UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Facoltà di Ingegneria

Corso di Laurea in INGEGNERIA INFORMATICA

Tesi di Laurea in RETI DI CALCOLATORI

Supporto alla Gestione di Gruppi per l'Assegnamento di Task di Crowdsensing

Candidato:

Leo Gioia

Relatore:

Chiar.mo Prof. Ing. Antonio Corradi

Correlatori:

Prof. Ing. Luca Foschini

Dott. Ing. Raffaele Ianniello

Anno Accademico 2014/2015

Sessione II

Indice Generale

Introduzione

Nell'ultimo decennio la diffusione di dispositivi mobili utilizzabili in qualsiasi momento e in qualsiasi luogo ha subito una crescita senza precedenti. Nello stesso periodo sono aumentati anche i sensori presenti al loro interno e attualmente nella maggior parte di essi sono presenti almeno GPS, microfono, videocamera, accelerometro, giroscopio e WiFi.

Tutto ciò permette di avere a disposizione un elevato numero di dispositivi capaci di acquisire ed elaborare informazioni su ciò che accade intorno ad essi e comunicarle ad altri dispositivi.

Queste nuove tecnologie forniscono uno strumento importantissimo nello scenario delle Smart City, ovvero quelle città che valutano le necessità dei cittadini attraverso l'ottenimento di feedback che vengono utilizzati per il bene di una comunità.

Il crowdsensing, ossia la tecnica di delegare la raccolta di informazioni agli stessi cittadini, permette di avere un risparmio non indifferente delle risorse rispetto ad una soluzione basata su un'infrastruttura fissa; di contro la scelta del crowdsensing indebolisce il controllo dei dispositivi per la raccolta dei dati in quanto non sono più monitorati da un'infrastruttura centrale.

Uno dei grandi problemi da risolvere è trovare una modalità di incentivazione al fornire dati spontaneamente da parte delle persone che ne vivono all'interno perché non tutti i cittadini sono dotati di un senso di appartenenza ad una comunità o senso civico tale da fornire i dati necessari.

Questo lavoro di tesi si pone l'obiettivo di aumentare la collaborazione degli utenti partecipanti al progetto ParticipAct, un esperimento condotto dall'università di Bologna che sfrutta il crowdsourcing.

Al fine di aumentare la collaborazione degli utenti del progetto si raggrupperanno gli utenti in gruppi geolocalizzati, in modo da permettere loro di poter effettuare delle operazioni in comunità.

La tesi è suddivisa in capitoli, ognuno dei quali affronta i diversi temi riguardanti l'analisi e l'implementazione del progetto, andando dal grande al piccolo e dal generale al particolare.

Nel **Capitolo 1** si introdurrà il concetto di crowdsensing, analizzandone l'uso nello scenario di Smart City e descrivendo le applicazioni che hanno avuto maggior successo in questo settore.
Nel **Capitolo 2** si farà una panoramica generale sull'attuale stato del progetto ParticipAct e si parlerà del framework Most, che è uno dei pilastri del progetto.
Il **Capitolo 3** fornirà al lettore un'infarinatura generale su Spring, Android e sulle altre tecnologie utilizzate in questo lavoro di tesi.
Il **Capitolo 4** affronterà in modo dettagliato l'implementazione sia del Server che del Client del progetto ParticipAct.
Nel **Capitolo 5** si spiegheranno i vantaggi dell'introduzione dei gruppi all'interno del progetto e i dettagli implementativi delle nuove funzionalità.
Il **Capitolo 6** affronterà i vantaggi introdotti dal Task Comunitario presentando alcuni dettagli della sua esecuzione.
Nel **Capitolo 7** verranno, invece, mostrati i risultati sperimentali ottenuti attraverso le nuove tecnologie inserite nel progetto.

I risultati ottenuti hanno mostrato fin da subito un forte incremento nella raccolta dei dati. In più, all'interno dei gruppi, si è vista una forte volontà di raggiungere l'obiettivo al più presto.

Capitolo 1: Smart Cities

In questo primo capitolo verranno introdotti i principi fondamentali del Crowdsensing facendo una panoramica dei vari sensori utilizzabili sugli Smartphone e di come il Crowdsensing può essere utilizzato in una Smart City.

Verrà, poi, presentato lo stato dell'arte rappresentato da applicazioni e piattaforme attualmente disponibili per comprendere al meglio il progetto ParticipAct.

Infine, verrà mostrato come queste tecniche possono essere utilizzate per migliorare la qualità della vita degli abitanti in uno scenario di Smart City.

1.1 Crowdsensing

A partire dal 2013 la diffusione degli smartphone si è espansa a macchia d'olio ricoprendo quasi tutto il mercato della telefonia mobile. Ciò è dovuto al fatto che questi dispositivi sono ormai diventati estremamente economici grazie al forte mercato di cui dispongono e alla richiesta di massa a cui sono soggetti.

Nonostante il loro prezzo economico, le loro risorse computazionali sono tutt'altro che povere, queste sono aumentate in maniera esponenziale aprendo la possibilità a nuovi scenari di ricerca e di sviluppo.

Negli ultimi anni, in particolare, si è affermata l'idea di poter sfruttare questi dispositivi per poter raccogliere delle informazioni sull'utente attraverso una raccolta continua di dati sensibili. Questo processo è chiamato crowdsensing.

Questa tecnica è molto diffusa per la sua utilità in scenari dove l'utente è parte attiva della vita cittadina e attraverso il suo smartphone può facilitare il monitoraggio di alcune situazioni d'interesse nel contesto cittadino.

In accordo con i requisiti di partecipazione del cittadino al processo di sviluppo della città, ogni utente può diventare un componente attivo ed essenziale nella raccolta dei dati, fornendo attraverso il suo smartphone delle informazioni di interesse (ad esempio facendo una fotografia).

Questo scenario comporta delle interessanti sfide dal punto di vista sociale e tecnico. Infatti è importante motivare i cittadini alla partecipazione prevedendo incentivi e fornendo nuovi tipi di servizi. Di pari passo è fondamentale che le piattaforme di sensing installate sui dispositivi dell'utente siano il meno intrusive possibile (principio di minima intrusione), in modo da non intaccare sulle abitudini del cittadino nell'utilizzo del suo dispositivo.

In generale, dunque, l'obiettivo del crowdsensing è quello di coordinare possibilmente grandi quantità di gruppi di persone per ottenere dei dati complessi su di essi, sia accedendo ai sensori di cui gli smartphone sono dotati, sia attraverso la partecipazione attiva degli utenti.

Ci sono molte caratteristiche desiderabili di cui un buon modello di crowdsensing deve avvalersi, al fine di rispettare il principio di minima intrusione sugli utenti e allo stesso tempo massimizzare la quantità e la qualità dei dati che vengono raccolti:

- E' fondamentale che venga sempre rispettata la libertà di aderire o meno ad una campagna di crowdsensing.

