami automaticamente i soccorsi [11]. Un'implementazioni di tali sistemi la possiamo vedere con HealthWear [10], un sistema composto da [fig:1.4]:

- una maglia dotata di sensori, che permettono la misurazione di: ECG, HR, SpO2, temperatura, respirazione e attività motoria dell'utente,
- una unità portatile che permette di inviare i dati tramite GPRS ad un sito centrale per processare i dati.

Figura 1.4: HealthWear: unità portatile(dispositivo di sinistra), maglia con sensori (dispositivo a destra)

Si pensi poi a quei casi in cui l'utente stia seguendo una terapia, nella quale è prevista l'assunzione periodica di farmaci, il dispositivo potrebbe notificargli automaticamente dei promemoria [9]. Con i wearable devices più avanzati, sfruttando la possibilità di programmarli e la potenza computazionale di cui sono dotati, risulta interessante, a fronte dei dati rilevati, pensare di implementare opportuni comportamenti, che permettano di effettuare veri e propri trattamenti in real-time direttamente attraverso il dispositivo. Per esempio i malati di diabete, grazie ad opportuni wearable devices, potrebbero costantemente tenere sotto controllo il livello di glucosio nel sangue e, in relazione ad esso, il sistema potrebbe in automatico somministrare la giusta quantità di insulina [11]. Infine, sfruttando i wearable devices, è possibile creare anche sistemi che permettano di limitare o superare le difficoltà derivanti da disabilità [11].

1.2.2 Wearable devices a supporto del lavoro del Medico

Come detto in precedenza, i wearable devices risultano particolarmente adatti in tutti quei contesti in cui l'utente ha le mani impegnate oppure quelli in cui l'interazione con un sistema computazionale non richiede di distogliere l'attenzione dell'utente dal lavoro che sta svolgendo. Uno di questi contesti è indubbiamente il lavoro del medico. Sfruttando nuovi sistemi che includono i wearable devices è possibile pensare a una nuova figura di dottore, un vero e proprio *Medico aumentato*. Ad esempio mentre un medico sta visitando una

persona, potrebbe tornargli molto utile ottenere in automatico tutte le informazioni relative a esami sostenuti dal paziente, questo potrebbe essere fatto attraverso degli smartglasses. Un chirurgo che opera, per vedere i parametri vitali del paziente, è costretto a spostare lo sguardo verso il display del macchinario che li misura. Sfruttando ancora gli smartglasses, si potrebbe pensare di mostrare tali parametri direttamente attraverso gli occhiali [7]. Anche in questo caso, utilizzando le risorse computazionali dei wearable devices, è possibile implementare sistemi reattivi e proattivi, che oltre a fornire informazioni al medico lo supportino anche nelle decisioni da prendere. Un esempio concreto è l'applicazione proposta da C. Baber [2][3], il quale presenta un sistema rivolto ai paramedici dell'ambulanza [figue:1.5]. Il paramedico alla fine di un intervento è tenuto a compilare un report contenente:

- i parametri vitali del paziente misurati durante l'intervento
- tutte le attività svolte in relazione ai parametri vitali

il progetto proposto da C.Baber costituisce un sistema automatico per la compilazione di tale report.

Figura 1.5: Primo prototipo del progetto relativo all'applicazione di C.Barber [2].

Il sistema funziona nel modo seguente: l'utente attraverso dei comandi vocali può selezionare tra i parametri vitali quale vuole inserire. Successivamente, in base al valore dei parametri inseriti, il sistema automaticamente presenta quali manovre possono essere effettuate e quali farmaci possono essere somministrati. L'utente, una volta presa una decisione, registra quale azione esegue, sempre attraverso un comando vocale.

1.3 Approcci e paradigmi allo sviluppo di applicazioni per il wearable computing

In letteratura sono reperibili diverse analisi riguardo lo sviluppo di wearable devices, dove ognuna affronta nello specifico varie problematiche. In questo lavoro non si ha l'obbiettivo di riassumere tutte le problematiche e relative soluzioni, ma di illustrare le linee guida generali da tenere a mente sempre. Si può partire da una premessa iniziale, considerabile come necessaria per iniziare una buona progettazione. Come accennato in precedenza, le applicazioni per sistemi desktop o per sistemi mobile vengono sviluppate con l'assunzione che l'utente, durante l'uso del sistema, non svolga contemporaneamente altre attività. Per prima cosa occorre considerare il fatto che i wearable devices, sotto un certo punto di vista, potrebbero essere considerati come delle vere e proprie estensioni dell'utente [5], quindi, tra l'utente e l'ambiente, non si pongono solo come mediatori all'interazione, ma possono anche essere affiancati ponendosi come un vero e proprio supporto [fig:1.2]. Tali sistemi fungendo da supporto, permettono all'utente di interagire con il sistema e nello stesso istante svolgere anche altri lavori. Si può chiaramente dedurre che come requisito di base, il sistema non debba in nessun modo costituire un intralcio ad altre attività dell'utente. Detto ciò, occorre quindi in fase di progettazione, porre l'utente al centro e in funzione di esso sviluppare il sistema. Considerando anche che i wearable computer possono essere visti come dei veri e propri sistemi embedded [2], anche per essi in genere si tende a sviluppare applicazioni *specific purpose*. Da questo punto di vista C.Barber [2] consiglia di seguire un approccio Activity-based in fase di analisi e modellazione, cioè un approccio concentrato sullo scenario di utilizzo dell'utente. Tale approccio permette di concentrarsi sui requisiti posti dall'utente mettendo da parte, in una prima fase, problemi legati ad aspetti tecnici in modo da ottenere uno scenario di utilizzo chiaro e conciso.

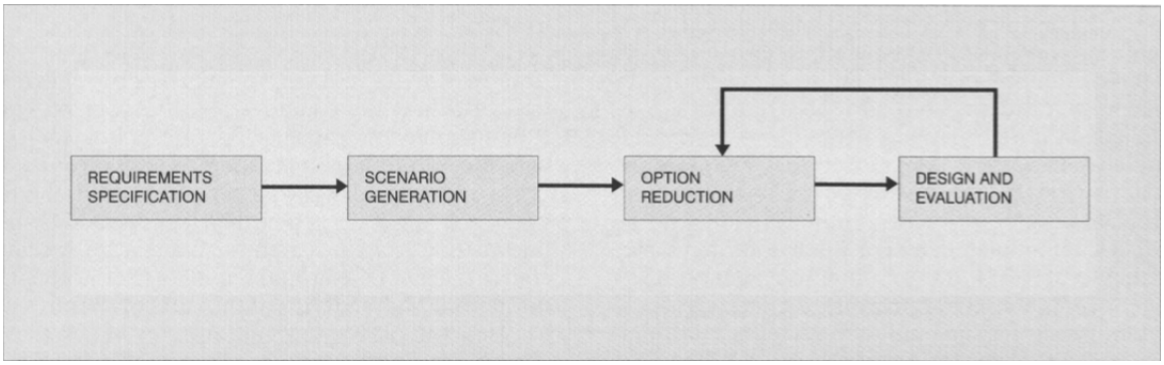


Figura 1.6: Schema Activity-Based design methodology, tratto da [2]

Activity-Based design methodology [2] si compone di 4 passi [schema fig:1.6]:

1. **Requisiti dell'utente**: al primo passo si parte focalizzandosi sui requisiti dell'utente. Esistono dei requisiti legati a fattori umani, che sono sempre fissi per tutti i wearable devices e sono:

   - le informazioni inviate dal sistema all'utente devono essere facilmente visibili e brevi,
   - l'utente non deve essere sovraccaricato di informazioni,
   - le informazioni devono essere facilmente interpretabili e memorizzabili dall'utente.

   I limiti elencati sopra, sono legati alla capacità dell'utente di memorizzare e alla capacità del sistema a trasmettere le informazioni (es: dimensione del display limita il numero di parole che possono essere scritte). Successivamente, abbiamo dei requisiti connessi alle capacità dell'utente nell'utilizzare sistemi informatici:

   - l'utente deve essere capace di interpretare le informazioni rilevanti velocemente,
   - l'attrezzatura deve supportare un'interazione continua con l'utente,
   - i Comandi impartiti dovranno essere facili da ricordare e i dialoghi facili da seguire.

   Infine, a seconda del contesto, se il sistema viene usato mentre l'utente svolge altre attività:

   - ci dovrà essere minima interferenza tra l'attività dell'utente e l'uso del computer,
   - ll sistema dovrà essere usato mentre si è in movimento,
   - il sistema deve poter essere usato quando entrambe le mani sono occupate.

   Altri requisiti specifici dell'applicazione, che deve essere sviluppata, sono legati allo scenario di utilizzo.

2. **Generazione dello Scenario**: in questa fase, insieme agli utenti finali viene definito uno scenario di utilizzo concreto, in modo da poter individuare i requisiti specifici per l'applicazione. Questa fase ci permette anche di associare i compiti che svolgerà il sistema con le altre attività che l'utente potrebbe effettuare nello stesso momento. Una volta definiti

i requisiti specifici, si chiede all'utente finale di ordinarli per importanza. Contemporaneamente alla definizione dei requisiti specifici, si stabiliscono possibili combinazioni di modalità di interazione tra sistema e utente (es: comandi impartiti a voce e risultato mostrato da un display oppure comandi impartiti a voce e risposta del sistema fornita sotto forma di audio).

3. **Riduzione delle opzioni**: in questa fase in funzione dei requisiti specifici definiti in precedenza, vengono analizzati i pro e i contro delle varie combinazioni stabilite, in modo da eliminare quelle che risultano già inadatte.

4. **Implementazione del prototipo e valutazione**: per le opzioni rimaste vengono implementati dei prototipi e dei criteri di valutazione, con i quali si eseguono dei test insieme agli utenti finali, in modo da capire quale opzione sia la migliore e quali modifiche devono essere apportate.

Per favorire una ancor maggior integrazione tra utente e wearable system, Thad Starner [4] elenca degli attributi chiave:

- Il sistema deve fornire accesso costante e persistente alle informazioni con il minimo sforzo da parte dell'utente

- Per fornire un supporto cognitivo il sistema dovrebbe essere in grado di osservare e creare un modello: del suo stesso stato, dell'ambiente in cui l'utente si trova e dello stato psicofisico di quest'ultimo. Dovrebbe allo stesso tempo mostrare tale modello all'utente, in modo da fornire la possibilità di correggerlo in caso di incomprensioni.

- Il sistema deve adattare le modalità di interazione con l'utente in base al contesto in cui si trova, cioè dovrebbe cambiare automaticamente modalità in funzione del comportamento sociale più educato (es:se messaggiando con un'altra persona attraverso degli smartglass, l'utente si mette alla guida, il sistema automaticamente cambia modalità e invece di mostrare i messaggi a video, li legge attraverso la modalità di *Text to Speech*).

- Aumentare e mediare l'interazione con l'ambiente circostante, fornendo informazioni filtrate in base al luogo fisico in cui si trova l'utente e ai suoi bisogni/gusti, attraverso un'interfaccia adatta alle preferenze e abilità dell'utente

Come avviene per i sistemi embedded, anche per i wearable devices occorre progettare opportunamente la parte hardware facendo attenzione a diversi

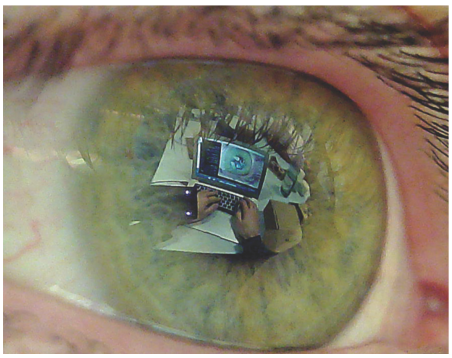
aspetti. I circuiti dei wearable devices devono essere alimentati, occorre quindi progettare opportunamente un circuito di alimentazione. In genere si utilizzano batterie, questo comporta di prestare particolare attenzione al consumo di energia, raggiungendo un trade-off tra tempo di utilizzo e durata delle batteria. Potrebbe tornare molto utile l'utilizzo di batterie ricaricabili, le quali potrebbero venir caricate direttamente da azioni compiute dall'utente (es: pressione di bottoni, giramento di ghiere, ecc...) o dai fenomeni fisici che rilevano (es: accelerazione gravitazionale) [4]. Essendo indossati è necessario prestare particolare attenzione anche alla dissipazione di calore per evitare disturbi nell'utente durante l'utilizzo, progettando sistemi di raffreddamento ad-hoc oppure sviluppando software che tenga in considerazione il problema (es: evitare di eseguire due operazioni contemporaneamente che surriscalderebbero troppo il sistema) [4]. Occorre progettare opportunamente anche la parte di comunicazione tra il wearable device e altri sistemi computazionali, tenendo a mente che non sempre la rete potrebbe essere disponibile [4]. I wearable devices potrebbero gestire dati sensibili(ad esempio dati sanitari rilevati [11]), ciò impone di tenere in considerazione anche aspetti di sicurezza dei dati e tutela della privacy, implementando le dovute protezioni sia a livello hardware che software.

1.4 Eyewear Computing & Sviluppo di interfacce per Smart Glass

Nell'ultimo anno, lo sviluppo sempre maggiore della ricerca sull'uso di occhiali smart in ambito di virtual reality ed augmented reality e dispositivi che potevano essere indossati in testa, ha portato all'introduzione di un nuovo termine, il quale ha permesso di denotare tutto il settore della wearable computing che utilizza, come mezzo di interazione principale, gli occhi. Tale settore prende il nome di Eyewear computing. Attraverso tali dispositivi è possibile testare nuovi metodi di apprendimento [7], nonché effettuare analisi su larga scala relativamente a comportamenti e sensazioni umane [12]. Su questi dispositivi si potrebbero trovare diversi sistemi di input [12]:

- Tracciamento oculare (eye-tracking): attraverso tale sistema è possibile tracciare il movimento degli occhi, ciò permette di analizzare attività e processi cognitivi dell'utente [fig:1.7a]. Sfruttando poi camere speciali, che permettano di effettuare corneal imaging, è possibile analizzare l'immagine riflessa dalla cornea per ottenere dati relativi a ciò che sta osservando l'utente [fig:1.7b].

(a) Prototipo in grado di effettuare eye-tracking



(b) Immagine riflessa nella cornea utilizzata per effettuare corneal imaging

Figura 1.7: Esempio di eye tracking e corneal imaging [12].

- Egocentric Camera: sono camere che permettono di ottenere una visione in prima persona dell'utente (similmente a quello che si ottiene con il corneal imaging), in modo tale da ottenere informazioni su ciò che circonda l'utente. Esiste però una differenza sostanziale tra la vista umana e quella ottenuta attraverso la camera. La camera acquisisce tutto il campo visivo, mentre le persone mettono a fuoco solo il punto che stanno guardando.

- Microfoni: usati per impartire comandi vocali al sistema

- Sensori inerziali: permettono di acquisire informazioni relative ai movimenti dell'utente. Usati, ad esempio, per rilevare espressioni facciali, in modo tale da interpretare gli stati emotivi dell'utente.

- Sensori per la rilevazione di attività cerebrali: usati per rilevare i processi cognitivi dell'utente.

- Touchpad e bottoni [7]

- Gesture recognition [4]: per il riconoscimento dei gesti dell'utente

Come detto in precedenza, principalmente come senso umano per l'interazione con il eyewear device, viene usata principalmente la vista, soprattutto per quanto riguarda l'output del dispositivo. Tali sistemi però essendo montati sulla testa, offrono la possibilità di utilizzare anche altri sensi tra cui: suono, olfatto e tatto, aprendo la strada a nuove applicazioni dove l'interazione tra uomo e macchina sfrutta nuovi approcci multi-modali.

Per quanto riguarda la progettazione di tali sistemi vengono usati diversi parametri [12][4][7]:

- eyebox
- distanza fra gli occhi
- campo visivo
- posizionamento e distanza delle immagini virtuali
- banda delle lunghezze d'onda
- qualità e risoluzione delle immagini
- ...e a livello consumer anche l'estetica

Relativamente alla progettazione delle interfacce grafiche per tali dispositivi vi è distinzione fra:

- Optical see-through: cioè dispositivi che permettono di vedere oltre al display
- Video see-through: cioè dispositivi che permettono di vedere oltre allo schermo, ma l'immagine mostrata è quella acquisita da una telecamera
- Display Opachi: che non permettono di guardare oltre al dispositivo, ma solo di visualizzare scritte e immagini, come un normale display

Ogni occhiale poi, potrebbe presentare ottiche per entrambi gli occhi o solo per uno. Per i vari dispositivi possono essere forniti direttamente dalla casa costruttrice delle linee guida specifiche per la progettazione delle interfacce [7]. In generale però, escludendo i visori per la realtà virtuale e riprendendo anche ciò che è stato detto in precedenza, potremmo dire che occorre costruire interfacce grafiche semplici, precise e concise. Evitando scritte troppo lunghe, parole complicate e fornendo all'utente solo informazioni rilevanti.

1.5 Usare la voce come input per il sistema

In contesti hands-free il primo sistema per controllare i dispositivi wearable è sicuramente l'uso della voce. Nell'ultimo periodo, si può notare che anche il mondo mobile è sempre più invaso da applicazioni che supportano l'uso di comandi vocali. Basti pensare a tutte le grandi piattaforme mobile: Android, iOS e Windows Phone, le quali possiedono un loro assistente vocale integrato al sistema operativo. I sistemi di riconoscimento vocale, grazie anche al

grande sviluppo che sta avendo l'intelligenza artificiale in questi anni, stanno migliorando a livelli esponenziali e sempre più aziende fornisco API per implementare applicazioni che utilizzano speech engine. Esiste però uno grossa differenza tra di essi. Alcuni speech engine per offrire la funzionalità di riconoscimento vocale si basano su sistemi online che utilizzano servizi in cloud. In questo caso il sistema per effettuare il riconoscimento si appoggia su datacenter dotati di potenti server. Questo comporta una maggior accuratezza, ma in caso di problemi con la rete potrebbero essere introdotte latenze e in assenza totale di rete il sistema non funzionerebbe [13]. Esistono poi speech engine offline che sfruttano vocabolari interni, i quali eliminano questo problema, ma comportano anche una minor accuratezza dei risultati e le parole che riescono a riconoscere è limitato a quelle presenti nel vocabolario. Molti per funzionare correttamente, necessitano anche di un periodo di training, in cui all'utente è richiesto di leggere testi prestabiliti, in modo tale che il sistema riesca ad adattarsi al modo di parlare dell'utilizzatore.