- Bisogna garantire la privacy di un utente che deve essere in grado di poter scegliere se e quando sospendere o riprendere una attività di raccolta dati.

- Si deve sempre assicurare la trasparenza dei dati che vengono raccolti.

- E' necessario prevedere delle strategie di assegnazione mirate delle attività di raccolta dati in modo da richiedere la partecipazione di utenti che sono realisticamente utili alla campagna.

- Capacità di esprimere la durata e la località di una determinata campagna.

- Lo stato del sistema di crowdsensing deve essere il più coerente possibile.

Sulla base di questi principi molte infrastrutture sono state proposte e alcune verranno analizzate nella sezione stato dell'arte.

Prima di procedere tuttavia all'analisi di questi sistemi, si vuole nel seguito dare una descrizione mirata sulle tipologie di sensori principali presenti negli attuali smartphone che li rendono dei dispositivi

abilitanti al crowdsensing. Successivamente verrà analizzato come questa metodologia possa essere utilizzata in contesti urbani.

1.1.1 Utilizzo di Smartphone come sensori

Mano a mano che gli smartphones andavano affermandosi come piattaforme di computazione efficiente, acquisendo nuove e più ricche funzionalità, l'introduzione di nuove sensori embedded seguiva e sosteneva di pari passo questo processo di sviluppo. Ad esempio, l'accelerometro fu introdotto per la prima volta per rispondere alle esigenze di orientazione necessarie per un utilizzo ottimale dell'interfaccia grafica e della camera del dispositivo [1].

Le informazioni ricavate dal sensore servono per capire la posizione del telefono (verticale od orizzontale) in modo da poter orientare l'interfaccia in modalità landscape o portrait e per posizionare correttamente le foto catturate dalla camera.

In **Figura 1** vengono mostrati i sensori che sono presenti nel dispositivo Samsung Galaxy S4 [2].

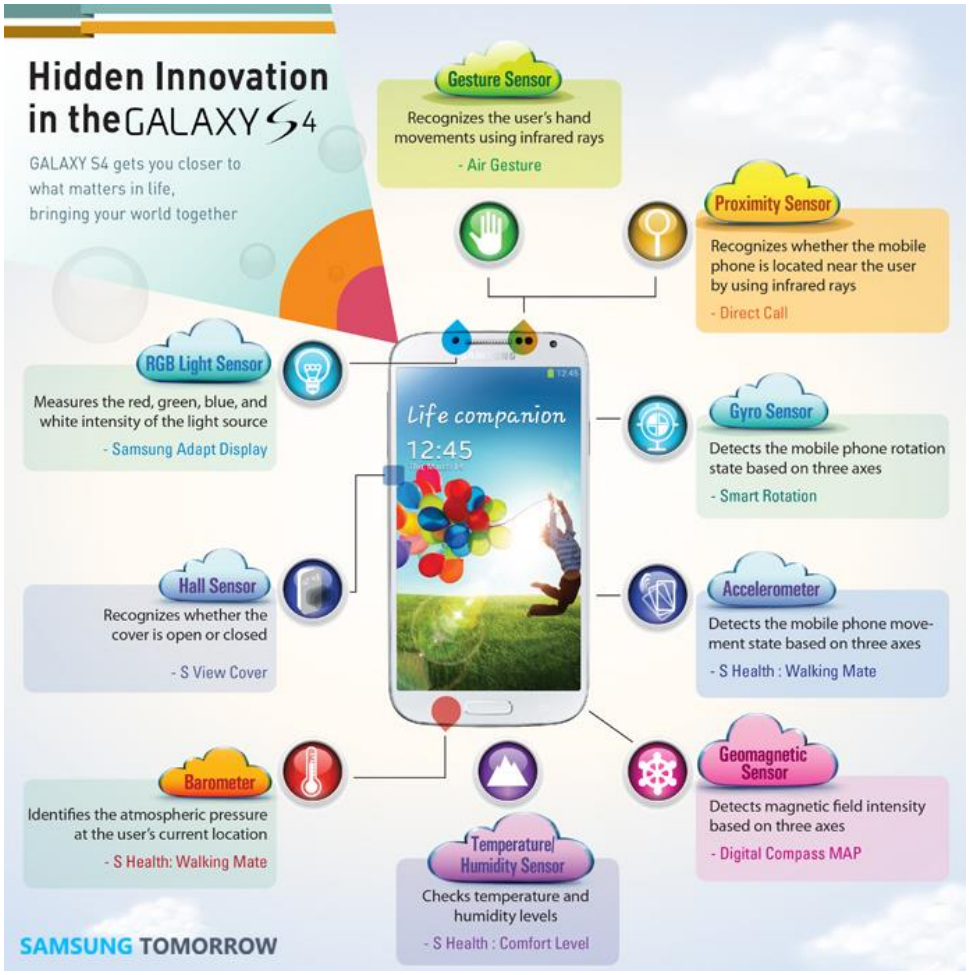

Figura 1: Sensori disponibili su Samsung Galaxy S4

Nel seguito analizzeremo le caratteristiche fondamentali di questi sensori, cercando di spiegare quali sono i principali casi d'uso nei quali trovano applicazione.

I sensori di luce e prossimità permettono al telefono di ottenere delle informazioni contestuali per la gestione dell'interfaccia grafica. Il GPS abilita lo sviluppo di servizi basati sulla locazione del telefono, in questo senso anche la bussola ed il giroscopio rappresentano delle utili estensioni, arricchendo i dati sulla posizione del dispositivo con informazioni come la direzione e l'orientazione.

Nonostante i sensori siano stati principalmente integrati per questioni legate allo sviluppo di applicativi location-based e per una corretta gestione dell'interfaccia grafica, è di facile comprensione che i dati che vengono messi a disposizione, possono essere sfruttati per fini diversi, soprattutto fornendo un'importante opportunità di monitoraggio sulla persona e sui fenomeni che lo circondano.

Tuttavia è evidente che sistemi più complessi possono anche utilizzare la combinazione di più dati di sensing, per offrire dei servizi con maggiori e più articolate funzionalità.

In ultima analisi, va detto che al giorno d'oggi sempre più sensori vengono integrati nei telefoni e la domanda che sorge naturale porsi è quali nuovi sensori possano essere più utili nei prossimi anni.

Rispondere a questa domanda tuttavia non è banale; infatti è difficile predire quali saranno i sensori che verranno integrati nelle prossime generazioni di smartphone. Molto dipenderà dai costi e dallo sviluppo di applicazioni di mobile sensing. Più la loro diffusione aumenterà, ed i costi verranno abbattuti, maggiori saranno i sensori che verranno integrati nei telefoni.

1.1.2 Crowdsensing per smart city

Le statistiche parlano chiaro, il 50 percento della popolazione mondiale vive in città ed entro il 2050 la popolazione nelle città sarà circa di 6.4 milioni di persone, inoltre le città consumano più del 75 percento dell'energia globale e sono responsabili del 80 percento dei gas ad effetto serra.

Queste cifre motivano l'interesse crescente per le smart cities, ossia quelle città per le quali “gli investimenti effettuati in infrastrutture di comunicazione, tradizionali (trasporti) e moderne (ICT), riferite al

capitale umano e sociale, assicurano uno sviluppo economico sostenibile e un'alta qualità della vita, una gestione sapiente delle risorse naturali, attraverso l'impegno e l'azione partecipativa" [3].

Un modo per poter ottenere dati sulla città in maniere scalabile è quello di ricorrere al crowdsensing, ossia delegare questa raccolta ai cittadini che decidono di partecipare attivamente alla vita della città.

Ciò ha diversi vantaggi rispetto ad un approccio tradizionale che richiede lo sviluppo di un'infrastruttura specifica con un elevato costo e non sempre possibile ad esempio per via della conformazione territoriale. Ma questo approccio pone anche nuove sfide legate principalmente alla selezione degli utenti ai quali richiedere i dati di interesse, per cui crowdsensing e infrastruttura fissa possono coesistere per ottenere i migliori risultati.

1.2 Stato dell'arte

La grande quantità di dati di sensing a disposizione ha permesso la diffusione di nuove applicazioni che sono in grado di offrire servizi di forte impatto su molte problematiche del mondo moderno cercando di aiutare a migliorare la qualità della vita.

Nell'ambito della ricerca sono stati fatti grandissimi passi negli studi per la realizzazione di svariati sistemi che porterebbero ad un cambiamento radicale in alcuni settori socio-economici come quello dei trasporti, dei social network, dell'ambiente, della salute e del benessere.

Un forte aiuto in questa direzione è stato offerto dagli smartphone che hanno aperto un nuovo modo per poter affrontare questi problemi con un tipo di approccio totalmente ribaltato rispetto a come è sempre stato fatto.

Un esempio su un possibile utilizzo può essere visto nell'utilizzo delle applicazioni di sensing per monitorare il traffico che, quasi a costo zero, permettono di stravolgere i metodi tradizionali di svolgere questa analisi (possiamo notare come una di queste, Vtrack [4], è stata utilizzata per fornire informazioni del traffico sfruttando l'utilizzo dei telefoni cellulari).

Tutti questi nuovi sensori trovano utilizzo anche in campo ambientale, dove è possibile, ad esempio, monitorare lo stato d'inquinamento di un luogo fornendo ai cittadini un report in tempo reale e dando consigli su come migliorare il proprio stato di vita.

Oltre alle applicazioni in queste macro aree, la nuova moltitudine di sensori permette di utilizzare il proprio smartphone come uno strumento per monitorare lo stato di salute di un individuo in modo da correggere le cattive abitudini.

Una di queste applicazioni è UbiFit Garden [5], sviluppata grazie alla collaborazione tra l'Università di Washington e la Intel: le informazioni sull'attività fisica di una persona vengono catturate ed elaborate al fine di dare degli obiettivi all'utente per raggiungere un migliore stato di benessere.

In questa sezione andremo ad analizzare quali sono i principali sistemi che sono stati sviluppati fino ad oggi. All'inizio introdurremo in modo generale delle classificazioni sulle tipologie di sistemi che si possono avere, in modo da inquadrare nel miglior modo possibile il contesto in cui vanno ad inserirsi. Successivamente analizzeremo nel dettaglio alcuni progetti che sono stati realizzati, per poter discutere le problematiche ed i possibili sviluppi futuri in questo settore.

1.2.1 Classificazione dei vari stati dell'arte

Attualmente abbiamo diversi sistemi di monitoraggio, ma volendo fare una classificazione degli stessi possiamo analizzarli su due piani complementari: la dimensione del gruppo di individui coinvolti nel processo di sensing e il ruolo dell'utente che può essere attivo o passivo (vedi Figura 2).

Per quanto riguarda la dimensione del gruppo di individui, possiamo individuare sostanzialmente tre livelli: personale, di gruppo e comunitario (o pubblico). I dati sono associati al livello personale se sono d'interesse per il solo individuo che li ha raccolti, al livello di gruppo se i dati sono utili ad un gruppo privato di individui e al livello pubblico se sono di interesse pubblico.

Invece, per quanto riguarda il ruolo dell'utente nel processo di raccolta dati possiamo distinguere due diversi tipi di partecipazione : una partecipazione attiva, se l'utente è consapevole della raccolta dei dati per l'applicazione, e una partecipazione opportunistica, se l'utente non è consapevole della raccolta dei dati. Per esplicitare la differenza si può pensare alla richiesta di scattare una foto come esempio di partecipazione attiva, mentre invece il tracciamento della posizione del dispositivo è un esempio di sensing opportunistico.