Purtroppo l'uso del riconoscimento vocale, può presentare alcuni problemi. In genere per acquisire il comando vocale, vengono usati dei classici microfoni, che in contesti rumorosi non permettono il corretto riconoscimento. Per risolvere il problema alcuni speech engine applicano già dei filtri al segnale acquisito dal microfono, ma in alcuni casi ciò potrebbe non bastare. Un'ulteriore alternativa, che riduce ulteriormente l'errore, è utilizzare un laringofono insieme ad un microfono, ma questo comporta apportare delle modifiche allo speech engine [13][14][15][16][17][18][19][20], che in alcuni casi non sono possibili.

Anche il formato dei comandi potrebbe influire sull'efficacia del riconoscimento, ad esempio Google, nelle sue linee guida, afferma che un comando per essere riconosciuto dal suo assistente vocale deve essere almeno di tre sillabe [7].

Capitolo 2

Il Progetto TraumaTracker

In questo capitolo si andrà ad introdurre il sistema **TraumaTracker**, analizzando prima il caso di studio per poi illustrarne l'architettura generale, il tutto integrato dalla descrizione delle motivazioni che hanno portato alle varie scelte progettuali.

Lo sviluppo del sistema è stato effettuato da un team composto da professori,dottorandi e studenti tra cui io. La fase di progettazione generale dell'intero sistema è stata effettuata da tutto il team, mentre per quanto riguarda la progettazione interna delle singole parti, ogni componente del gruppo ne ha presa in carico una o più. Successivamente saranno illustrate le scelte e decisioni prese a livello di team. Le parti prese in carico da me saranno invece illustrate nei capitoli successivi.

Si premette anche che il sistema è tutt'ora in fase di sviluppo, quindi di seguito saranno illustrati i processi e le attività, che hanno portato all'attuale prototipo. Alcune scelte poi sono state accompagnate dall'obiettivo di voler offrire al cliente, nel minor tempo possibile, prototipi incrementali, che permettessero di effettuare dei "test & feel", in modo da raffinare progressivamente requisiti e prestazioni del sistema, in funzione delle impressioni ed esigenze dell'utente. Si precisa che a prescindere dall'approccio utilizzato, tra gli obbiettivi principali del team di sviluppo è rimasto quello di costruire un prodotto che rispetti le buone regole dell'ingegneria.

2.1 Analisi

L'idea del sistema **TraumaTracker** nasce come risposta ad un'importante esigenza di medici e chirurghi del trauma center dell'Ospedale M.Bufalini di Cesena, i quali necessitano di un sistema informatico che li supporti durante il loro lavoro. L'intero sistema è sviluppato da un team composto da diverse persone, che fin dall'inizio, ha mantenuto uno stretto rapporto con gli utenti

finali(i medici). Si comincia definendo un caso di studio, su cui analizzare requisiti ed esigenze che il sistema finale dovrà soddisfare.

2.1.1 Caso di studio

Quando in caso di emergenza, un'ambulanza/eliambulanza porta un paziente al pronto soccorso, esso viene preso in carico da un team di medici ed infermieri, i quali effettuano le prime cure in modo da stabilizzare le sue condizioni per poi valutare a quale reparto dovrà essere assegnato. L'intero team, in gergo, viene chiamato *Trauma Team* e tra i medici che ne fanno parte, vi è uno che prende il nome di *Trauma Leader*, il quale oltre a intervenire personalmente sul paziente, coordina e dirige le attività del resto del gruppo. Durante un intervento il *Trauma Leader* può effettuare diverse operazioni che possono essere distinte in: misurazione dei parametri vitali, manovre rianimatorie e somministrazione di farmaci. Per quanto riguarda i parametri vitali, alcuni di essi vengono misurati da appositi macchinari dotati di display, i quali permettono di visualizzare il valore relativo al parametro misurato. Ciò però comporta il fatto che il *Trauma Leader*, per vedere il valore associato al parametro vitale di interesse, mentre è impegnato ad eseguire qualche manovra, è costretto a spostare lo sguardo dal paziente al display. In casi di massima gravità, ciò costituisce una perdita di tempo prezioso, soprattutto se il display si trova alle spalle del *Trauma Leader*. Una volta terminato l'intervento, il *Trauma Leader* è tenuto a compilare un report contenente: i parametri vitali misurati al paziente con i rispettivi valori, le manovre effettuate e i farmaci somministrati con le rispettive quantità. Per ogni attività deve indicare in che momento l'ha eseguita e in quale stanza del pronto soccorso. Tale report, ancora tutt'oggi, viene compilato dal *Trauma Leader* in base a quello che si ricorda dell'intervento e può capitare che qualche particolare possa sfuggire, considerando anche che spesso durante il loro lavoro, i componenti del *Trauma Team* si trovano ad affrontare situazioni altamente concitate. Il report in alcuni casi potrebbe avere valenza legale, quindi necessita di essere compilato nel miglior modo possibile.

Il sistema da sviluppare, si pone quindi l'obiettivo generale di rendere automatici, ove possibile, tutti quei processi di un intervento,in primis la memorizzazione dei vari eventi, i quali potrebbero causare distrazioni inutili al **Trauma Leader**, compromettendo l'efficienza dell'intero team. Oltre all'utilità in fase operativa, il sistema permetterà di acquisire informazioni utili anche per elaborare analisi statistiche sui vari interventi, in modo da individuare problemi nelle linee guida e procedure seguite dal *Trauma team* e offrire la possibilità di correggerli.

2.1.2 Requisiti

Analizzando il caso di studio sopracitato è possibile individuare diversi requisiti che il sistema dovrà possedere:

- Memorizzare in qualche modo le manovre e le somministrazioni impartite/effettuate dal trauma leader
- Per ogni manovra riuscire ad associare l'orario e il luogo in cui vengono effettuate
- Per quei parametri vitali in cui vi è un macchinario dotato di display per la misurazione, mostrare al trauma leader il valore senza che egli debba guardare il macchinario che li misura
- A fine intervento a seguito delle informazioni acquisite, generare automaticamente un report
- Il sistema non deve intralciare le attività del medico durante un intervento

a seguito di diversi incontri tra il team di sviluppo e i medici del Bufalini sono stati individuati ulteriori requisiti:

- Per rendere il report più completo, il sistema dovrebbe dar la possibilità di acquisire foto
- Nella scelta dell'hardware per implementare il sistema, occorre evitare di utilizzare dispositivi che vadano ad appesantire l'equipaggiamento del medico
- Offrire la possiblità di gestire il sistema attraverso un approccio *Hand-free*

In relazione ai requisiti sopracitati si procede definendo i casi d'uso del sistema [fig:2.1]. Pensando anche agli scenari relativi, si può già intuire che per poter aggiungere le informazioni riguardo a: farmaci, manovre o parametri vitali, risulta necessario informare il sistema che un intervento sia iniziato. Informando il sistema di ciò, permetterà anche di associarci un report, il quale sarà da considerarsi completo una volta che l'utente notificherà il termine dell'intervento. Un'altra nota che occorre fare, riguarda la visualizzazione dei parametri vitali. Considerando il fatto che il Trauma Leader ogni volta che richiederà di visualizzare i parametri, vorrà poi inserire i dati visti all'interno del report, è stato ritenuto opportuno, anche confrontandoci con i medici, di automatizzare il tutto, in modo tale che il sistema, alla richiesta di visualizzazione dei parametri, inserisca automaticamente il loro valore nel report.

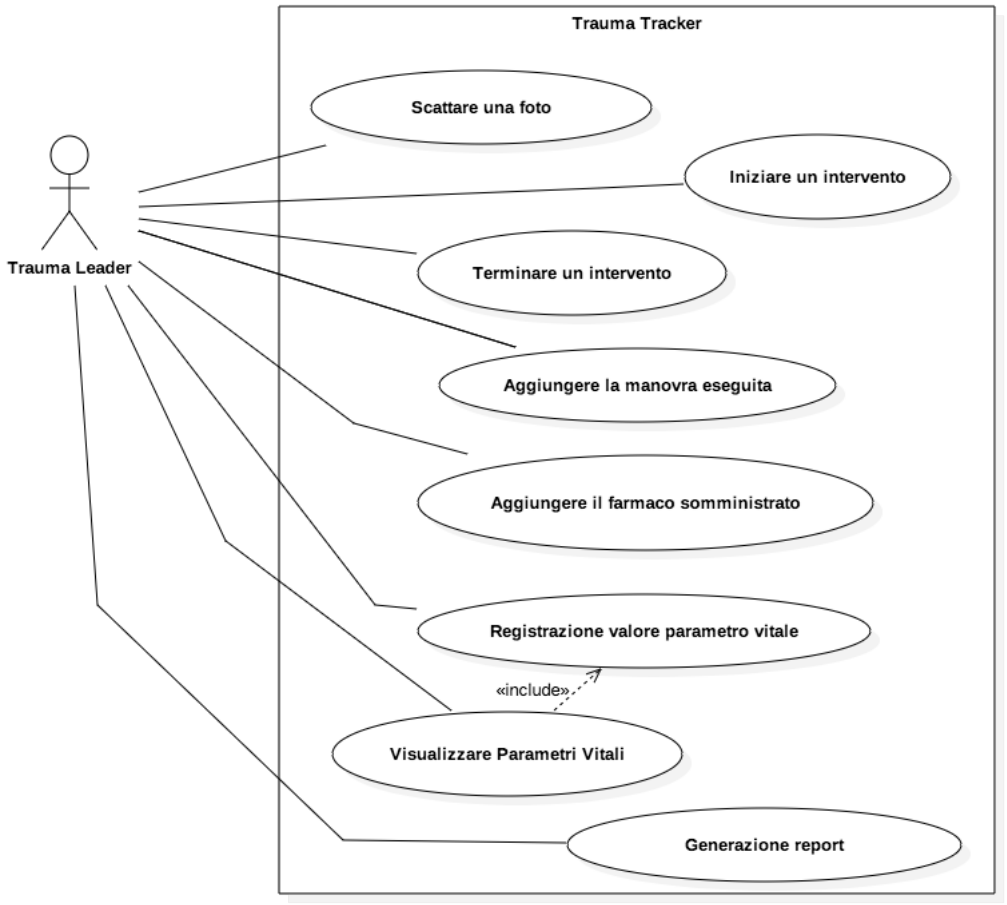


Figura 2.1: Diagramma dei casi d'uso

2.2 Architettura logica

A seguito della definizione dei requisiti e dei casi d'uso, si continui la fase di analisi individuando le macro entità che svolgeranno i diversi compiti e le loro interazioni. Dalla figura 2.2 si può vedere come logicamente possono essere divisi i vari compiti del sistema.

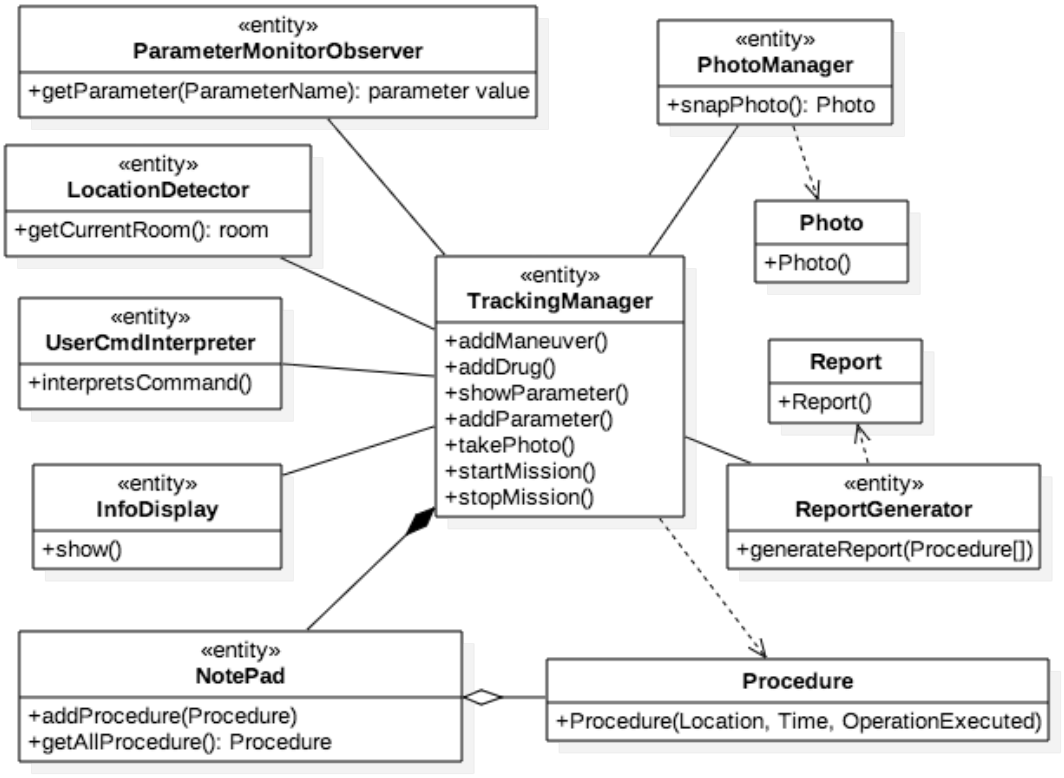


Figura 2.2: Entità logiche che eseguono i compiti

Quando l'utente notifica al sistema, attraverso un comando, quale azione sta eseguendo, esso viene interpretato dall'entità *UserCmdInterpreter*. Questa entità incapsula anche la possibile gestione hands-free dell'input. Una volta interpretato il comando, *UserCmdInterpeter* richiederà al *TrackingManager* di eseguire l'operazione corrispondente. Quest'ultimo costituisce l'entità addetta a gestire la memorizzazione delle procedure, che mano a mano l'utente impartisce o esegue. Per memorizzare una procedura (*Procedure*), oltre all'orario in cui è stata eseguita, è necessario che il *TrackingManager* recuperi l'informazione relativa alla stanza in cui l'utente la svolge. Per fare ciò il *TrackingManager* ricorre all'entità *Location Detector*, la quale si occupa di rilevare in quale stanza si trova l'utente. Una volta recuperate tutte le informazioni, esse vengono salvate sull'entità *NotePad* contenuta all'interno e viene visualizzato all'utente un messaggio di conferma attraverso l'entità *InfoDisplay* [figura:2.3].

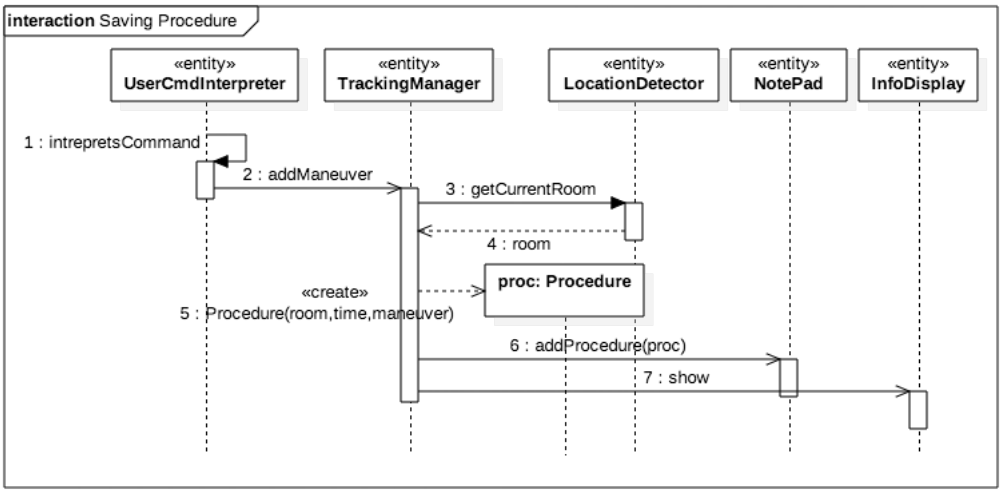


Figura 2.3: Interazione delle entità per la memorizzazione di una procedura

Nel caso l'utente ordina di scattare una foto, il *TrackingManager* richiede al *PhotoManager* di acquisire una foto e, come nel caso precedente, salva le informazioni relative alla fotografia sull'entità *NotePad* [figura:2.4]

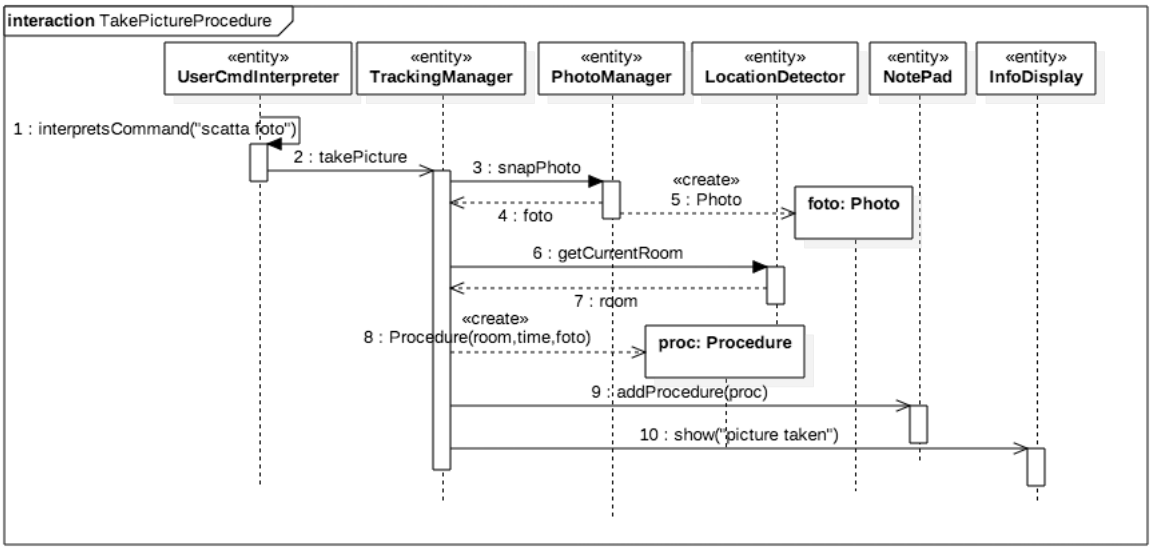


Figura 2.4: Architettura Generale Speech Recognizer

Se l'utente dovesse richiedere la visualizzazione dei parametri vitali, il *TrackingManager* recupera il dato relativo attraverso l'entità *ParameterMonito-*

rObserver, il quale si occupa della comunicazione con i diversi macchinari per la misurazione e, anche in questo caso, dovrà memorizzare la procedura [figura:2.5].