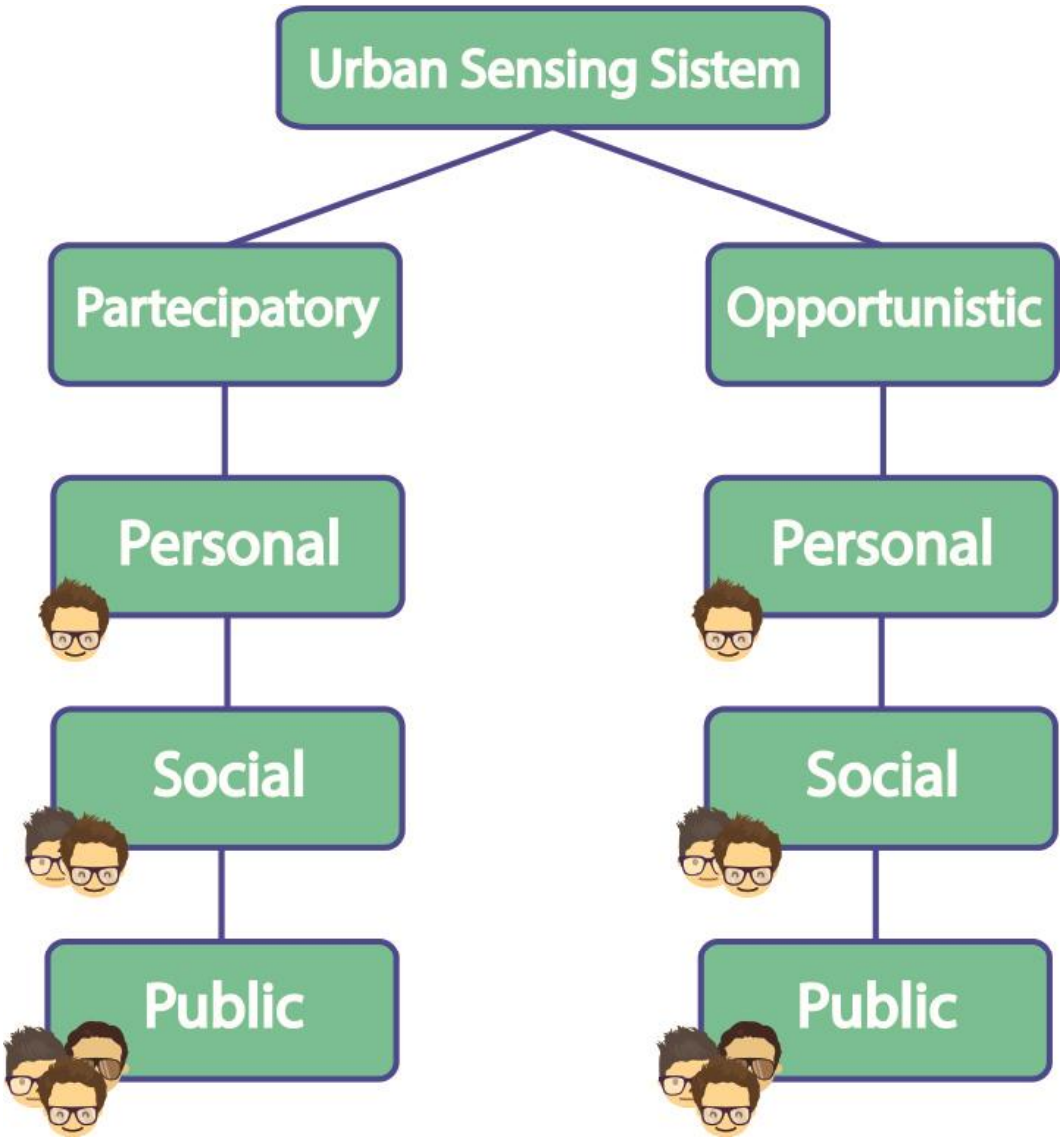

Figura 2: Classificazione dei sistemi di sensing

1.2.2 Panoramica sugli attuali sistemi

Matador [6] è un primo esempio di sistema di crowdsensing che punta ad ottimizzare i consumi energetici sul client mobile. L'architettura del sistema prevede un server ed un client che si sincronizzano periodicamente aggiornando le informazioni sui task disponibili.

Per poter ottenere un significativo risparmio energetico l'applicazione mobile Matador utilizza due diversi sistemi di rilevamento della posizione dell'utente ed un algoritmo che commuta i due sistemi in maniera automatica.

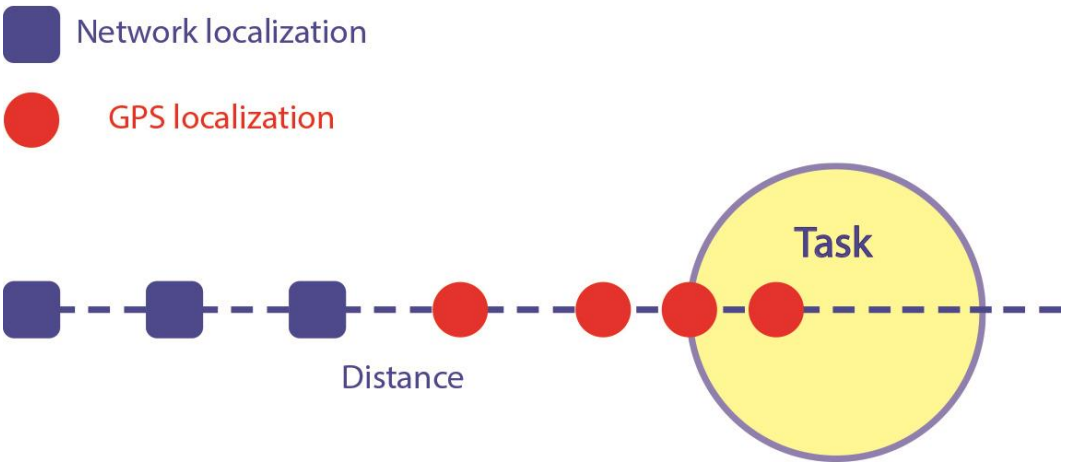

Figura 3: I due sistemi di rilevamento della posizione di Matador

Come si può vedere dall'immagine (vedi Figura 3), il sistema utilizza un approccio più preciso e costoso quando l'utente è nelle prossimità di un Task che può svolgere, mentre un approccio meno preciso e meno dispendioso se l'utente è ancora lontano.

Per capire quando effettuare lo switch di sistema di localizzazione viene presa in considerazione oltre alla distanza dal Task anche la velocità e la direzione con le quali si sta spostando l'utente, il quale verrà notificato solamente dei Task considerati rilevanti così da essere disturbato il meno possibile.

mCrowd [7] è una applicazione per iPhone che permette agli utenti di svolgere e creare Task di diverse tipologie. Possibili Task sono il tag di una data foto, domande testuali, domande bastate sulla locazione degli utenti e richiesta di immagini.