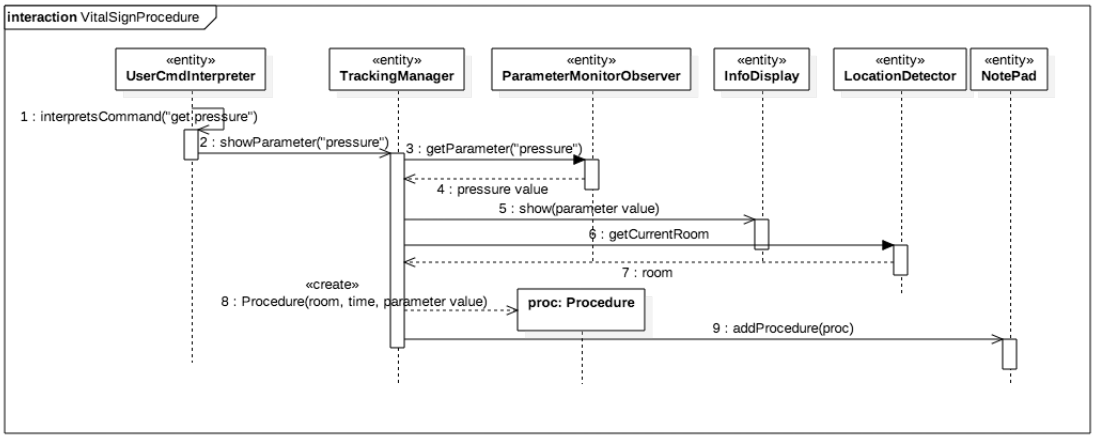


Figura 2.5: Interazione delle entità per l'acquisizione di una foto

Infine nel caso in cui l'utente termini l'intervento. Il *TrackingManager* deve inviare tutte le informazioni memorizzate sul *NotePad* all'entità ReportGenerator, la quale una volta elaborate opportunamente le informazioni, genererà il report finale [figura: 2.6]

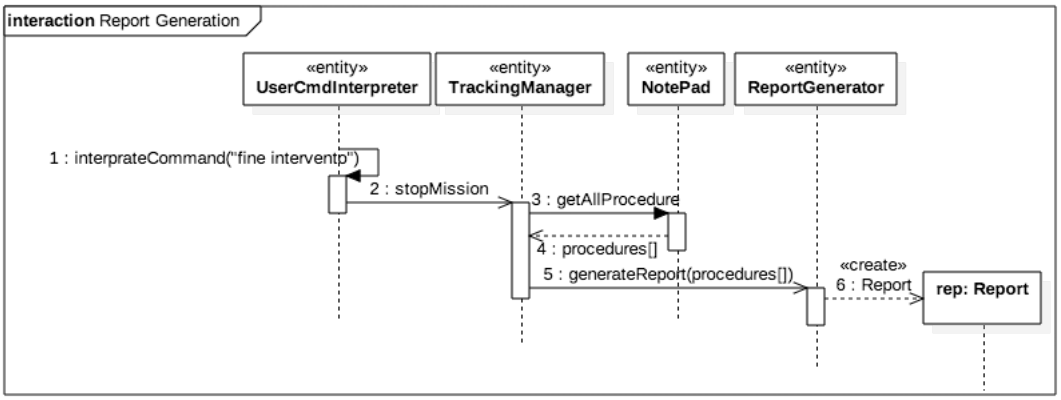


Figura 2.6: Interazione delle entità per la generazioe del report finale

2.3 Progettazione

Riflettendo su quanto emerso durante l'analisi, si può intuire che una soluzione wearable/mobile potrebbe essere particolarmente adatta all'implementazione del sistema, considerando che esso debba seguire costantemente le attività del *Trauma Leader* e debba offrire un'interazione *hands-free*. Ragionando più approfonditamente, insieme ai componenti del *Trauma team*, sulla gestione del report finale, è emerso che risulterebbe particolarmente utile poter consultarlo, direttamente dai loro computer, al di fuori degli interventi. Un'ulteriore osservazione da fare è che, per poter ottenere i dati relativi ai parametri vitali, vi è la necessità di implementare un componente che permetta di interfacciarsi con i vari macchinari, le cui operazioni potrebbero richiedere requisiti non disponibili su sistemi mobile. Da ciò si può dedurre che implementare il sistema solo su un dispositivo centralizzato potrebbe non costituire la soluzione migliore, a tal proposito si è deciso di distribuire il sistema su dispositivi differenti.

2.3.1 Architettura Distribuita

In relazione a quanto elaborato nell'architettura logica [figura:2.2]. Durante la progettazione, è stato quindi deciso di utilizzare uno smartphone Android, su cui eseguire le operazioni riservate all'entità *TrackingManager*. Questo permette di sfruttare direttamente un sistema operativo, che mette a disposizione delle API robuste e ben documentate, rendendo più semplice lo sviluppo ed evitando di perdere troppo tempo sulla gestione di basso livello. Per quanto riguarda l'approccio hands-free, si è ritenuto opportuno utilizzare la voce come possibile input al sistema. Android fornisce già API per sfruttare engine di riconoscimento vocale, quindi lo smartphone gestirà anche i compiti riservati all'entità *UserCmdInterpreter*. Stessa cosa per quanto riguarda le operazioni del *LocationManager*. Si precisa che essendo l'ambiente di utilizzo un locale indoor, per la localizzazione non era possibile l'uso del gps, si è scelto quindi di ricorre ad una soluzione che sfruttasse la tecnologia Beacon. L'uso diretto di uno smartphone per l'output visivo però, potrebbe costituire un ostacolo all'attività del medico, per tale motivo, per gestire le funzionalità dell'entità *InfoDisplay*, si è preferito ricorrere ad un paio di smartglass. In questo caso, considerando che, tra i requisiti del sistema, non vi è la necessità di ricorrere a tecnologie di realtà aumentata e tenendo presente che i medici sono obbligati ad indossare degli occhiali protettivi durante un intervento, dopo uno screening dei dispositivi disponibili sul mercato, si è scelto di utilizzare smartglass della Vuzix (`https://www.vuzix.com/`), precisamente il modello M-100, in attesa dell'uscita del nuovo modello M-300. Tali device rispetto ad altri dispositivi offrono la possibilità di essere agganciati direttamente ad una

propria montatura ed essendo dotati di una fotocamera integrata, è sembrato opportuno sfruttarli direttamente anche come mezzo per acquisire le foto (entità logica *PhotoManager*). Ci si potrebbe chiedere a questo punto, perché non implementare direttamente tutte le funzioni sugli smartglass. Purtroppo sui modelli attualmente disponibili sul mercato, non risulta possibile avere le stesse risorse presenti su uno smartphone. Questo però non vieta in futuro, quando il gap tra smartglass e smartphone sarà colmato, di migrare tutte le funzioni svolte dal telefono direttamente agli occhiali. Per quanto riguarda la gestione della generazione del report finale (entità *ReportGenerator*) è stato scelto di non implementare tali funzioni sullo smartphone, ma progettare un applicativo server che offra anche le funzionalità di un web server. Anche per recuperare i dati dai vari monitor (entità *ParameterMonitorObserver*) è stato scelto di progettare un secondo applicativo server che fungesse da gateway tra il sistema *TraumaTracker* e i monitor dei parametri vitali. [figura:2.7][1].

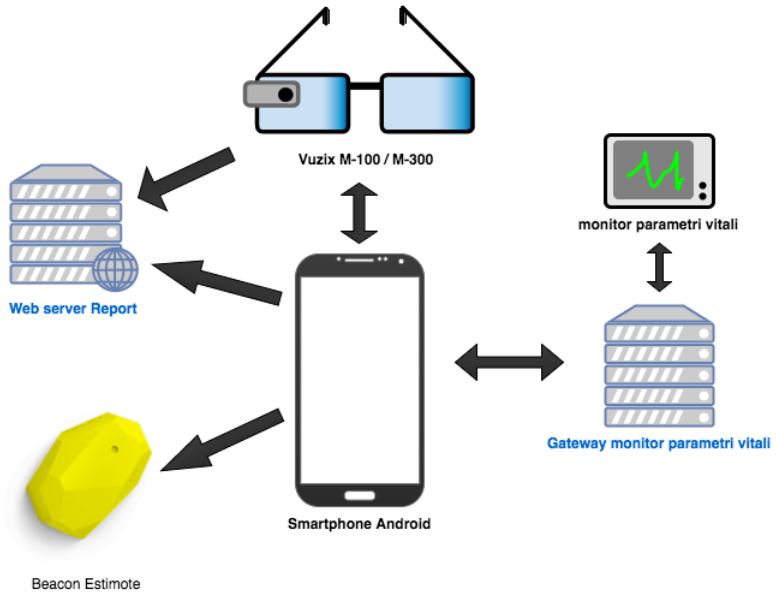


Figura 2.7: Componenti che formano l'intero sistema Trauma Tacker

Detto ciò si ha quindi [figura: 2.8]:

- Una app eseguita sullo smartphone che svolgerà le funzioni dell'entità *TrackingManager*, *UserCmdInterpreter* e *LocationManager*, la quale prenderà il nome di **TraumaTrackerCore**, in sigla *TTCore*

[1]Immagine Beacon tratta da `http://estimote.com/`

- una app in esecuzione sugli smartglass chiamata **TraumaTrackerGlass**, in sigla *TTGlass*, che si occuperà dei compiti riservati alle entità logiche *InfoDisplay* e *PhotoManager*
- un applicativo server chiamato **TraumaReportBase**, che effettuerà le funzioni demandate a *ReportGenerator*
- un secondo applicativo server chiamato *VitalSignGateway* che svolgerà le mansioni di *ParameterMonitorObserver*

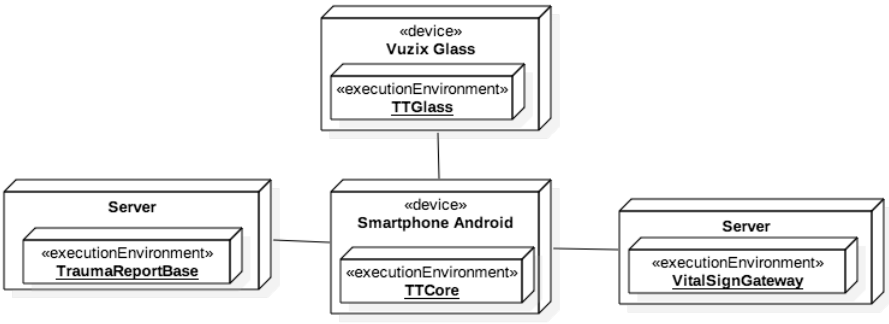


Figura 2.8: Diagramma di deployment delle varie parti del sistema

2.4 Attività del sistema

Una volta deciso come suddividere il sistema nelle varie parti, facendo riferimento anche a quanto elaborato nella parte di architettura logica, risulta possibile ragionare sulle macro-attività che dovrà svolgere il sistema, grazie alle quali, sarà possibile orientarsi per la progettazione delle singole parti. Dalla figura 2.9 si evince che, a seguito dell'acquisizione di una foto, gli occhiali restituiscono solo un riferimento allo smartphone. Questo perché, tenendo in considerazione che l'invio di una fotografia dagli smartglass allo smartphone poteva introdurre notevoli latenze, si è ritenuto più opportuno inviare come risposta dagli occhiali al telefono, solo il riferimento alla fotografia e di mantenere solo sulla memoria degli smartglass il file. Dalla figura 2.9 risulta anche possibile notare quali azioni esso potrà eseguire in parallelo. Si noti ad esempio, il caso della richiesta di parametri vitali. Lo smartphone prima di tutto dovrà in qualche modo recuperare i parametri dal gateway dei monitor di misurazione. Una volta ricevuto il valore di interesse, dovrà inviare il dato agli smartglass per mostrarlo all'utente e, nello stesso momento, dovrà recuperare il luogo in cui si trova quest'ultimo e prender nota riguardo al valore del parametro recuperato.

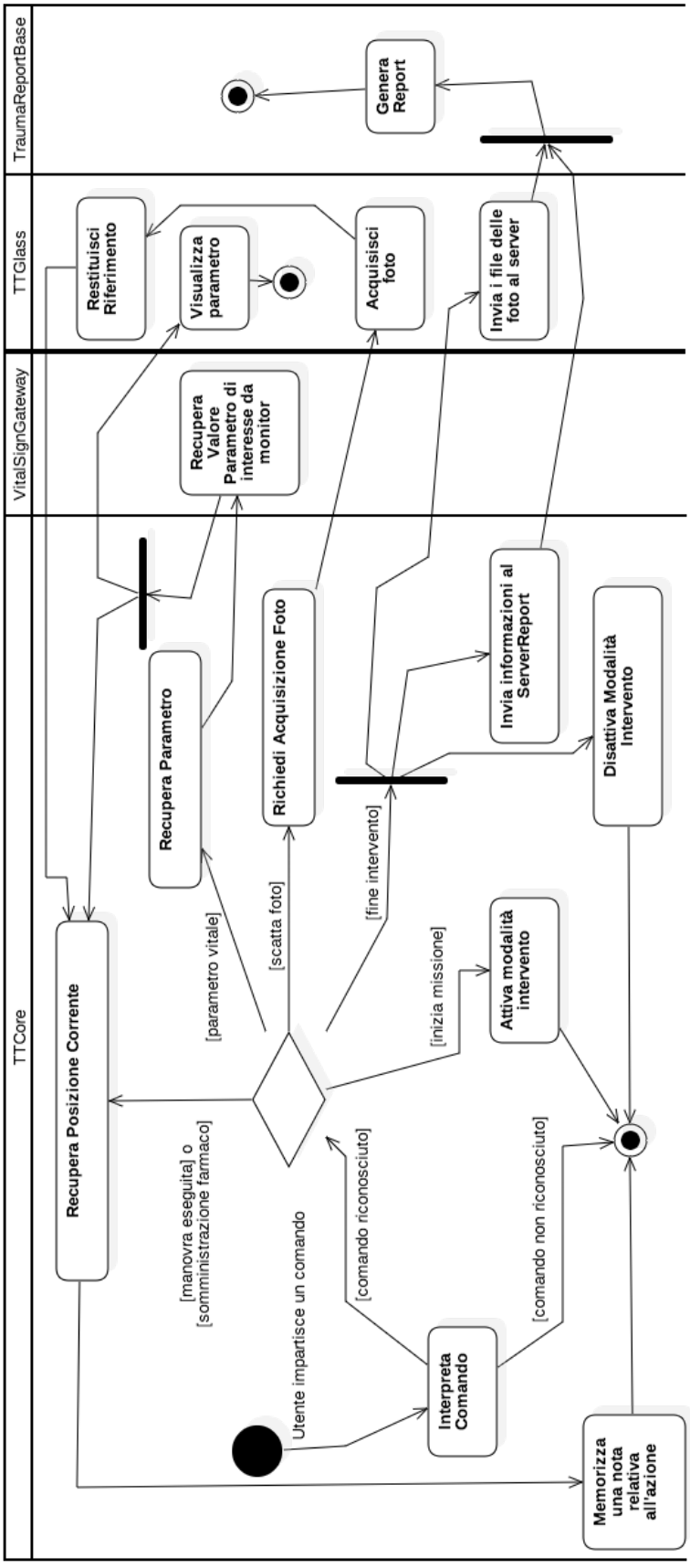


Figura 2.9: Attività eseguite a seguito di un comando dell'utente

Un altro caso, è la fine di un intervento. L'applicazione eseguita sullo smartphone, oltre a modificare il suo stato interno, dovrà inviare i dati raccolti fino a quel momento, al server che gestisce il report finale. Allo stesso tempo dovrà notificare anche agli smartglass del termine dell'intervento, in modo tale che anch'essi inviino le foto acquisite al *TraumaReportBase*. Quest'ultimo una volta che avrà ricevuto tutte le informazioni sia dallo smartphone, che dagli smartglass sarà in grado di generare il report finale.

2.5 TraumaTrackerCore

Per quanto riguarda la parte di *TraumaTracker* eseguita sullo smartphone, considerando la necessità di dover creare un sistema reattivo, che supportasse interazioni asincrone tra diversi componenti, è risultato interessante modellare l'architettura del software attraverso un approccio orientato agli Agenti [21]. Nella fattispecie, è stato utilizzato il metamodello A&A (Agent & Artifact) [22], che prevede l'utilizzo nella modellazione di due entità principali: gli agenti e gli artefatti. Gli agenti sono usati per modellare entità proattive, che eseguono uno o più compiti in autonomia, interagendo con l'ambiente in cui sono immersi. Gli artefatti sono entità passive con le quali è possibile modellare tale ambiente, rappresentando risorse e servizi. Guardando al futuro, questa scelta permette già di predisporre il sistema a possibili modifiche che integrino agilmente comportamenti oltre che reattivi anche proattivi.

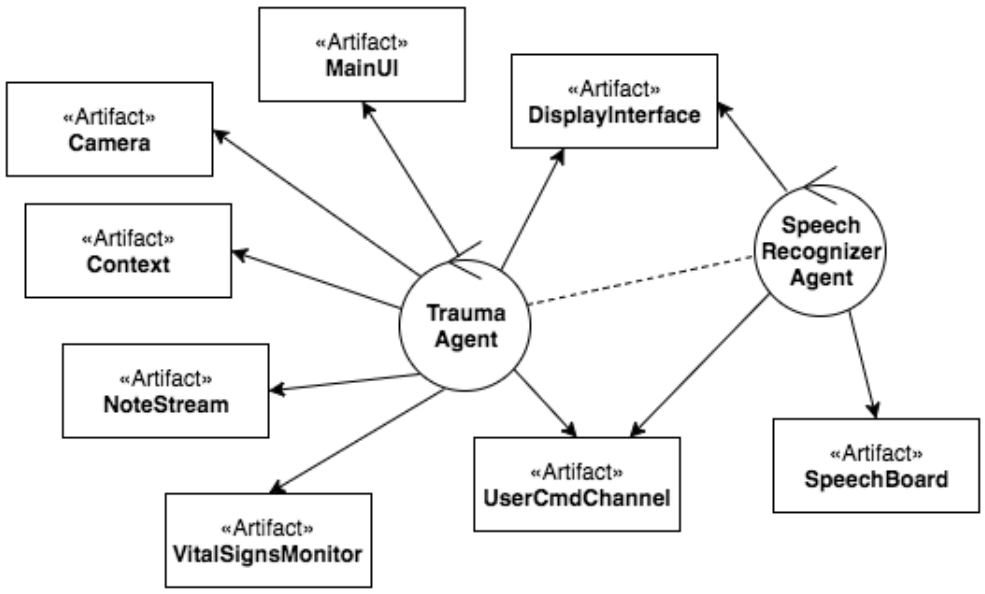


Figura 2.10: Architettura ad Agenti - Smartphone App

2.6 Agenti

In fase di progettazione, come si può vedere dalla figura [fig:2.10], si è pensato di modellare la logica di controllo del software attraverso due agenti.

Trauma Agent Il *Trauma Agent* costituisce l'agente principale dell'applicazione. Oltre ad occuparsi del setup del sistema, esso, a seconda del comando dell'utente, notificatogli dallo *SpeechRecognizerAgent* attraverso lo *UserCmdChannel*, ha il compito di memorizzare sul *NoteStream* le varie informazioni relative all'attività eseguita, recuperando, dall'artefatto *Context*, la posizione corrente. Se il comando corrisponde alla richiesta di visualizzazione dei parametri vitali,il *Trauma Agent*, attraverso l'artefatto *VitalSignMonitor*, prende il valore relativo al parametro e lo visualizza sugli occhiali(sfruttando l'artefatto *DisplayInterface*), successivamente ne memorizzerà anche il valore. Nel caso l'utente chieda di scattare una foto, il *Trauma Agent*, sfruttando l'artefatto *Camera*, richiede agli smartglasses di acquisire una foto.

SpeechRecognizerAgent Lo *SpeechRecognizerAgent* rimane in ascolto costantemente dei comandi dell'utente, attraverso l'artefatto *SpeechBoard*. Ad ogni comando ricevuto, lo interpreta e notifica il *Trauma Agent* sfruttando lo *UserCmdChannel*. Ha anche il compito di visualizzare sugli smartglass, attraverso l'artefatto *DisplayInterface*, quando l'engine del riconoscimento vocale è attivo o no (ulteriori spiegazioni a riguardo sono riportate nel prossimo capitolo)

2.7 Artefatti

Successivamente andiamo a presentare i vari componenti del sistema che sono stati modellati come artefatti.

Camera incapsula la gestione della comunicazione con gli smartglass, per l'acquisizione di una foto. Come detto in precedenza, a seguito dell'acquisizione delle foto gli smartglass restituiscono solo il riferimento alla foto scattata. Sarà poi in fase finale il server dei report, che a seguito della terminazione di un intervento, quando riceverà dallo smartphone l'elenco dei comandi registrati e dagli smartglass i file relativi alle fotografie, riassocerà il riferimento con il file corrispondente.

Context Tale artefatto incapsula tutta la gestione della comunicazione con i Beacon, fornendo al *Trauma Agent* l'informazione relativa alla stanza in cui

si trova l'utente. Come altri artefatti, permette all'agente di ignorare tutte le problematiche di basso livello legate alla gestione della scansione dei Beacon.

DisplayInterface si occupa della comunicazione con gli smartglass per gestire la visualizzazione delle varie informazioni attraverso il display montato sugli occhiali.

MainUi è un artefatto che racchiude la gestione dell'interfaccia grafica su smartphone, essa al momento viene utilizzata come schermata di log per i vari eventi dell'applicazione. In essa è racchiusa anche la gestione del setup iniziale (ad esempio incapsula l'inizializzazione della connessione con gli occhiali).

NoteStream rappresenta l'astrazione di un taccuino (entità logica *NotePad* 2.2), sul quale il *Trauma Agent* aggiunge le varie procedure eseguite dall'utente sotto forma di note. Si precisa che ciò che viene inserito nel *NoteStream* non è il report finale, ma solo l'insieme delle varie attività compiute dall'utente, infatti, come accennato prima, a seguito dell'acquisizione di una foto da parte del sistema, in tale artefatto verrà aggiunta la nota contenente data, ora, luogo, e il riferimento alla foto. Quest'ultimo sarà sostituito con il file corrispondente solo a intervento terminato dal *ServerReport* e solo successivamente avremo il report finale.

SpeechBoard incapsula la gestione dello speech engine utilizzato per interpretare i comandi dell'utente (ulteriori dettagli implementativi sono rimandati ai prossimi capitoli). Attraverso tale artefatto lo *SpeechRecognizerAgent* riceve i comandi impartiti dall'utente e notifica opportunamente il *Trauma Agent* su quale operazione vada eseguita sfruttando l'artefatto *UserCmdChannel*.

UserCmdChannel incapsula la gestione delle notifiche verso il *TraumaAgent* riguardo ai comandi che l'utente vuole eseguire.

VitalSignsMonitor incapsula la gestione della comunicazione dello smartphone con il server gateway, che legge i parametri vitali dai vari monitor, permettendo al *Trauma Agent* di ignorare tutte le problematiche di basso livello collegate alla comunicazione in rete.

2.7.1 Note su JaCa-Android

Avendo Android come sistema operativo sullo smartphone, per l'implementazione è stata utilizzata la piattaforma JaCa-Android [23]. Essa, nascendo

dall'unione delle piattaforme Jason e Cartago, permette l'implementazione di applicazioni ad agenti su Android. Jason viene usato come linguaggio di programmazione per implementare e creare Agenti[23], mentre Cartago è un framework ed una infrastruttura per la programmazione e l'esecuzione di sistemi multi-agente[24]. Cartago sfruttando la nozione di artefatto permette di definire la struttura e l'ambiente in cui gli agenti lavorano [23]. Essendo costruito sopra Android potremmo fare un'associazione tra le componenti del sistema operativo e le componenti di JaCa. Attraverso gli *Agenti* vengono modellate le logiche di controllo dei vari task di una applicazione, mentre attraverso gli *artefatti* vengono modellati i servizi e i componenti messi a disposizione dal sistema Android. Grazie alla nozione di *artefatto*, la quale permette anche di implementare facilmente meccanismi di coordinazione e sincronizzazione, vengono nascosti agli agenti dettagli di basso livello legati alla gestione di Android [23].

2.8 TraumaTrackerView

Come si può vedere dalla figura 2.7. Il secondo device che forma il sistema oltre allo smartphone sono gli smartglass Vusix M-100/M-300 [fig:2.11]. Su questi device si trova Android ed a differenza del caso precedente, dovendo implementare molte meno funzioni rispetto allo smartphone, si è scelto di non utilizzare un approccio ad agenti, ma di implementare una classica applicazione Android orientata agli oggetti. L'applicazione gestisce la comunicazione con lo smartphone, dal quale riceve informazioni riguardo lo stato dell'engine del riconoscimento vocale e dei comandi impartiti dall'utente. A seconda del comando, l'applicazione mostra all'utente le diverse informazioni attraverso un'interfaccia opportunamente progettata[2]. Nel caso in cui il comando dell'utente sia quello di scattare una foto, l'applicazione gestisce l'acquisizione e, come detto prima, restituisce il riferimento del file allo smartphone. Oltre alle funzioni precedenti, in caso di terminazione della fase di intervento da parte dell'utente, l'applicazione gestisce l'invio delle varie immagini acquisite al *TraumaReportBase*. Per evitare troppi consumi durante l'utilizzo, è stato deciso di effettuare l'invio delle immagini solo a fine intervento

[2]rimando ai capitoli successivi i dettagli dell'implementazione dell'interfaccia

(a) Vuzix M-100

(b) Vuzix M-300

Figura 2.11: Modelli di smartglass usati per l'implementazione. *Immagini tratte da* `www.vuzix.com`

2.9 TraumaReportBase

Riprendendo il requisito detto inizialmente, riguardo alla possibilità di fornire l'accesso ai report terminati anche da altri computer dei medici, facilitandone la consultazione e lo studio, a tale scopo è stato ritenuto opportuno implementare un applicativo server, che fornisse anche la funzionalità di web server, da eseguire su un device a parte rispetto allo smartphone. Tale soluzione permette di poter consultare i report anche quando lo smartphone è spento o non ha *TraumaTrackerCore* attivata. Tale server al termine di un intervento riceve le informazioni memorizzate dallo smartphone e le varie foto scattate dagli smartglasses. Dopo aver sostituito i vari riferimenti alle fotografie con i corrispondenti file, genera una pagina html, accessibile comodamente da un computer.

In futuro tale parte del sistema sarà modificata in modo da fornire anche la possibilità di scaricare il report completo anche in altri formati, come ad esempio il formato Excell.

2.10 VitalSignGateway

All'interno dell'ospedale, per la misurazione dei parametri, potrebbero venir usati dispositivi di produttori diversi, il cui interfacciamento potrebbe richiedere l'implementazione di approcci differenti, a causa dei possibili protocolli utilizzati per la comunicazione. Per tale motivo è stato ritenuto opportuno progettare un server che funga da gateway tra lo smartphone e i vari monitor, il quale permetta allo smartphone di recuperare i dati relativi ai parametri di interesse utilizzando una sintattica e una semantica comune per la rappre-

sentazione dei dati relativi ai parametri, ignorando i dettagli implementativi richiesti per il recupero del dato dal macchinario. Ciò permette di rendere il sistema maggiormente scalabile, in relazione al cambiamento degli strumenti direttamente utilizzati in ospedale per le misurazioni.

Al momento sul server è stata implementata solo l'interazione con monitor prodotti da Drager[3]. Per essere maggiormente precisi, con questi monitor per questioni tecniche, non è possibile comunicare direttamente con essi. Perciò il server recupera le informazioni attraverso un secondo server sviluppato da Draeger, che fornisce tutti i dati misurati dai monitor prodotti da lei

Figura 2.12: Monitor Draeger M-540 in dotazione all'ospedale Bufalini di Cesena

Potrebbe sorgere il problema di come recuperare i dati dai dispositivi che non forniscono modalità di interfacciamento. A tal proposito parallelamente allo sviluppo di **TraumaTracker** è stato iniziato lo sviluppato di un sistema che attraverso una webcam riesce ad interpretare i dati mostrati da un monitor e a fornirli al sistema **TraumaTracker**.

[3]immagine tratta da:`http://www.draeger.com/sites/enus_ca/Pages/Hospital/Infinity-M540-monitor.aspx`

2.11 Protocolli di comunicazione utilizzati

Per la comunicazione tra i vari devices, il team si è confrontato su quali potevano essere le soluzioni migliori. Iniziamo analizzando l'interazione tra Smartphone e Smartglass. In questo caso era possibile scegliere o di utilizzare la tecnologia Bluetooth oppure Wifi. Bluetooth nonostante offra velocità di trasmissioni più basse rispetto a Wifi, garantisce un consumo inferiore a livello di batteria. Considerando che i due dispositivi vengono utilizzati a breve distanza l'uno dall'altro e tenendo presente che la dimensione dei messaggi non è elevata, l'uso del protocollo Bluetooth è sembrata la soluzione migliore. Tale protocollo viene utilizzato anche per la comunicazione tra Beacon e smartphone. In questo caso però si ricorre alla versione Bluetooth Smart 4.0, la quale garantisce di avere livelli di consumi ancora più bassi sui dispositivi Beacon. Per quanto riguarda la comunicazione tra gli smartglass e il web server e tra quest'ultimo e lo smartphone viene utilizzata la rete Wifi. Stessa cosa vale anche per la comunicazione tra lo smarphone e il server gateway. La scelta di utilizzare wifi deriva anche dal fatto che i device si possono trovare ad elevate distanze l'uno dall'altro e dato che lo smatphone si muove con l'utente attraverso le varie stanze, sfruttando l'infrastruttura di rete Wifi dell'ospedale ci permette di avere una comunicazione stabile.

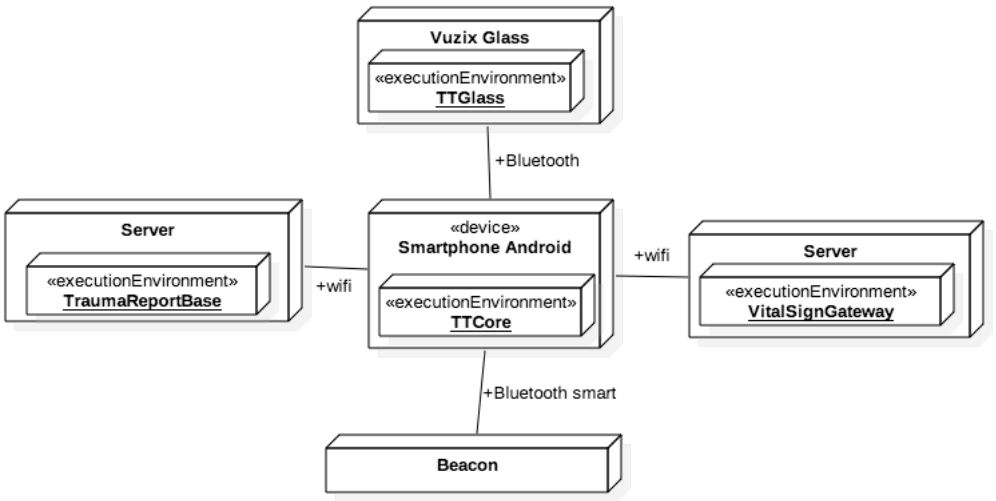


Figura 2.13: Protocolli di comunicazione utilizzati tra i vari dispositivi

2.12 Note di sviluppo

Ora che sono state introdotte le varie componenti del sistema **TraumaTracker** risulta necessario fare un piccola nota sull'implementazione. Come detto nella premessa, durante l'intero sviluppo sono stati creati diversi prototipi per far provare agli utenti, le varie funzioni del sistema, che venivano a mano a mano implementate. Dato che lo sviluppo delle varie parti è stato portato avanti in parallelo, fin da subito in fase di progettazione sono stati definiti i modi con cui dovevano interfacciarsi le varie parti del sistema, ciò ha permesso ad ogni componente del team di simulare, attraverso test o componenti sviluppate ad-hoc, l'interazione con il resto del sistema. Questo approccio ha portato a velocizzare notevolmente la fase di integrazione nel momento della creazione dei vari prototipi incrementali.

Capitolo 3

Il Sottosistema Speech-Recognizer

Dopo aver introdotto il sistema *TraumaTracker* illustrandone l'analisi e la progettazione fatta a livello di team, si passa ora alla descrizione delle parti che ho preso in carico personalmente, partendo dallo Speech Recognizer.
Come descritto in precedenza, *TraumaTrackerCore* per interpretare i comandi impartiti dall'utente utilizza l'artefatto *SpeechBoard*, a sua volta esso, per offrire le funzionalità di speech recognition, internamente fa ricorso ad una libreria chiamata *SpeechRecognizerLib* progettata e sviluppata da me.

3.1 Analisi

Tra i possibili approcci hands-free utilizzabili, in fase di analisi si è optato per l'uso di comandi vocali per controllare il sistema. Tale decisione è stata presa immaginando il sistema *TraumaTracker* come se fosse un vero e proprio assistente del *Trauma Leader* e che quindi, come quest'ultimo impartisce i comandi al resto del *Trauma Team*, impartirà i comandi anche a *TraumaTracker*.
In relazione alle varie attività compiute durante un intervento, è stato fornito dai medici un primo elenco di comandi che il sistema dovrà riconoscere [tabella: 3.1]. Tale elenco viene diviso in: manovre rianimatorie, parametri vitali e medicinali (farmaci/infusioni). Per quanto riguarda i farmaci e le infusioni, essi hanno sempre associata una quantità relativa alla somministrazione e un'unità di misura. L'unità di misura è sempre fissa rispetto al medicinale pronunciato, mentre la quantità varia a seconda del caso.
Se i comandi relativi alle manovre e ai medicinali si riferiscono sempre ad azioni compiute e quindi solo da memorizzare nel report, per quanto riguarda i parametri vitali, invece possiamo avere due opzioni:

- La prima, si riferisce all'intenzione dell'utente di visualizzare il valore misurato dal macchinario interfacciato con il *VitalSignGateway*. L'utente, in questo caso, pronuncerà solo il nome del parametro e, come detto nel capitolo precedentemente, sarà il sistema che recupererà il valore, lo mostrerà all'utente e lo memorizzerà sul report.

- Potrebbero però esistere parametri non misurati da macchinari, ma basati su valutazioni fatte dal medico al paziente. In questo caso quando il medico pronuncerà il nome del parametro, vi assocerà anche un valore numerico. La sua intenzione, in questo caso, sarà solo quella di voler memorizzare l'informazione nel report.

Manovre Rianimatorie	**Parametri Vitali**	**Farmaci ed Infusioni**
Intubazione Orotracheale	Vie Aeree	Cristalloidi
Pelvic Binder	Pressione Sistolica	Zero Negativo
Infusore a Pressione	Pressione Diastolica	Tranex
Sutura Ferita Emorragica	EtCO2	Acido Tranexamico
Toracotomia Resuscitativa	Pupille	Morfina
Dreanaggio Toracico	Motorio	Crioprecipitati
Catetere Alto Flusso	Decompressione Pleurica	Emazie Concentrate
Posizionamento Fissatore Esterno	Frequenza Cardiaca	Fibrinogeno
Reboa	Saturazione	Mannitolo
Tpod	Temperatura	Propofol
	Occhi	Ketamina
	Verbale	Succinilcolina
		Soluzione Ipertonica
		Plasma Fresco Congelato
		Poolpiastrine
		Midazolam
		Fentanil
		Rocuronio
		Tiopentone

Tabella 3.1: Elenco comandi fornito dai medici

Oltre ai termini medici definiti sopra, il sistema dovrà riuscire a riconoscere anche alcuni comandi di controllo, che sono: *inizia intervento* per avviare la

fase di intervento, *fine intervento* per terminare la fase di intervento, *scatta foto* per scattare una foto, *trauma pausa* per mettere in pausa il riconoscimento dei termini medici durante un intervento e *trauma ascolta* per riattivare il riconoscimento dei termini medici. In un secondo momento è stato aggiunto all'elenco dei termini medici anche il comando generale *Parametri Vitali*. Grazie a tale comando l'utente, invece che richiamare solo la visualizzazione di un parametro vitale, richiama subito la visualizzazione contemporanea di quelli principali prestabiliti. Considerando che la rete su cui si appoggia *TraumaTracker* potrebbe essere una LAN non connessa ad Internet, risulta quindi necessario utilizzare un algoritmo di speech recognizing, che non si basi su servizi online. In relazione a quanto detto, riguardo all'unità di misura dei medicinali, discutendone anche con il *Trauma Team*, si è deciso che il comando relativo all'infusione o al farmaco sarà formato da *termine + quantità* e la relativa unità di misura sarà associata automaticamente dal sistema.

3.1.1 Requisiti

Riflettendo su quanto elaborato sopra, è possibile formalizzare i requisiti che dovrà rispettare il riconoscimento vocale:

- Riconoscere i termini forniti dai medici
- Riconoscere per i medicinali anche le quantità associate e collegare la relativa unità di misura
- Se presente, riconoscere per i parametri vitali il possibile valore associato.
- Riconoscere i comandi di controllo ed adattarsi in funzione di essi.
- Basarsi su un engine di riconoscimento offline.