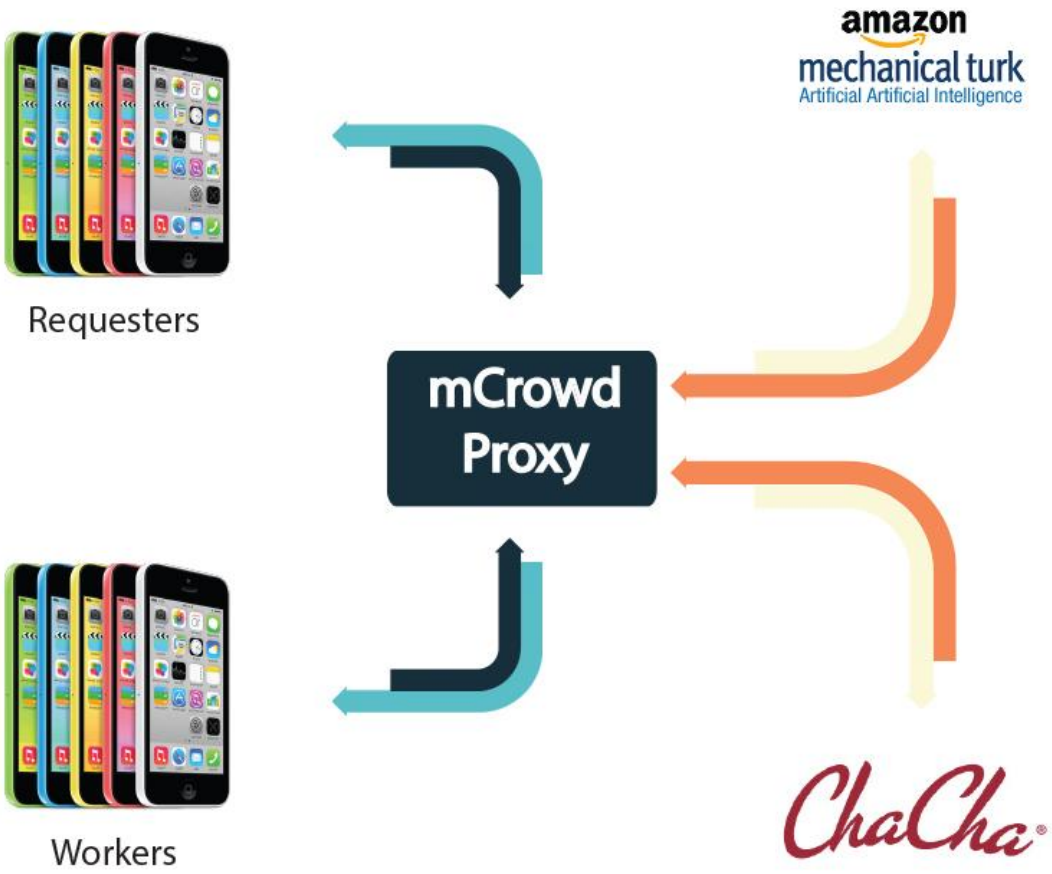

Figura 4: Architettura del sistema mCrowd

Tutte le richieste di creazione di Task sono mediate dall'mCrowd Proxy (vedi Figura 4) che le inoltra agli altri utenti mCrowd e invia ad altri servizi di crowdsourcing come Amazon Mechanical Turk per trovare per trovare altri utenti disposti a esaudire la richiesta.

Medusa [8] propone un linguaggio di programmazione ad alto livello chiamato MediaScript che permette anche ai non tecnici di richiedere l'esecuzione di un Task. Il runtime di Medusa si occuperà di trovare

gli utenti disposti a portare a termine la richiesta mentre il linguaggio fornisce l'astrazione degli stages come passi intermedi in un processo di crowdsensing e dei connectors che hanno il compito di flow control tra gli stages. Per trovare utenti disposti ad eseguire i Task, Medusa si affida ad Amazon Mechanical Turk utilizzando degli incentivi economici.

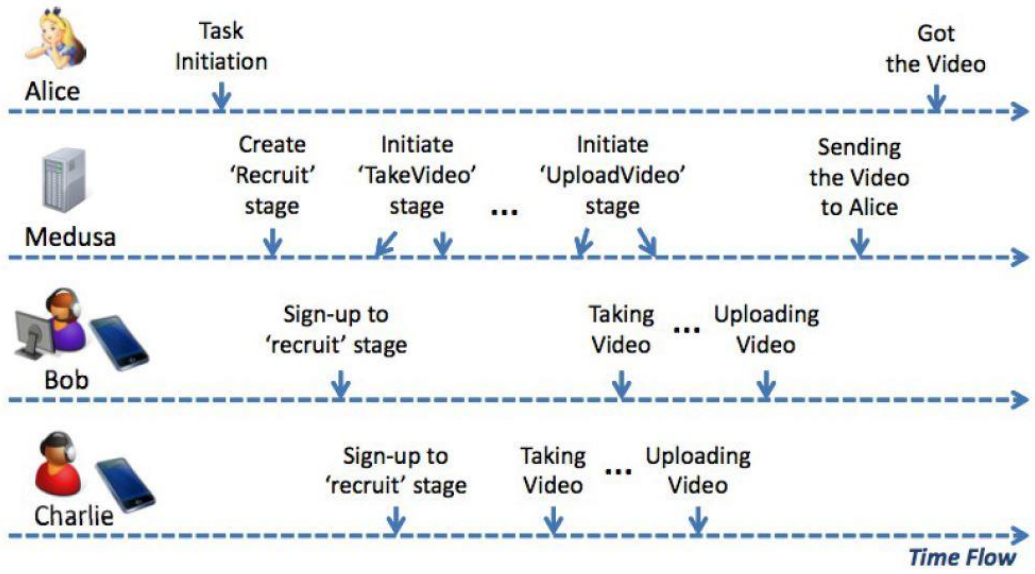

Figura 5: Esempio di richiesta in Medusa

Ohmage [9] è una piattaforma web e mobile che si occupa di raccogliere dati sulla salute delle persone sia in maniera automatica e passiva che chiedendo agli utenti.

Utilizza dei trigger spaziali e temporali per chiedere dei report agli utilizzatori, sistemi di localizzazione come GPS, WiFi e celle telefoniche e analizza l'audio catturato per inferire informazioni sulle interazioni sociali avute durante la giornate.

Tutti questi dati sono caricati in un server, per poter poi essere visualizzati attraverso un sito web (vedi Figura 6).

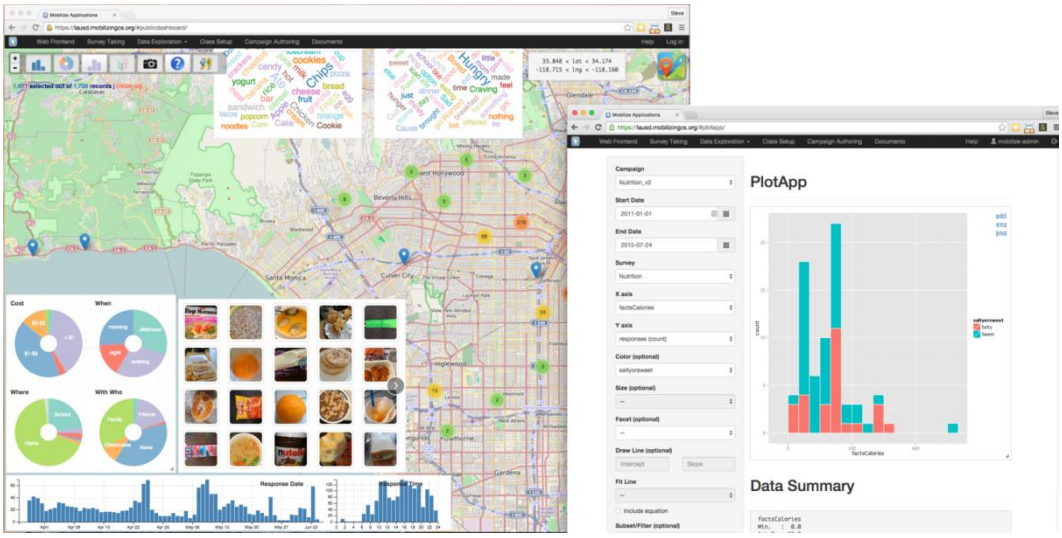

Figura 6: Interfaccia web ohmage

PEIR [10] è una applicazione composta da un client mobile che automaticamente raccoglie dati sugli spostamenti degli utenti e li invia ad un server che analizza l'impatto ambientale dei mezzi che una persona utilizza per spostarsi e l'esposizione degli individui allo smog e ai fast food. Vengono utilizzate sia sorgenti dinamiche di dati, come ad esempio la situazione climatica, che dati statici come la posizione delle scuole e degli ospedali e viene fatta activity detection per stimare con che mezzo di trasposto le persone si spostano. Tutti questi dati sono utilizzati per calcolare sia l'impatto dell'individuo sull'ambiente che l'esposizione di quest'ultimo ad alcuni fattori considerati nocivi.