Ragionando sui comandi di controllo, si deduce che il sistema di speech recognition potrà trovarsi in diverse situazioni, nelle quali il riconoscimento di alcuni termini potrebbe essere attivo o meno [fig:3.1]. Appena avviato il sistema si troverà in uno stato di *Intervento non attivo*. In questo stato il riconoscitore dovrà essere sensibile solo al comando di controllo `inizia intervento`. Nel momento in cui l'utente inizierà un intervento (transizione allo stato *Intervento Attivo*), il sistema dovrà reagire attivando anche il riconoscimento di tutti i restanti comandi di controllo. Risulta possibile individuare due ulteriori sotto-stati. Appena il sistema entra nello stato *Intervento Attivo*, anche il riconoscimento dei termini medici sarà abilitato, il quale rimarrà attivo fino a quando l'utente non terminerà l'intervento (ritorno allo stato *Intervento non attivo*) oppure non metterà in pausa il riconoscimento, in quest'ultimo caso

il sistema passerà nel sotto-stato *Riconoscimento Termini Medici in pausa* in cui solo comandi di controllo sono abilitati.

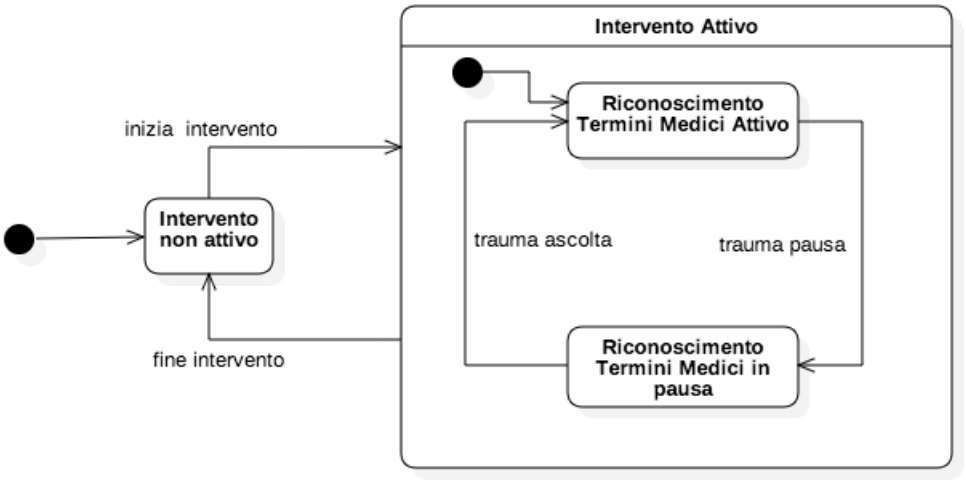


Figura 3.1: Stati del riconoscimento

Considerando quanto detto, è possibile affermare che i comandi di controllo dovranno avere una priorità rispetto ai termini medici, quindi, in fase di checking del comando riconosciuto, andranno controllati per primi.

3.2 Progettazione

Una volta focalizzati i requisiti da soddisfare e gli stati in cui il riconoscitore si può trovare, si può passare alla fase di progettazione. Prima considerazione da fare è che ci si trova a dover progettare uno speech recognizer ad-hoc, che sarà eseguito su un sistema Android, quindi occorre analizzare se e quali strumenti mette a disposizioni il sistema operativo per utilizzare un riconoscimento vocale di base.

3.2.1 API Android

Nel caso si voglia implementare il riconoscimento vocale in una propria applicazione, tra le varie API (application programming interface) fornite dal sistema operativo, ve ne sono alcune dedicate al riconoscimento vocale, contenute nel package `android.speech`[1]. In tale package si possono trovare:

[1]API disponibili alla pagina `https://developer.android.com/reference/android/speech/package-summary.html`

SpeechRecognizer: una classe che fornisce l'accesso al servizio di speech recognition. Si precisa che il servizio di speech recognition non è implementato direttamente dal sistema operativo, ma fa parte di applicazioni terze. Infatti questa classe fornisce due modi per essere istanziata: fornendo il riferimento al servizio specifico da usare oppure, nel caso non venga specificato nulla, la classe automaticamente recupera il componente impostato come predefinito, che fornisce il servizio di riconoscimento vocale. Una volta istanziata, tale classe tra i metodi forniti, ne offre uno per avviare il recognizer, il quale come parametro in input accetta un `Intent`, con cui vengono specificate le opzioni di riconoscimento. Android obbliga ad invocare i metodi di questa classe solo dal main thread.

RecognizerIntent: una classe che fornisce le opzioni (sotto forma di costanti), che possono essere inserite nell'intent con cui viene avviato il riconoscimento vocale. Purtroppo il funzionamento di alcune opzioni varia a seconda dell'implementazione del componente che offre il servizio di speech recognition, quindi alcune potrebbero non avere effetto.

RecognitionListener: un'interfaccia che definisce le callback con le quali ricevere notifiche, a seguito degli eventi dello SpeechRecognizer, contenenti risultati delle interpretazioni o messaggi di errore. Dato che lo *SpeechRecognizer* viene eseguito sul `main thread`, occorre tenere a mente che anche le callback di *RecognitionListener* verranno eseguite in esso.

3.3 Engine di riconoscimento

In base a quanto detto nella descrizione della classe *SpeechRecognizer*, vi è la necessità di ricercare un engine per eseguire il riconoscimento di base. Per il mondo Android, esistono diversi speech engine da poter utilizzare per implementare il riconoscimento vocale. Purtroppo quasi tutti offrono un servizio basato su elaborazioni in cloud, mentre solo alcuni permettono di effettuare un riconoscimento offline. Tra questi troviamo *Google speech engine*.

3.3.1 Google Speech Engine

L'engine di riconoscimento vocale di Google è integrato all'interno dell'applicazione *Google Search app*[2] per Android. Quindi per poter essere utilizzato occorre installare tale applicazione sullo smartphone.

[2]Link della app sullo store Google Play - `https://play.google.com/store/apps/details?id=com.google.android.googlequicksearchbox&hl=it`

In genere i servizi di riconoscimento vocale, per funzionare, sfruttano algoritmi di intelligenza artificiale. Per quanto riguarda quelli in cloud, essi hanno la possibilità di eseguire elaborazioni su una gran mole di dati. D'altro canto i vari programmi disponibili su sistemi desktop, che offrono un servizio offline, per funzionare non possono fare altrettanto. Infatti prima di offrire prestazioni buone, necessitano di periodi di training, nei quali all'utente viene richiesto di leggere dei testi predefiniti ad alta voce, in modo che il sistema apprenda e adatti l'algoritmo di riconoscimento. L'engine offline offerto da Google, invece riesce a offrire fin da subito prestazioni buone senza richiedere fasi di training. In realtà, l'algoritmo alla base dell'engine ha già subito dei periodi di training, infatti come descritto all'interno di [13], il team che l'ha sviluppato, ha allenato l'algoritmo direttamente su campioni di pronunce estratti da ricerche vocali sul motore di Google(un quantitativo di circa 2,000 ore) e per renderlo robusto a fronte di disturbi ambientali, ha ripetuto più volte il training fornendo versioni distorte degli stessi campioni simulando rumori e riverberi, fino ad un totale di 20 versioni distorte differenti per ogni campione. Oltre al training subito, le prestazioni dell'engine sono dovute anche ad algoritmi di intelligenza artificiale usati per implementarlo, tra cui: un modello quantizzato Long Short-Term Memory allenato con Connectionist temporal classification, che permette di predire fonemi indipendenti dal contesto. Oltre a questi due algoritmi, sono stati anche utilizzati: una compressione Singular Value Decomposition e una forma di interpolazione Bayesiana.[13]

L'engine di Google, oltre ad offrire un servizio offline, può sfruttare anche un servizio online. Esso in automatico, a seconda che il device sia connesso o meno alla rete, sceglie tra modalità online e offline. Per forzare solo l'utilizzo della modalità offline, dalla versione Android API level 23 è possibile utilizzare l'opzione di configurazione `EXTRA_PREFER_OFFLINE` contenuta in *RecognizerIntet*.

3.3.2 Problematiche

A seguito di diverse prove effettuate, si è dedotto che l'engine offline di Google sfrutta un vocabolario limitato, in cui non sono presenti tutte le parole della lingua italiana e, tra quelle mancanti, vi sono diverse che compongono i comandi da riconoscere . Purtroppo tale dizionario non è modificabile manualmente e non risulta possibile inserire i termini che mancano o limitare il set di vocaboli a quelli di interesse. Nelle prove eseguite, si è notato che per quei termini che non sono presenti nel vocabolario, quando l'utente li pronuncia, il sistema restituisce parole simili. Ad esempio, *cristalloidi* viene interpretato dal sistema come *cristalli di*. Altro particolare è che quando vengono pronunciati dei numeri, alcuni di essi vengono scritti in parola invece che in cifre.

3.4 Architettura Riconoscimento

Si va ora ad illustrare come è stato adattato, attraverso opportune tecniche, il riconoscimento vocale di base al caso di studio. Si parte introducendo l'interfacciamento con l'API di Android descritte precedentemente, le quali permettono di utilizzare l'engine di riconoscimento di base.

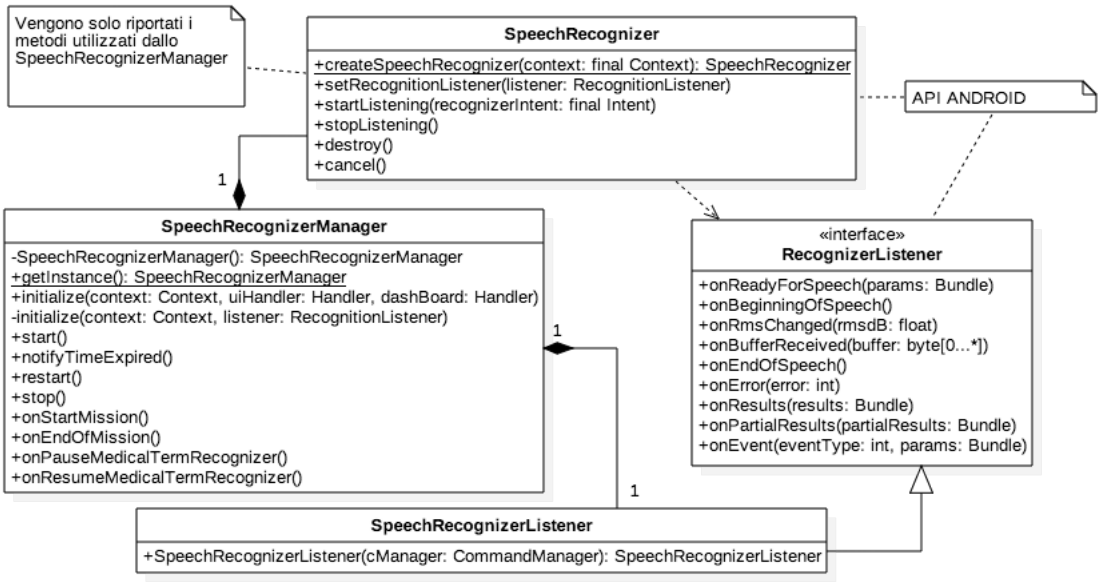


Figura 3.2: Interfacciamento con le API di Android

Come mostrato nella figura 3.2. È possibile notare che *SpeechRecognizer* viene incapsulato in *SpeechRecognizerManager*, quest'ultimo permette di inizializzare il recognizer e gestirne avvio, blocco e riavvio.
SpeechRecognizerManager incapsula anche *SpeechRecognizerListener*, il quale costituisce l'implementazione per il listener del riconoscitore vocale di base. Esso verrà notificato dai vari eventi che vengono generati dallo speech engine, compreso il risultato dell'elaborazione che quest'ultimo riesce a fare.
Lo *SpeechRecognizerManager* quando inizializza lo *SpeechRecognizer*, crea un'istanza di esso passandogli il context, attraverso il quale si interfaccerà con lo speech engine, successivamente vi registra lo *SpeechRecognizerListener*. Una volta inizializzato verrà avviato l'ascolto.

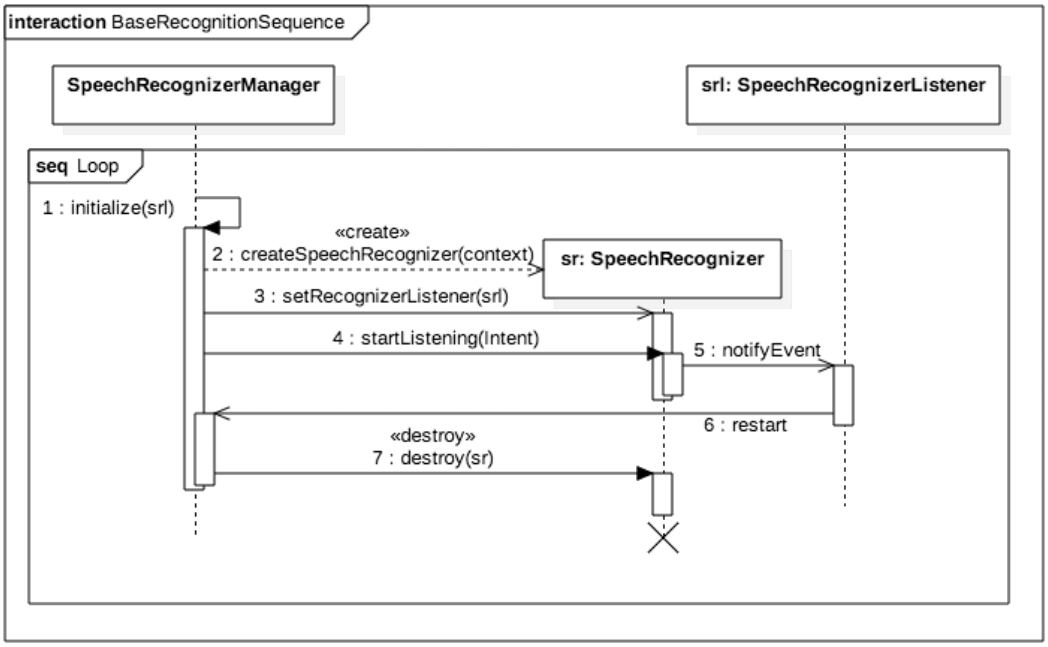


Figura 3.3: Loop riconoscimento di base

Di default, l'istanza *SpeechRecognizer* una volta che restituisce un risultato si ferma. Considerando che nel nostro caso di studio il sistema dovrebbe essere continuamente in ascolto, lo *SpeechRecognizerListener*, una volta che riceve il risultato da parte dello *SpeechRecognizer*(sia esso la notifica di un errore o il risultato del riconoscimento di base), riavvia l'ascolto attraverso *SpeechRecognizerManager*. In questo caso l'istanza dello *SpeechRecognizer* viene eliminata e verrà ripetuta l'inizializzazione [fig:3.3].
Come è possibile osservare dalla figura 3.4, *SpeechRecognizerManager* incapsula anche la classe *CommandManager*. Tale classe gestisce la semantica dei risultati forniti dallo speech engine, analizzando se corrispondono a comandi di controllo oppure, sfruttando *MedicalSpeechIntepreter*, controlla se si trattano di termini medici.
All'interno di *CommandManager* sono racchiuse tutte quelle tecniche che permettono di adattare l'engine di base di Google al caso di studio analizzato.

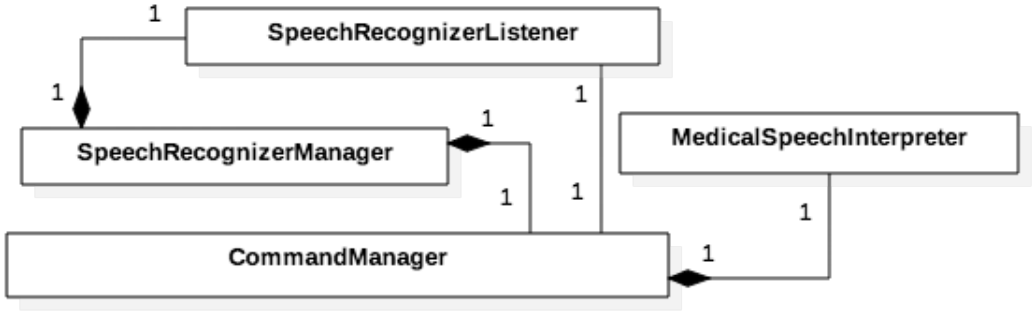


Figura 3.4: Architettura generale *SpeechRecognizerLib*

3.4.1 Interazione tra Libreria e SpeechBoard

Prima di entrare più a fondo nel funzionamento degli algoritmi implementati, occorre capire come l'artefatto *SpeechBoard* interagisce con il recognizer. Esso accede ai servizi di riconoscimento vocale attraverso il singleton *SpeechRecognizerManager*. Per essere usato lo *SpeechRecognizerManager* richiede di essere inizializzato passandogli un riferimento a tre elementi: il primo è l'application context con il quale accederà allo speech engine, il secondo(`uiHandler`) rappresenta l'observer che vuole essere notificato dei risultati delle elaborazioni dello *SpeechRecognizerManager*, mentre il terzo (`dashBoardHandler`) rappresenta l'observer che vuole essere notificato quando viene attivato o disattivato il riconoscimento vocale di base, in modo da avvisare l'utente in che momento può parlare oppure no. All'interno di *TraumaTracker*, a livello implementativo, i due observer sono entrambi un riferimento a *SpeechBoard*, dato che esso incapsula totalmente la gestione dello speech recognizer. Dal lato della libreria *SpeechRecognizerLib*, l'uso di due observer è dovuta ad una scelta progettuale, nella quale si voleva avere l'astrazione di due canali: uno per inviare i messaggi che devono essere elaborati dal resto del sistema(uiHandler) e uno su cui mandare solo messaggi informativi all'utente(dashBoardHandler). *SpeechRecognizerManager* oltre a permettere di essere avviato e fermato, permette anche di cambiare il suo stato relativo a quali termini possono essere riconosciuti o no [fig:3.1]. Ciò rende possibile l'integrazione dell'input vocale con altre forme di input.