Ogni utente può visualizzare dei report personalizzati attraverso una interfaccia web (vedi Figura 7).

Figura 7: Interfaccia web di PEIR

USense [11] è un middleware creato per semplificare lo sviluppo di applicazioni di crowdsensing. Permette agli utenti di configurare una serie di preferenze e fornisce agli sviluppatori di applicazioni un supporto per essere notificati quando un determinato evento di interesse è riscontrato, in modo tale da poterlo gestire con la adeguata logica applicativa.

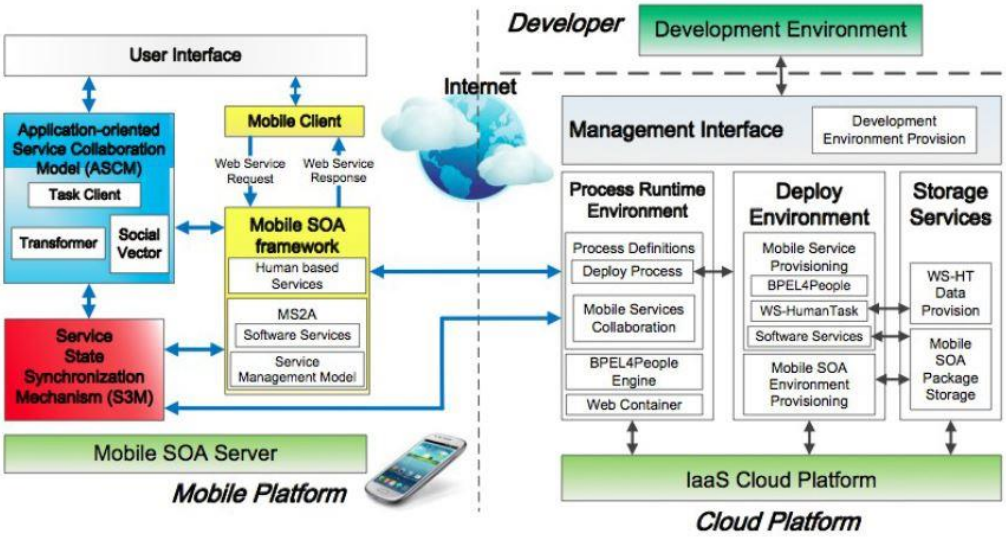

Figura 8: Architettura di Vita

Vita [12] un sistema di crowdsensing ad architettura distribuita (vedi Figura 8) composto da un server centrale e un client mobile. Realizzato utilizzando tecnologie open-source Vita si pone l'obiettivo di realizzare un supporto per le applicazioni di crowdsensing che attualmente devono re-implementare molte funzionalità orizzontali delegabili ad un middleware. Vita permette infatti a diverse applicazioni in esecuzione sullo stesso dispositivo di condividere dati di sensing così da evitare uno spreco di risorse e gestisce anche alcune tipologie di malfunzionamenti come disconnessioni dalla rete o crash.

Anche la parte di comunicazione tra client mobile e server è gestita da Vita seguendo il principio service-oriented architecture (SOA), per cui lo sviluppatore di una nuova applicazione deve solamente implementare la logica di business necessaria.

WiFiScout [13] è un progetto di crowdsensing che si pone come obiettivo quello di raccogliere informazioni sulle reti WiFi. Per motivare gli utenti a condividere le informazioni raccolte viene utilizzato un sistema di incentivazione basato su gamification. Ogni

utente visualizza una mappa divisa in zone (vedi Figura 9) e ogni zona può essere conquistata fornendo informazioni sulle reti presenti.

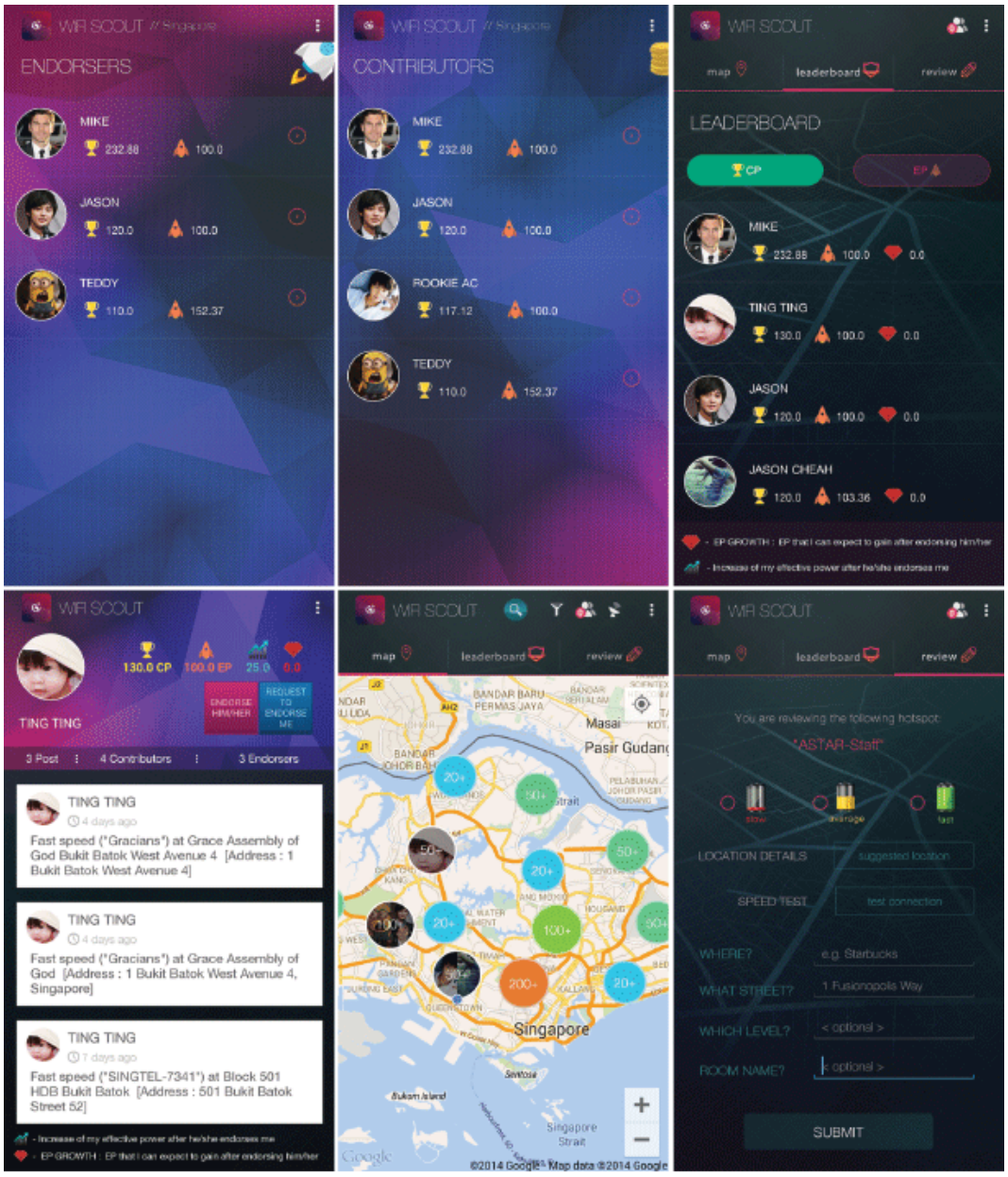

Figura 9: Interfaccia grafica di WiFiScout

NAIST Photo [14] utilizza la piattaforma Foursquare e i check-in degli utenti per proporre alcune richieste di fotografie. Quando un utente effettua il check-in in un locale il server controlla se ci sono richieste per tale luogo ed eventualmente le inoltra all'utente tramite email.

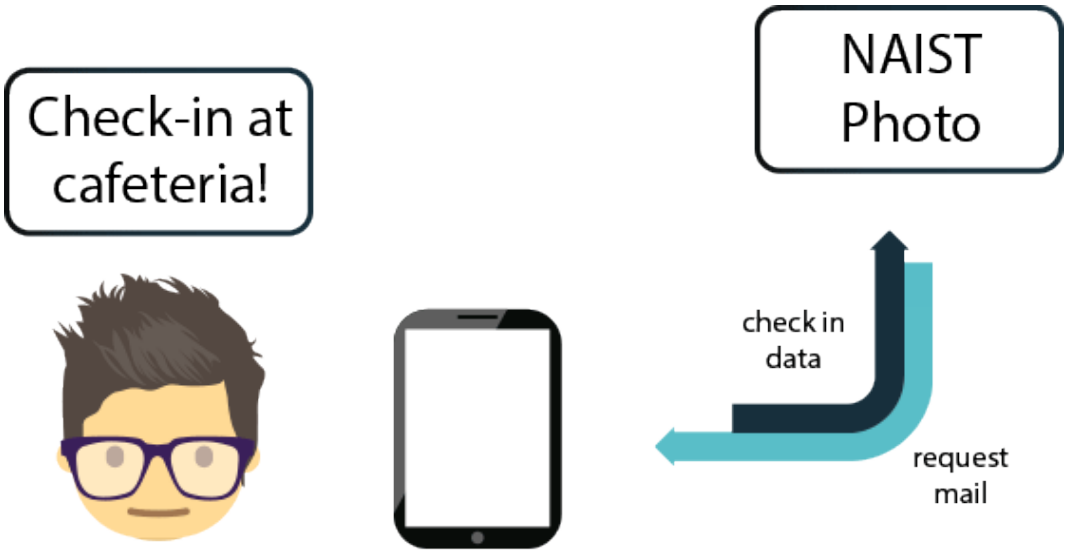

Figura 10: Funzionamento di NAIST Photo

L'utente che riceve la richiesta può quindi decidere di eseguirla ottenendo dei punti in cambio che potranno successivamente essere scambiati con Yen giapponesi.

Alien vs Mobile User [15] raccoglie dati sugli access point WiFi per creare una mappa di un campus universitario. Gli smartphones dei partecipanti acquisiscono informazioni come BSSID, SSID, frequenza e potenza del segnale degli access point che incontrano e per incentivare gli utenti a partecipare il progetto è strutturata come un gioco. Alcuni alieni virtuali sono in giro per il campus e quando un

utente si trova in prossimità di uno di essi può visualizzarlo in una mappa per poterlo individuare e sparargli.