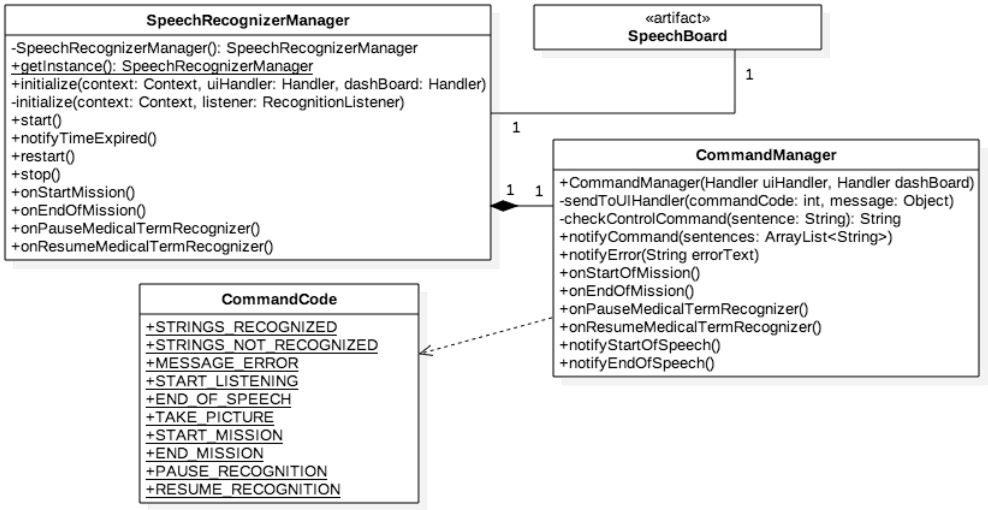


Figura 3.5: Associazione tra la libreria *SpeechRecognizerLib* e l'artefatto *SpeechBoard*

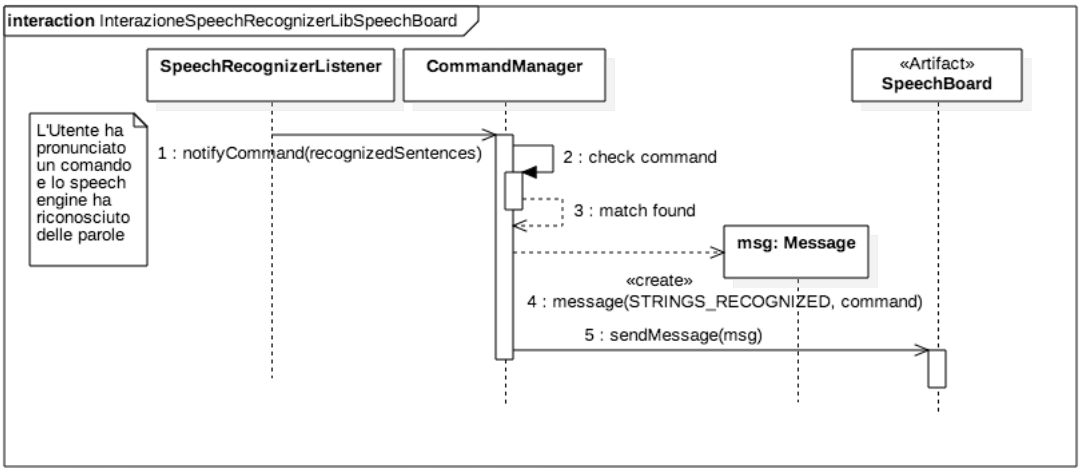


Figura 3.6: Interazione tra la libreria e l'artefatto *SpeechBoard* nel caso di comando riconosciuto

La libreria, per notificare dell'avvenimento dei vari eventi gli observer *uiHandeler* e *dashBoardHandler*, utilizza un approccio a scambio di messaggi. I tipi di messaggi scambiati possono essere visti in figura 3.5 alla classe *CommandCode*. A seconda del tipo di evento scatenato, la classe *CommandManager* creerà il messaggio opportuno e poi lo invierà all'artefatto *SpeechBoard*. Nella

figura 3.6 è possibile avere un esempio di interazione, tra la libreria e il resto del sistema, se un termine viene riconosciuto(`STRING_RECOGNIZED`).

In questo caso il messaggio di risposta sarà di tipo `STRINGS_RECOGNIZED` e come contenuto avrà il comando riconosciuto.
In relazione a quanto detto prima, riguardo al riavvio del recognizer ciclicamente, tra lo stop e il riavvio vi è un piccolo intervallo di tempo, in cui il sistema non è in ascolto di ciò che può pronunciare l'utente. Questo intervallo deve essere notificato all'utilizzatore e per farlo si ricorre all'invio di due messaggi al *dashBoardHandler*. Il messaggio di tipo `END_OF_SPEECH` riguarda il momento in cui il sistema smette di ascoltare, mentre `START_LISTENING` il momento in cui il sistema inizia l'ascolto.

3.5 Riconoscimento comandi

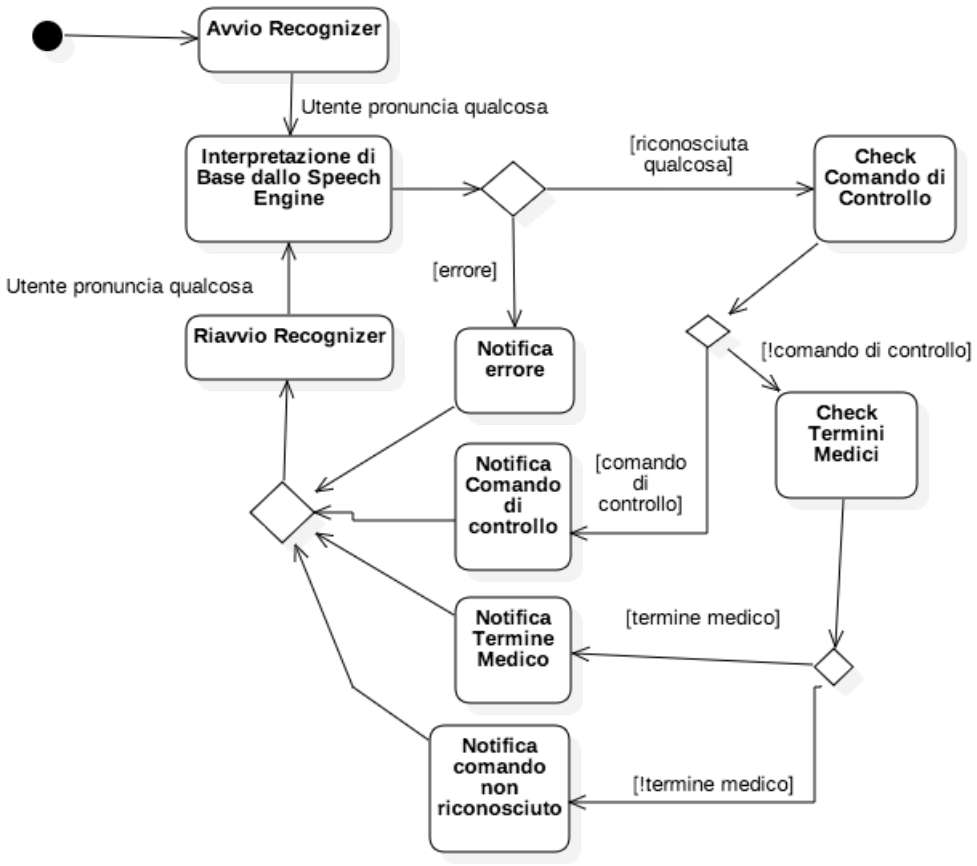


Figura 3.7: Ciclo base di riconoscimento

Ora che è stato introdotto come la libreria si interfaccia con Android e con il resto del sistema *TraumaTracker*, si passi ad analizzare come dalle parole riconosciute dallo speech engine, si riesca ad associare i comandi definiti nella sezione di analisi (tabella: 3.1 + comandi di controllo). All'interno di *CommandManager* si trova la gestione dell'interpretazione dei comandi. Come mostrato in figura 3.7, quando l'utente pronuncia qualcosa lo speech engine cerca di interpretarlo in qualche modo, il risultato dell'elaborazione viene poi passato dallo *SpeechRecognizerListener* al *CommandManager*. In relazione a quanto detto in fase di analisi, i comandi di controllo hanno una priorità più elevata nella fase di checking dei termini. Per fare ciò il *CommandManager*, appena riceve il risultato dell'elaborazione, se equivale ad un errore lo notifica allo *SpeechBoard*. Altrimenti, se corrisponde ad una parola interpretata, come prima cosa effettuerà un check per vedere se equivale ad un comando di controllo. In caso negativo controllerà se corrisponde ad un termine medico. All'interno di *CommandManager* è incapsulata anche la gestione dello stato del recognizer relativa alla figura 3.1. Infatti, nel caso in cui un intervento non sia attivo, *CommandManager* si limiterà a controllare solamente il comando `Inizia intervento`. A intervento attivo invece, eseguirà il check su tutti i comandi di controllo e, a seconda che l'utente abbia o meno messo in pausa il riconoscimento dei termini medici, controllerà anch'essi. Lo stato del recognizer, oltre che essere modificato direttamente in funzione dei comandi di controllo riconosciuti, è possibile cambiarlo attraverso le funzioni messe a disposizione *SpeechRecognizerManager*: onStartMission, onEndOfMission, onPauseMedicalTermRecognizer, onResumeMedicalTermRecognizer.

3.5.1 Gestione comandi di controllo

Per quanto riguarda il riconoscimento dei comandi di controllo, essendo parole comuni della lingua italiana, vengono riconosciute facilmente dallo speech engine. Quindi basterà, in questo caso, ricorrere a semplici confronti fra stringhe. Infatti il *CommandManager*, in relazione allo stato in cui si trova, confronterà la stringa passata dallo *SpeechRecognizerListener* con i termini contenuti in *ControlCommand* [fig:3.8]. Nel caso in cui la parola riconosciuta corrisponde ad un comando controllo, il *CommandManager* lo notificherà allo *SpeechBoard* con un opportuno messaggio e nel caso comporti una transazione di stato per la libreria, modificherà il suo stato interno

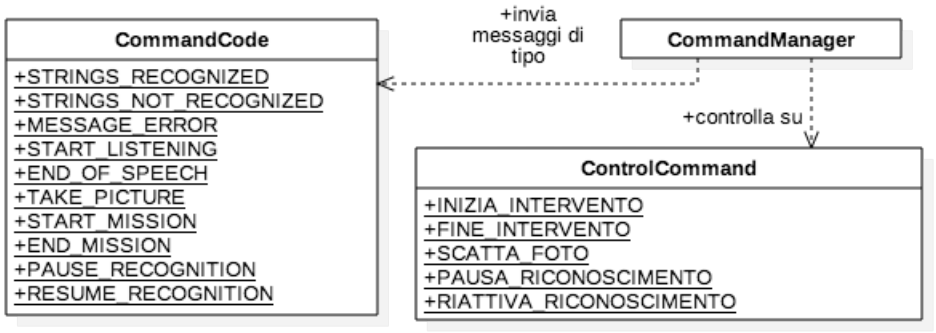


Figura 3.8: Classi coinvolte nel check dei comandi di controllo

Di seguito in figura 3.9 viene riportato un esempio di interazione fra le varie componenti quando l'utente pronuncia il comando `inizia intervento`. All'istruzione *4* vengono modificate le proprietà interne che tengono traccia dell'inizio dell'intervento. Successivamente il *CommandManager* notifica, con un messaggio di tipo `START_MISSION`, l'inizio dell'intervento anche allo *SpeechBoard*. Per completezza dato che ha riconosciuto un termine valido invierà anche un messaggio di tipo `STRINGS_RECOGNIZED`

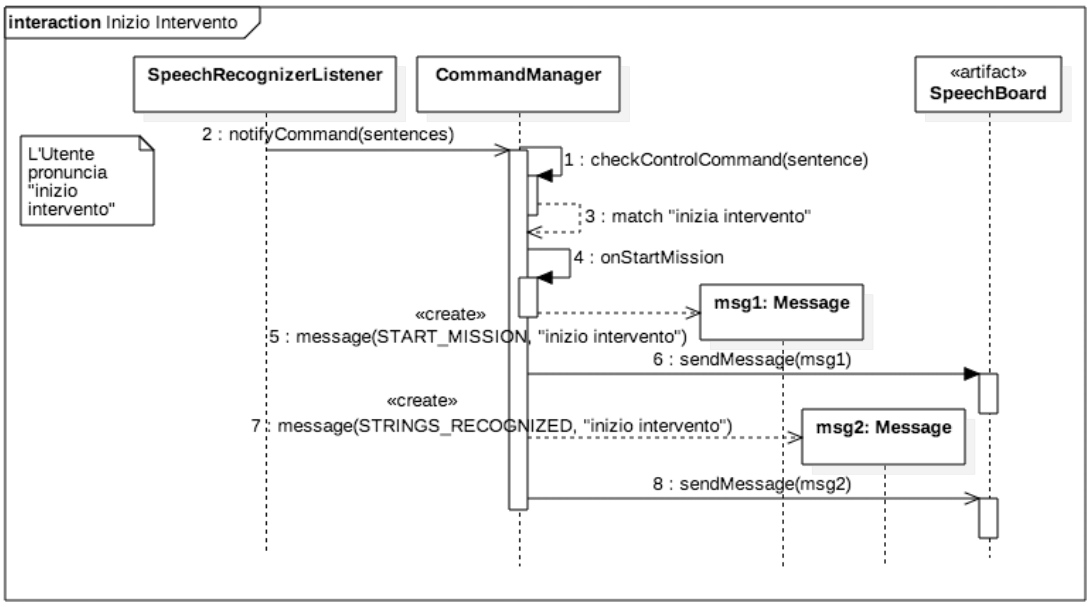


Figura 3.9: Sequenza di operazioni che avvengono quando l'utente pronuncia `inizio intervento`

3.5.2 Riconoscimento termini medici

Se la modalità intervento è attiva, il riconoscimento dei termini medici abilitato e la stringa passata dallo SpeechRecognizerListener non corrisponde ad un comando di controllo, il *CommandManager* passerà a fare il check dei termini medici. Per eseguirlo ricorrerà alla classe *MedicalSpeechInterpreter*.

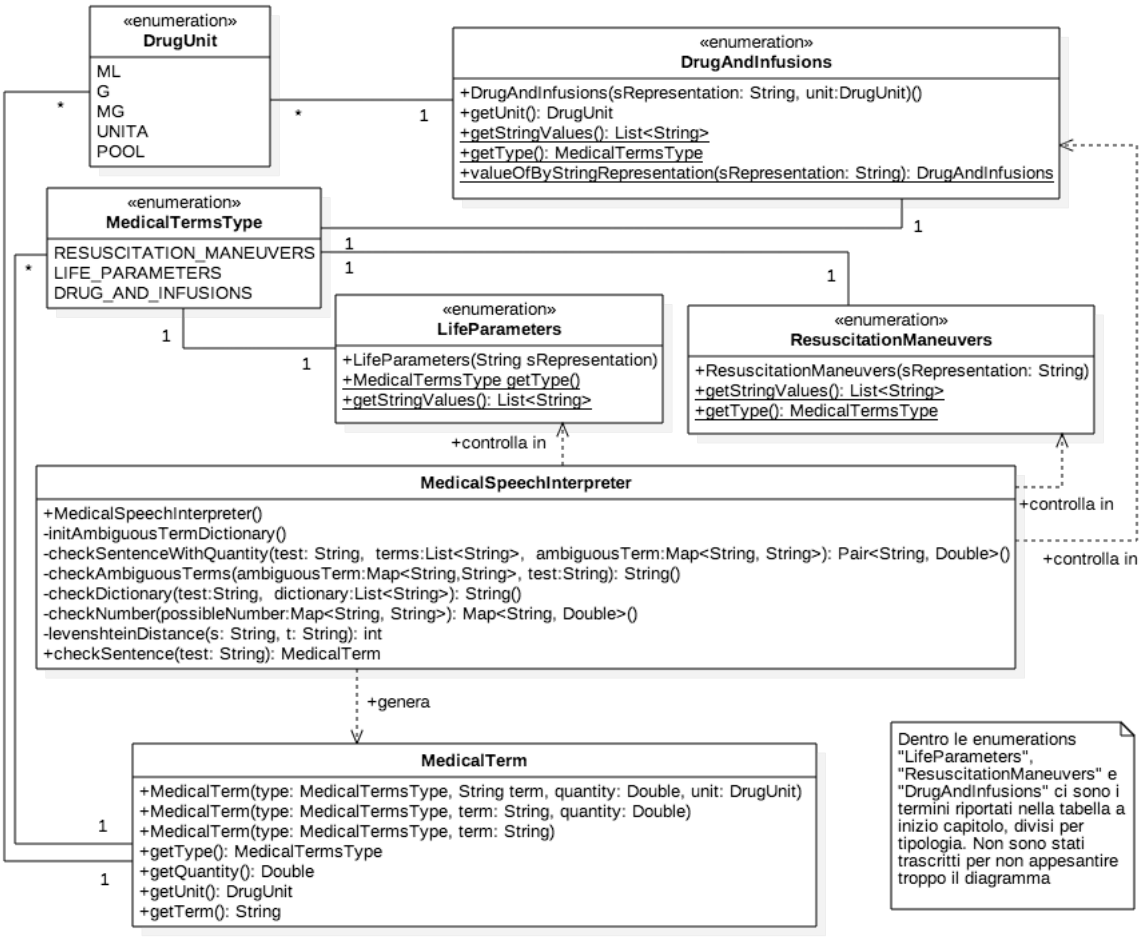


Figura 3.10: Architettura interprete termini medici

Dal diagramma in figura 3.10 si può vedere come è stata progettata a livello di classi, l'ontologia dei termini medici. *LifeParameters*, *ResuscitationManeuvers* e *DrugAndInfusions* sono enumeration che contengono i vari termini forniti dal *Trauma Team*. Essi sono divisi per tipologie, definite dai valori della enumeration *MedicalTermsType*. Dato che i farmaci e le infusioni hanno delle unità di misura fisse, per gestirle è stato scelto di associare ad ogni termine

di *DrugAndInfusions* un valore della enumerazione *DrugUnit*. Ciò permette al sistema di associare automaticamente l'unità di misura al medicinale, quando il *Trauma Leader* ne pronuncia il nome e la quantità. Le enumeration descritte vengono sfruttate dal *MedicalSpeechInterpreter*, per controllare se il termine fornito dallo speech engine è associabile ad un termine medico. Una volta che ha eseguito l'elaborazione, il *MedicalSpeechInterpreter*, se ha trovato una corrispondenza, genererà un'istanza della classe *MedicalTerm* con tutti i dati relativi al termine riconosciuto e lo passerà al *CommandManager*, che si occuperà di creare il messaggio di risposta da inviare alla *SpeechBoard*. In relazione a quanto detto in fase di analisi, per quanto riguarda i parametri vitali, il recognizer restituirà un risultato anche se non verrà pronunciata la quantità. Per i medicinali invece, verrà restituito un risultato solo se vi sarà anche la quantità associata.

Associazione termini medici con termini simili

In questo caso a differenza dei comandi di controllo, non tutti i termini sono presenti all'interno del vocabolario su cui si basa l'engine di riconoscimento. Come detto precedentemente, alcuni termini medici vengono riconosciuti come parole simili. È stato posto quindi il problema di come associare il risultato fornito dallo speech engine ai termini medici da riconoscere. Ragionando sulla similitudine, si è notato che la differenza, tra i comandi e le interpretazioni fornite dallo speech recognizer, consisteva in alcuni caratteri che componevano le due parole. Considerando ciò, si è pensato di ricorrere ad un algoritmo che permettesse di calcolare la distanza fra due stringhe. A tale scopo è stato scelto di utilizzare l'algoritmo per il calcolo della *Distanza di Levenshtein*. Tale distanza corrisponde al numero di caratteri che devono essere inseriti,eliminati o sostituiti, per trasformare una stringa in un'altra. Più le due parole sono simili, minore sarà la distanza tra esse. In relazione a quanto detto, confrontando il risultato dello speech engine con i termini da riconoscere, occorre trovare il termine con la minor distanza di Levenshtein. Ciò però non basta. Nel caso l'utente pronunci una parola che non sia un comando per il sistema, essa verrebbe comunque rilevata da *TraumaTracker* e associata al comando con la distanza minore. Per ovviare al problema, considerando che il valore massimo della distanza di Levenshtein è la lunghezza della stringa più lunga e il valore minimo è 0, è stato inserito un controllo, che permette di associare un termine medico a ciò che restituisce l'engine di riconoscimento, solo se la percentuale di somiglianza della distanza sia maggiore di un valore di accuratezza prestabilito.

Si definisca: `ld` come la distanza di Levenshtein tra due stringhe A e B, `lMax` come la lunghezza della stringa più lunga tra A e B. Si definisca poi `V`

come la percentuale di somiglianza fra le due stringhe. Quindi avremo che:

$$V = \frac{LSM - ld}{LSM * 100}$$

dove un valore di V = 100% significa che le due stringhe sono uguali mentre V = 0% significa che le due stringhe non hanno nessun carattere in comune. Per riuscire poi ad ottenere il primo termine con la percentuale di somiglianza più elevata, in fase di check, è stato effettuato un controllo dei termini più volte, a percentuali differenti [listato:3.1].

```
private String checkDictionary(String test, List<String> dictionary){
   double accuracy = 86;
   String max, min;
   for(; accuracy >= 54; accuracy -= 2) {
      for(String cmd : dictionary){
         if(cmd.length() >= test.length()){
            max = cmd;
            min = test;
         } else {
            max = test;
            min = cmd;
         }
         double ld = levenshteinDistance(min, max);
         double v = (max.length() - ld)/max.length()*100;
         if(v >= accuracy){
            return cmd;
         }
      }
   }
   return null;
}
```

Listato 3.1: Codice per trovare il termine più somigliante a quello restituito dallo speech engine

I valori di accuratezza sono stati scelti a seguito di molteplici prove. Da queste ultime è anche emerso che il minimo valore di accuratezza stabilito non garantiva l'associazione di alcuni termini, purtroppo l'abbassamento ulteriore del valore portava alla generazione di falsi risultati. A seguito di alcune osservazioni, si è notato che per i termini non associati, lo speech engine forniva sempre le stesse interpretazioni. Per risolvere il problema si è ricorso quindi ad una HashMap in cui sono state memorizzate le associazioni tra i risultati forniti sempre dallo SpeechRecognizer e il valore del relativo termine medico.

In fase di controllo il *MedicalSpeechInterpreter* controllerà prima la HashMap poi ricorrerà all'algoritmo nel listato 3.1.

Riconoscimento quantità associato ai termini

Per quanto riguarda il riconoscimento delle quantità associate ad un termine, il metodo descritto sopra non basta. Quindi, per i parametri vitali e i medicinali, è necessario adoperare un approccio leggermente differente. Analizzando ciò che restituisce lo speech engine, si può notare che tra una parola ed un numero vi è uno spazio. Sfruttando questa caratteristica si è pensato di spezzare la stringa ad ogni spazio. Ciclicamente sarà poi ricomposta un pezzo ad ogni iterazione e verrà effettuato il confronto con i termini medici, come descritto precedentemente. Ogni volta che si troverà una corrispondenza sarà memorizzata una coppia di valori composta dal termine medico e il resto della frase non ancora ricomposto [fig:3.11].

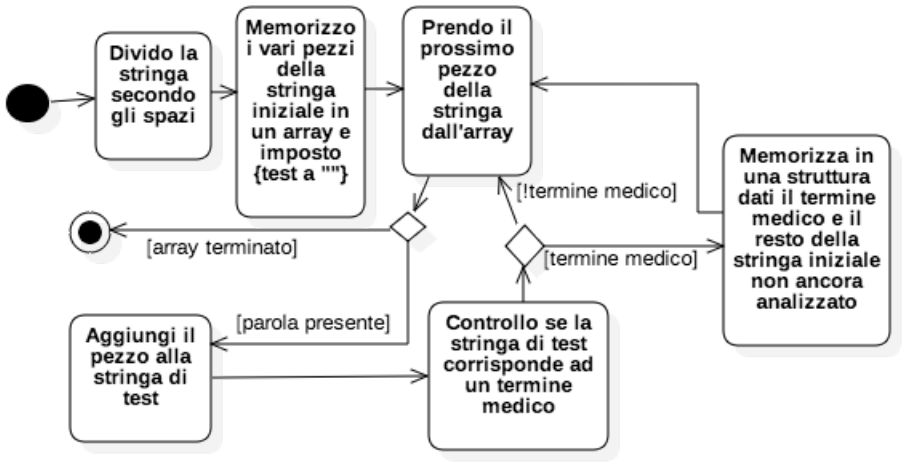


Figura 3.11: Attività eseguite per trovare il termine medico che potrebbe avere una quantità associata

Alla fine di tutte le iterazioni si avrà una struttura dati contente un set di termini medici con associati i possibili valori numerici. Si passa ora ad analizzare i possibili valori numerici. Come dichiarato precedentemente, può capitare che alcune volte l'engine interpreti alcuni numeri in lettere invece che in cifre, sarà quindi necessario, prima di tutto, effettuare un filtraggio per trasformarli in cifre. Successivamente vengono pulite ulteriori imperfezioni relativamente all'interpretazione delle quantità decimali. Infine si cercherà di tradurre il re-

sto delle parole della frase da stringhe a valori numerici, sommandoli a mano a mano [listato:3.2]

```
private Map<String, Double> checkNumber(Map<String, String>
    possibleNumber){
   Map<String, Double> res = new HashMap<String, Double>();
   boolean isPresentNumericValue = false;

   /* Number that the speech recognizer gives in letters*/
   Map<String, Double> NUMBER = new HashMap<String, Double>();
   NUMBER.put("mille", 1000.0);
   NUMBER.put("sei", 6.0);
   NUMBER.put("uno", 1.0);
   NUMBER.put("due", 2.0);
   NUMBER.put("sette", 7.0);
   NUMBER.put("nove", 9.0);
   NUMBER.put("dieci", 10.0);
   NUMBER.put("centomila", 100000.0);
   NUMBER.put("mezzo", 0.50);

   for(String tmp : possibleNumber.keySet()){
      String posNum = possibleNumber.get(tmp);

      double converted = 0;
      posNum = posNum.replace(',', '.');
      //clean imperfection
      if(posNum.contains(":30")){
         posNum = posNum.replace(":30", ".50");
      }
      if(posNum.contains(":00")){
         posNum = posNum.replace(":00", "");
      }
      if(posNum.contains(" . ")){
         posNum = posNum.replace(" . ", ".");
      } else if(posNum.contains(". ")){
         posNum = posNum.replace(". ", ".");
      } else if(posNum.contains(" punto ")){
         posNum = posNum.replace(" punto ", ".");
      }

      String[] sSplitted = posNum.split(" ");
      for(String s : sSplitted){
         s = s.toLowerCase(Locale.getDefault());
         try{
```

```
            converted += NUMBER.containsKey(s) ? NUMBER.get(s) :
                Double.parseDouble(s);
            isPresentNumericValue = true;
         } catch (NumberFormatException e){
            e.printStackTrace();
         }
      }

      if(isPresentNumericValue){
         res.put(tmp, converted);
      } else {
         res.put(tmp, null);
      }
   }
   return res;
}
```

Listato 3.2: Codice che permette di interpretare i possibili valori numerici associati ad un comando

3.5.3 Nota finale Speech recognizer

Al termine della descrizione del funzionamento dello speech recognizer, ci si potrebbe chiedere come può fare l'utente per annullare un comando impartito. Riflettendo sullo scenario di utilizzo del sistema, l'aggiunta della possibilità di annullamento di un comando può portare a rallentamenti nel lavoro dell'utente. Questo perché avrebbe richiesto al *Trauma Leader* di rendersi conto del comando sbagliato, impartire il comando di annullamento e poi di ripronunciare il comando corretto. Parlando anche con il *Trauma Team*, è risultato più conveniente per il medico ripronunciare solo il comando corretto quando si rende conto di quello errato, l'eliminazione di quest'ultimo sarà fatta manualmente in fase di controllo del report finale.

Capitolo 4

Il Sottosistema Glass Interface

Dopo aver parlato dello speech recognizer, si passa ora a presentare la progettazione di una prima interfaccia grafica per il sistema *TraumaTracker*. Si premette fin da subito che l'obiettivo non è quello di creare una *GUI* esteticamente bella, ma di progettarne una il più possibile funzionale.

4.1 Analisi problematiche

La prima considerazione da fare è che occorre ideare un'interfaccia che permetta all'utente di vedere velocemente e facilmente, le varie informazioni relative a *TraumaTracker*, in modo che l'uso del sistema non costituisca un intralcio al lavoro del *Trauma Leader*. In relazione al caso di studio presentato nel secondo capitolo e ad alcune considerazioni fatte nel terzo, l'interfaccia da sviluppare dovrà:

- informare l'utente se lo speech recognizer è in ascolto o no
- mostrare quando la modalità intervento è attiva o inattiva
- visualizzare i comandi riconosciuti
- informare se il riconoscimento dei termini medici è in pausa
- mostrare il valore dei parametri vitali
- visualizzare quando il sistema sta acquisendo una foto

L'interfaccia sarà gestita dall'applicazione *TraumaTrackerGlass* in esecuzione sugli smartglass. Risulta necessario focalizzare le caratteristiche del dispositivo per il quale sarà sviluppata. Nel caso di *TraumaTracker* ci si trova

ad utilizzare un paio di smartglass non see-through, in cui il display non occupa tutto il campo visivo dell'utente, ma è posto a lato di un occhio. Dalle indicazioni date dal costruttore, nel caso dei dispositivi Vuzix M-100/M-300, si ha una percezione dello schermo come quella di uno smartphone da 4/5 pollici ad una distanza di 14 pollici, con un aspect ratio di 16:9.

Di seguito saranno riportate le varie soluzioni progettuali accompagnate da piccoli mockup, che permettano di cogliere meglio l'idea.

4.2 Soluzioni progettuali

Seguendo l'ordine dell'elenco mostrato nella sezione precedente, si va ora a illustrare le scelte progettuali per ogni punto. Prima di iniziare occorre fare una considerazione generale. Per garantire una buona leggibilità con poco sforzo è opportuno scrivere tutti i testi in maiuscolo, utilizzando un font abbastanza grande e leggibile. Oltre ad accorgimenti riguardo al testo. Per facilitare la comprensione. Dovranno essere usati opportuni colori, cercando anche di prendere spunto da altri dispositivi usati solitamente dai medici durante il loro lavoro. Per la disposizione dei vari elementi, si immagini il display diviso in regioni [fig:4.1].

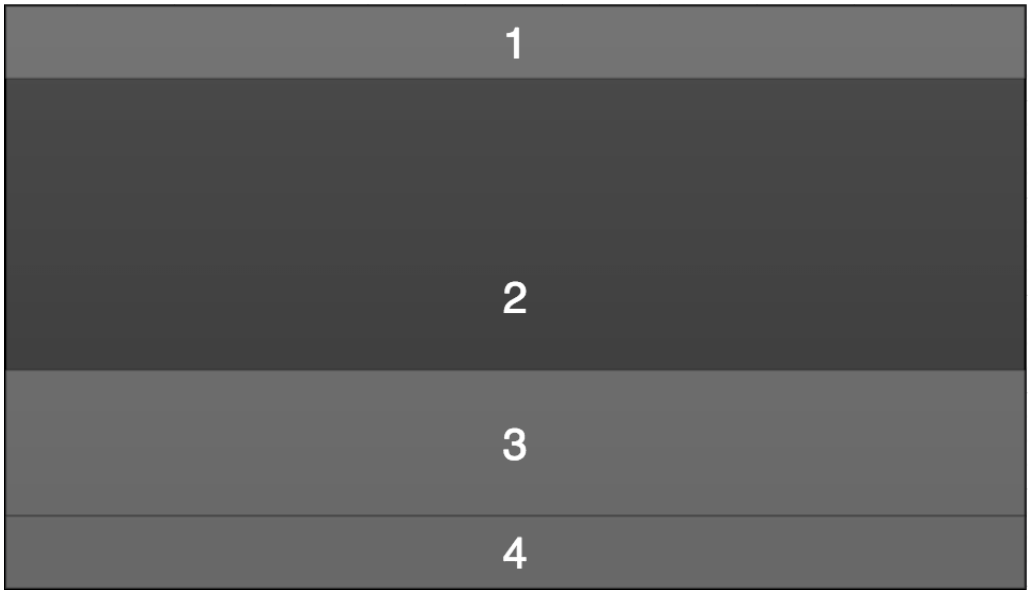


Figura 4.1: Macro divisione dello schermo

L'altezza delle regioni rispetto a quella del display del dispositivo saranno:

- per la 1 e la 4: $\frac{1}{8}$

- per la 2: $\frac{4}{8}$.
- per la 3: $\frac{2}{8}$

Il colore di base dello sfondo sarà nero.

4.2.1 Visualizzazione dell'ascolto da parte dello Speech Recognizer

Per quanto riguarda la visualizzazione dell'ascolto da parte dello Speech Recognizer. Dato che deve informare l'utente dell'istante in cui può parlare oppure no, occorre trovare una soluzione che sia immediatamente visibile anche con "la coda dell'occhio". A tal proposito si è pensato di posizionare una barra ai piedi dello schermo (regione 4), che quando lo speech recognizer è in ascolto, si illumini di verde con scritto *LISTENING* in nero al centro e si spenga nel momento in cui termina l'ascolto [fig:4.2].

Figura 4.2: Ascolto da parte dello Speech Recognizer attivo

4.2.2 Visualizzazione modalità intervento

Per la visualizzazione dello stato dell'intervento si è optato per mostrare la parola `ACTIVE` a destra della regione 1, quando la modalità intervento è attiva. Tale scritta sarà inserita all'interno di un riquadro blu.

Figura 4.3: Visualizzazione intervento attivo

4.2.3 Visualizzazione dei comandi riconosciuti

La regione 3 si è pensato di dedicarla alla visualizzazione dei comandi riconosciuti. In questo caso per i termini medici verrà mostrata la sigla della tipologia e il nome. Mentre per i comandi di controllo verrà visualizzato solo il nome. Le sigle dei termini medici saranno: *MAN* per le manovre rianimatorie, *PAR* per i parametri vitali e *DRUG* per i medicinali. Nella figura sottostante è mostrato l'esempio relativo alla manovra *Drenaggio Toracico*.

Figura 4.4: Visualizzazione del comando riconosciuto

4.2.4 Mostrare quando i termini medici sono in pausa

Sempre nella regione 1 viene mostrato se il riconoscimento dei termini medici è in pausa. Per fare ciò, quando il sistema è in pausa, si è scelto di mostrare un riquadro giallo, con una **P** al centro, alla sinistra della regione 1.

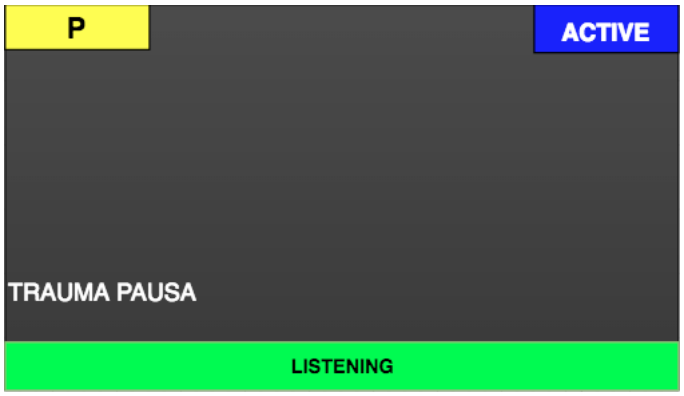


Figura 4.5: Visualizzazione dello stato di pausa del riconoscimento dei termini medici

4.2.5 Visualizzazione parametri vitali

La regione 2 invece è dedicata alla visualizzazione dei parametri vitali. Per tale scopo, si è scelto di creare una grafica che assomigliasse ai monitor utilizzati dai medici. Su di essi solitamente vengono visualizzati grafici e dati numerici relativi a vari parametri, divisi in riquadri di colori diversi. Per questioni di spazio sul dispositivo, consultandoci anche con il *Trauma Team*, si è scelto di mostrare solo i dati numerici per ogni parametro. Ciò però, verrà fatto rispettando la divisione in riquadri e la colorazione. Questo è stato fatto con l'obiettivo di dare la sensazione ai medici di utilizzare già qualcosa a loro familiare.

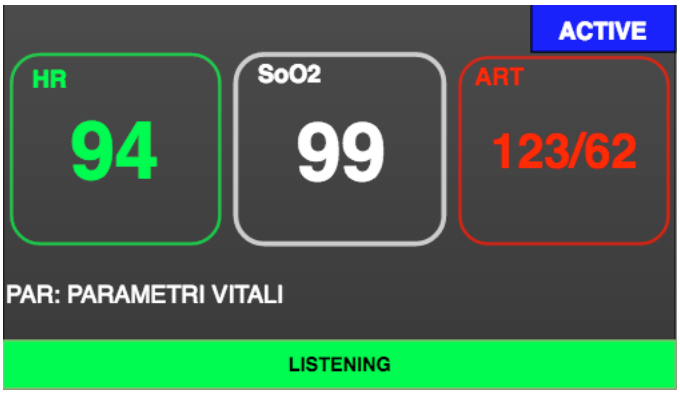


Figura 4.6: Esempio di visualizzazione dei parametri vitali predefiniti

4.2.6 Visualizzare l'acquisizione della foto

Quando l'utente richiede l'acquisizione di una foto, occorre mostrare il momento in cui il sistema la sta scattando. In modo tale che l'utente fissi il punto a cui vuole fare la foto, per permettere al sistema di mettere a fuoco e acquisire l'immagine. Per fare ciò, si è scelto di mostrare l'icona di una fotocamera al centro della regione 1.

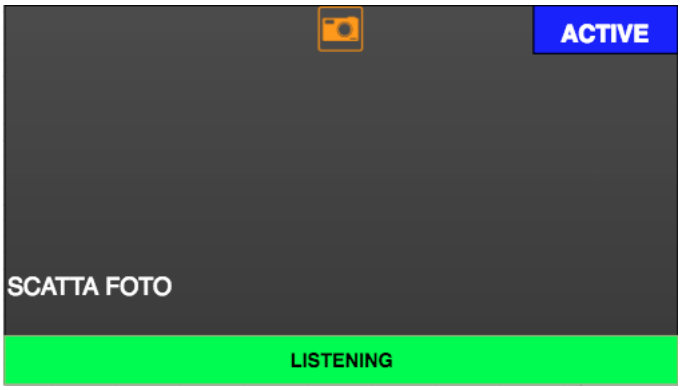


Figura 4.7: Istante in cui il sistema sta acquisendo una foto

Capitolo 5

Test e Valutazioni

In questo capitolo finale presentiamo i test effettuati mettendo insieme ciò che è stato sviluppato finora, a livello di *TraumaTrackerCore* e *TraumaTrackerGlass*. Si premette che allo stato attuale a livello di interfaccia grafica, rispetto alla progettazione presentata precedentemente, è stato implementata solo: la barra di notifica dell'ascolto da parte dello SpeechRecognizer, la visualizzazione del comando impartito, la visualizzazione dello stato dell'intervento e la notifica dell'acquisizione della foto. L'obiettivo di questi test è quello di capire se le potenzialità del prototipo sviluppato finora, possono già soddisfare alcuni requisiti posti all'inizio, sopratutto sotto il punto di vista dell'usabilità. Un secondo obiettivo è, in caso di problemi, capire se la causa principale è interna al sistema oppure dovuta ad un errato utilizzo da parte dell'utente.

5.1 Scenario di test

Per questi test sono state prese in esame 3 persone differenti, a cui è stato fatta pronunciare la stessa sequenza di operazioni più volte. Introducendo ad ogni test qualche caratteristica in più sul sistema **TraumaTracker**, che li potesse orientare verso il corretto utilizzo. Come dispositivi sono stati utilizzati: un paio di smartglass Vuzix M-100 su cui è stato eseguito *TraumaTrackerGlass*, un Nexus 7 su cui è stato eseguito *TraumaTrackerCore* ed un comune paio di auricolari per smartphone dotati di microfono, collegati al Nexus 7. Ad ogni persona è stato fornito un set di 14 comandi da pronunciare, presi dall'elenco definito nel capitolo 3. Il set di comandi utilizzato è:

1. Inizia intervento
2. Intubazione Orotracheale
3. Scatta foto

4. Drenaggio Toracico
5. Parametri vitali
6. Sutura ferita emorragica
7. Ketamina 10
8. Mannitolo 30
9. Crioprecipitati 15
10. Infusore a pressione
11. Morfina 30
12. Succinilcolica 25
13. Catere alto flusso
14. Fine intervento

L'intera sequenza di comandi è stato fatta ripetere per tre volte ad ogni utente, per un totale di circa 10 minuti a persona non considerando i tempi di pausa tra un test e l'altro. La prima volta è stato fornito il sistema, dicendo agli utenti di pronunciare l'elenco di comandi, senza fornire dettagli su come andassero pronunciati. Al secondo giro di test è stato spiegato il funzionamento del sistema e come pronunciare correttamente i comandi. Per la precisione è stato detto ai tester che, per un corretto uso, i comandi andavano pronunciati come se dovessero impartire un ordine ad un'altra persona. Mentre per quanto riguarda il funzionamento del sistema, è stato spiegato agli utenti che l'ascolto dello speech recognizer avviene in una finestra temporale, la quale comincia ogni volta che la barra in basso della notifica si illumina di verde e termina qualche secondo dopo che l'utente smette di parlare. Quindi tra un comando e l'altro, vi è la necessità di fare delle pause. Oltre alla barra, ad ogni ciclo, quando lo speech recognizer comincia l'ascolto, viene emesso automaticamente dal sistema un suono udibile, attraverso gli auricolari. Tale feature è stata spiegata ai tester ed è stato fatto ripetere il test per la terza volta, facendogli utilizzare anche gli auricolari.

Durante i test, agli utenti, oltre a pronunciare l'elenco di comandi, dovevano dire ciò che appariva sullo schermo degli smartglass.

5.2 Risultati

Dall'osservazione delle varie prove, si è potuto notare che: al primo giro gli utenti hanno avuto una leggera difficoltà nell'uso del sistema. Infatti vi è stata la necessità di ripetere diversi comandi, sopratutto i primi. Già a metà del primo ciclo di test, si è visto un leggero incremento delle prestazioni, in termini di comandi riconosciuti e lettura della risposta sugli occhiali. Al secondo giro, la spiegazione di come pronunciare correttamente i comandi ha portato ad avere un incremento delle prestazioni, pur continuando ad avere dei piccoli problemi sul riconoscimento di alcuni comandi. Tali termini sono stati fatti poi ripetere, correggendo la pronuncia dell'utente. Ciò ha portato successivamente ad una corretto riconoscimento da parte del sistema. Alla terza iterazione del test, con l'introduzione degli auricolari, per due utenti le prestazioni sono rimaste pressoché invariate, mentre per un terzo sono aumentate ulteriormente.

Chiedendo pareri agli utenti riguardo all'esperienza di utilizzo, essi hanno riferito che: appena si prende dimestichezza con il sistema, risulta facile l'uso, soprattutto il tempo richiesto per familiarizzare con gli occhiali è abbastanza breve. L'uso degli smartglass permette, con poco sforzo, di avere quasi immediatamente informazioni riguardo al sistema, in modo molto simile ad uno smartphone, con in più la possibilità di avere le mani libere per eseguire altre attività. È stato anche chiesto agli utenti se, secondo loro, l'occhiale era una fonte di distrazione. In risposta, hanno detto che non costituivano un grosso fastidio. L'unica pecca è che il dispositivo risulta abbastanza pesante e muovendo il capo, tende a muoversi anche l'occhiale, costringendo l'utente a riposizionare il device. Altra nota positiva sollevata dagli utenti, è che l'uso della barra di notifica colorata risulta particolarmente utile per capire, quasi istantaneamente, quando il sistema inizia ad ascoltare. L'utente, che ha avuto giovamento dall'uso dell'auricolare, ha riferito che quest'ultimo semplifica ulteriormente l'utilizzo del sistema. Questo perché gli risultava più facile reagire al suono emesso dalle cuffie, piuttosto che la barra di notifica sugli occhiali. Viceversa gli altri due utenti hanno riferito che l'auricolare non cambiava la loro esperienza di utilizzo, perché per loro la notifica sull'occhiale risultava più immediata. A prescindere da ciò l'uso dell'auricolare, per loro, non costituiva un disturbo.

5.2.1 Valutazioni

In relazione a quello che è emerso durante i test, riguardo agli errori del riconoscimento vocale, la maggior parte delle volte erano causati da una errato modo di parlare da parte dell'utente. È quindi possibile affermare che una corretta pronuncia risulta necessaria per ottenere buone prestazioni con

il sistema. L'apprendimento di ciò non risulta particolarmente difficile. Sarà quindi richiesto solo un breve periodo di training, in modo da permettere all'utente di familiarizzare con l'intero sistema. Per quanto riguarda gli occhiali anche su consiglio dei tester, vi è la necessita di aggiungere qualche accessorio, come ad esempio una cordoncino per occhiali, che permetta di mantenere più saldi gli occhiali al capo dell'utente. I commenti all'uso di colori che aumentino il significato di alcune notifiche, ha permesso di capire che le idee avute in fase di progettazione dell'interfaccia sono corrette ed è possibile continuare a seguirle per l'implementazione. Considerando che per un utente l'aggiunta della notifica audio ha portato giovamento, mentre per gli altri non costituisce un problema, si è ritenuto opportuno continuare ad utilizzare anch'essa.

Tutto sommato, anche in virtù dei test fatti e considerando l'errore intrinseco dello speech engine di base, ci si può ritenere abbastanza soddisfatti dei risultati raggiunti fino a questo momento. Naturalmente essendo ancora un progetto in via di sviluppo, non tutti i requisiti posti inizialmente sono stati soddisfatti appieno. Per quanto riguarda la parte sviluppata personalmente da me, ancora è possibile introdurre qualche miglioramento, che permetta di aumentare ancora di più le prestazioni dell'intero sistema.

Conclusioni

Analizzando quindi il mondo del wearable computing, in particolare l'ambito medico ospedaliero, si può ben intuire che i dispositivi indossabili introducono notevoli miglioramenti, sia dal punto di vista dell'acquisizione di informazioni, che in un abbattimento ingente del tempo necessario per la loro fruizione al servizio dei medici. Grazie alla possibilità di essere utilizzati senza impegnare le mani, risultano particolarmente adatti anche per un uso in fase operativa, fornendo al medico la possibilità di accedere ad informazioni importantissime, proprio nell'istante in cui ne ha bisogno.

Lo sviluppo di tali sistemi però, pur provenendo dal mondo dei sistemi embedded, richiede di rivedere un attimo i paradigmi utilizzati fino ad oggi. Ora occorre, fin da inizio sviluppo, porre l'utente al centro e costruire il sistema intorno ad esso, in funzione dei suoi limiti ed esigenze. Ciò porta generalmente a creare applicazioni specific purpose, in cui i sistemi e le tecnologie disponibili vengono concretamente adattate all'utente. Quindi, per quanto riguarda la progettazione e l'implementazione di tali sistemi, si pongono nuove sfide sia a livello hardware che software. Questo conduce anche all'introduzioni di nuovi metodi di interfacciamento con i sistemi, che permettono all'utente di usare i dispositivi in modo naturale e, in alcuni casi, di interagire con essi nello stesso modo in cui avviene l'iterazione fra persone; ad esempio l'uso della voce oppure attraverso appositi sensori che permettano ai device di rilevare lo stato psicofisico dell'utente. In relazione a quanto detto, un esempio concreto è costituito dal sistema *TraumaTracker*, il quale offre anche interessanti spunti di analisi di come, non solo i sistemi wearable in se possano portare giovamento, ma l'unione di quest'ultimi con altre infrastrutture informatiche, permetta di aprire la strada ad un utilizzo di essi ancora più proficuo. In *TraumaTracker* è possibile vedere un insieme di dispositivi diversi, che spaziano dal mondo dei sistemi embedded fino ai server, i quali cooperano creando un unico sistema reattivo, che permette all'utente di concentrarsi totalmente sul proprio lavoro, velocizzando il recupero di informazioni utili e sgravandolo da compiti di secondo piano, come ad esempio, la gestione del tracciamento delle attività effettuate durante un intervento. Pensando ai sistemi citati, non ci si può fermare solo su una visione puramente reattiva, ma con l'idea di un ospedale del

futuro, il cosiddetto *ospedale aumentato*, risulta facile pensare a sistemi come *TraumaTracker*, i quali oltre ad essere sistemi reattivi, possono facilmente diventare anche sistemi proattivi. Essi, non solo saranno in grado di reagire agli input generati dall'utente, ma grazie ad una elaborazione intelligente dei dati acquisiti, riusciranno a consigliare il medico nelle scelte da prendere. Ciò porta l'idea del supporto ad un livello superiore, diventando concretamente un assistente virtuale per il medico. Questo potrebbe essere implementato anche in *TraumaTracker* dove ad esempio il sistema, in funzione delle manovre o somministrazioni che ha fatto l'utente in un certo momento, potrebbe avvisare quando vi è la necessità di eseguire altre manovre. Oltre all'elaborazione in funzione delle azioni dell'utente, sfruttando i dati acquisiti dai vari devices per il monitoraggio dei pazienti, il sistema potrebbe notificare quando determinati valori scendono sotto una certa soglia e quindi mostrando in automatico un messaggio di allarme nel momento del bisogno. Oltre a ciò, l'uso di sistemi wearable per l'acquisizione di dati importanti relativi a specifici interventi, permette, in un ottica di Big Data, di ricorrere ad analisi ed elaborazioni raffinate, le quali, per un ospedale, rappresentano una preziosa miniera di informazioni, utilizzabile per migliorare processi e linee guida, finalizzate al trattamento dei pazienti e alla gestione delle varie attività ospedaliere.

Ringraziamenti

A conclusione di questo lavoro non mi resta che ringraziare tutti quelli che hanno contribuito alla sua creazione. In primis, desidero ringraziare la mia famiglia: Andrea, Manuela e Annalisa, i quali in tutti questi anni non mi hanno fatto mai mancare nulla, fornendomi un supporto costante, spronandomi a dare il meglio di me e incoraggiandomi nei momenti di difficoltà.

Non posso non ringraziare anche il Prof. Ricci per la grande opportunità offerta di lavorare su un progetto così interessante e socialmente utile, il quale mi ha permesso di mettere in pratica le conoscenze acquisite in questi anni di studio. Vorrei anche ringraziarlo per i preziosi insegnamenti che mi ha dato durante lo sviluppo, i quali hanno permesso di arricchire ulteriormente il mio percorso formativo. Successivamente desidero ringraziare l'Ing. Croatti per la grandissima disponibilità nel supportarmi in ogni dubbio o problema che ho avuto. Vorrei ringraziare anche la Prof. Prandi per la disponibilità e i preziosi consigli forniti.

Un ringraziamento va anche ai miei compagni di università, che mi hanno accompagnato durante il corso degli studi, offrendomi anche la possibilità di avere valide persone con le quali confrontarmi.

Infine vorrei ringraziare anche i medici del *trauma center* dell'ospedale M.Bufalini di Cesena, i quali hanno creduto in me e nel team di sviluppo per la creazione di *TraumaTracker*.

Bibliografia

[1] Woodrow Barfield *Fundamentals of Wearable Computers and Augmented Reality*, CRC Press, 2016.

[2] Chris Barber, David J. Hannif, Sandra I. Woolley *Contrasting paradigms for development of wearable computers*, IBM Systems Journal, 1999.

[3] Chris Barber , Theodoros N. Arvanitis, David Haniff, Robert Buckley, *A wearable Computer for Paramedics: studies in model-based, user-centred and industrial design*, University of Birmingham, 1999.

[4] Thad Starner, *The Challenges of Wearable Computing*, IEEE, 2001.

[5] Thad Starner, *Project Glass: An Extension of the Self*, IEEE, 2013.

[6] Francesca De Crescenzio, Massimiliano Fantini, Franco Persiani, Luigi Di Stefano, Pietro Azzari, Samuele Salti, *Augmented Reality for Aircraft Maintenance Training and Operations Support*, IEEE,2011.

[7] Pierpaolo Ercoli, *WEARABLE DEVICE E USER EXPERIENCE*, Tesi di laurea, Università di Bologna, 2015.

[8] Yun Zhou, Tao Xu, Bertrand David, Renè Chalon, *Innovative wearable interfaces: an exploratory analysys of paper-based interfaces with camera-glasses device unit*, Springer-Verlag London, 2013.

[9] P. Lukowicz, T. Kirstein, G.Troster, *Wearable Systems for Health Care Applications*, Schattauer GmbH,2004.

[10] Yao Meng, Hee-Cheol Kim *Wearable Systems and Applications for Healthcare*, IEEE, 2011.

[11] Marie Chan, Daniel Estève, Jean-Yves Fourniols, Christophe Escriba, Eric Campo, *Smart wearable systems: Current status and future challanges*, Elsevier, 2012.

[12] Andreas Bulling, Kai Kunze *Eyewear Computers for Human-Computer Interaction*, INTERACTIONS, 2016.

[13] Ian McGraw, Rohit Prabhavalkar, Raziel Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Hasim Sak, Alexander Gruenstein, Franc oise Beaufays, Carolina Parada *PERSONALIZED SPEECH RECOGNITION ON MOBILE DEVICES*, Google Inc, 2016.

[14] Luis Buera, Antonio Miguel, Oscar Saz, Alfonso Ortega, Eduardo Lleidaa *Feature Vector Normalization with Combined Standard and Throat Microphones for Robust ASR*, University of Zaragoza, 2008.

[15] Martin Graciarena, Horacio Franco, *Combining Standard and Throat Microphones for Robust Speech Recognition*, IEEE, 2003.

[16] Engin Erzin, *Improving Throat Microphone Speech Recognition by Joint Analysis of Throat and Acoustic Microphone Recordings*, IEEE, 2009.

[17] Tomas Dekens, Werner Verhelst, Francois Capma,Frédéric Beaugendre *IMPROVED SPEECH RECOGNITION IN NOISY ENVIRONMENTS BY USING A THROAT MICROPHONE FOR ACCURATE VOICING DETECTION*, EUSIPCO, 2010.

[18] Md Sahidullah, Rosa Gonzalez Hautamaki, Dennis Alexander Lehmann Thomsen, Tomi Kinnunen, Zheng-Hua Tan, Ville Hautamaki, Robert Parts, Martti Pitkanen, *Robust Speaker Recognition with Combined Use of Acoustic and Throat Microphone Speech*, Md Sahidullah, 2016.

[19] Stéphane Dupont, Christophe Ris, Damien Bachelart, *Combined use of close-talk and throat microphones for improved speech recognition under non-stationary background noise*, University of East Anglia, 2004.

[20] A. Shahina, B. Yegnanarayana, M. R. Kesheorey, *Throat Microphone Signal for Speaker Recognition*, Eighth International Conference on Spoken Language Processing. 2004

[21] Jennings, Nicholas R. , *An Agent-based Approach for Building Complex Software Systems*, ACM, 2001.

[22] Alessandro Ricci, Mirko Viroli, Andrea Omicini , *"Give Agents their Artifacts": The A&A Approach for Engineering Working Environments in MAS*, IFAAMAS, 2007.

[23] Andrea Santi, Marco Guidi, Alessandro Ricci, *JaCa-Android: An Agent-Based Platform for Building Smart Mobile Applications*, Springer-Verlag, 2011.

[24] Olivier Boissier, Rafael H. Bordini, Jomi F. Hubner, Alessandro Ricci, Andrea Santi, *Multi-agent oriented programming with JaCaMo*, Science of Computer Programming, 2013