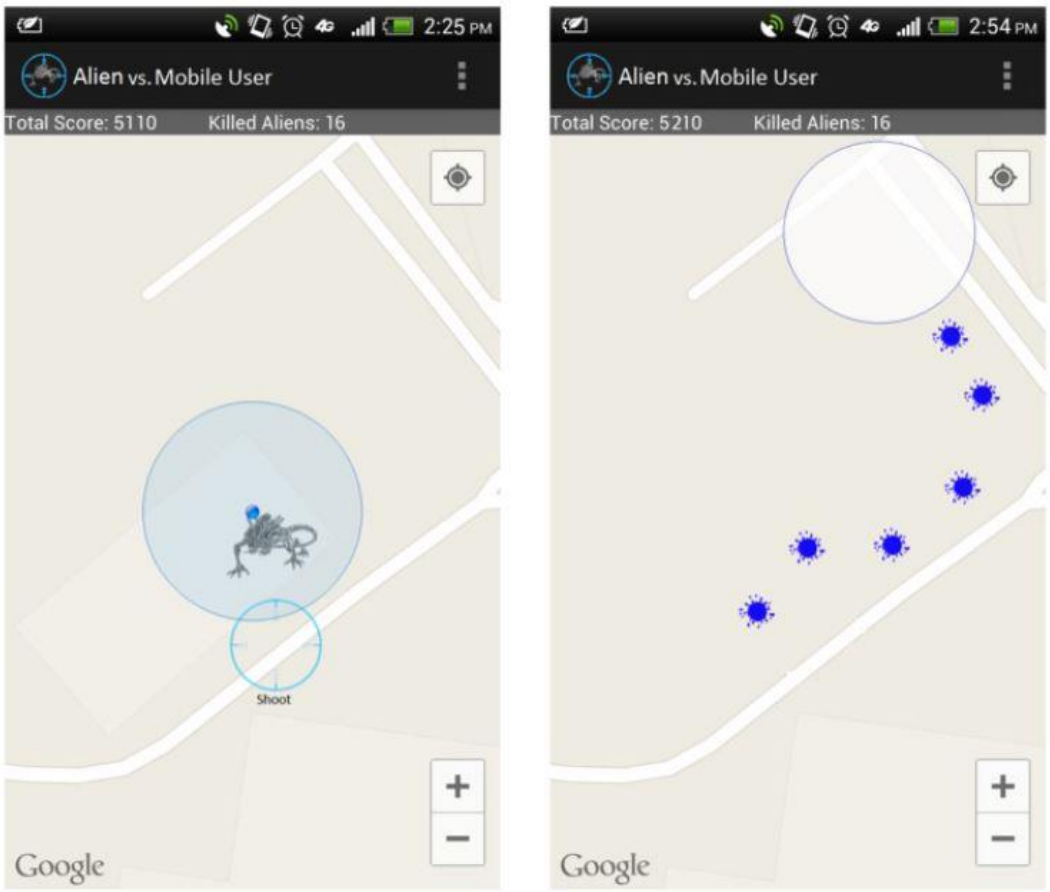

Figura 11: Mappa di Alien Vs Mobile User

Ogni volta che un utente uccide un alieno ottiene dei punti che vengono utilizzati per creare una classifica di tutti gli utenti che stanno giocando. Gli amministratori del progetto variando la posizione degli alieni possono influenzare direttamente i percorsi dei partecipanti all'interno del campus, riuscendo a garantire che non ci siano zone escluse dalla raccolta dei dati.

Twitch [16] propone ai suoi utilizzatori delle micro richieste di vario tipo della durata di qualche secondo. Sfruttando il fatto che le persone sono solite sbloccare il proprio telefono nel tempo libero, Twitch si

pone come alternativa alla solita schermata di sblocco richiedendo di svolgere alcune veloci azioni.

Figura 12: Schermata di Twitch

Le richieste sono di tre tipologie: sondaggi, votazione di foto e valutazione di informazioni estratte automaticamente dal web ai fini di ottenere informazioni strutturate.

SmartRoadSense [17] è un sistema di crowdsensing per il monitoraggio continuo delle condizioni del manto stradale. SRS sfrutta gli accelerometri degli smartphone per rilevare e classificare le irregolarità del manto stradale e trasmetterle a un server che le riconduce alle strade più vicine, le aggrega a quelle ricevute da altri utenti e le mostra su una mappa interattiva pubblicata online. I dati aggregati sono resi disponibili online in modalità aperta per favorirne il riuso. In figura 13 è rappresentata l'interfaccia grafica di SmartRoadSense.

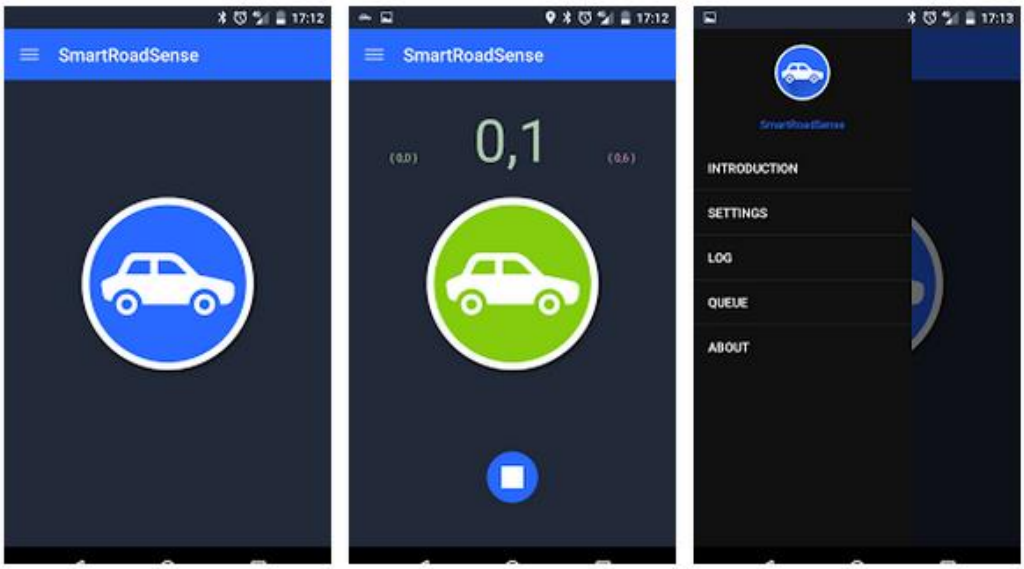

Figura 13: Interfaccia grafica SmartRoadSense

Smartdatanet [18] permette di condividere e aggregare informazioni prodotte da differenti servizi per creare nuove applicazioni, dando vita a comunità più intelligenti e sostenibili. I flussi di dati utilizzati derivano dal mondo delle cose (Internet of things, ad esempio telecamere, sensori di traffico, centraline meteo) e dal mondo delle persone (Internet of people, ad esempio tweet, o segnalazioni inviate da smartphone).

Figura 14: Interfaccia Web di Smartdatanet

Le soluzioni precedentemente descritte possono essere suddivise in categorie non mutuamente inclusive analizzando l'area di applicazione, la dimensione del gruppo di individui coinvolti e il modello di incentivazione utilizzato.

Tra i progetti che si focalizzano sugli aspetti tecnici del crowdsensing con dimensione del gruppo di utenti di una comunità o pubblico e tralasciano la parte di incentivazione troviamo Matador e USense. Il focus in questi progetti è fornire un'infrastruttura software ottimizzata e in grado di coordinare le richieste degli amministratori della piattaforma con gli utenti che ne fanno parte.

Esiste una classe di progetti che ha avvertito la necessità di fornire meccanismi di incentivazione agli utenti ma ha deciso di sfruttare una piattaforma già presente: Amazon Mechanical Turk (MTurk). MTurk è un servizio internet di crowdsourcing che permette attraverso l'uso di application programming interface (API) di pubblicare richieste di esecuzione di alcune azioni. I Requester possono pubblicare obiettivi conosciuti come HIT (Human Intelligence Tasks) e i Worker, o informalmente Turker, possono ricercare tra gli obiettivi esistenti e completarli in cambio di un pagamento deciso dal requester. In questa classe di progetti troviamo mCrowd, Medusa e Vita.

Progetti destinati ad un uso personale dei dati raccolti per avere un feedback sul proprio stile di vita sono Ohmage e PEIR. In questi sistemi non è previsto un sistema di incentivazione, perché l'utente che decide di partecipare è già fortemente motivato a raccogliere dati essendone il beneficiario.

WiFiScout, NAIST Photo e Alien VS Mobile User utilizzano invece tecniche di gamification come punti e classifiche per aumentare il coinvolgimento degli utenti nel progetto. I primi due operano in una scala pubblica mentre l'ultimo è pensato per un campus universitario. NAIS Photo è in realtà una soluzione ibrida poiché permette agli

utenti di scambiare i punti accumulati durante l'utilizzo del sistema in denaro, utilizzando quindi i punti come una moneta virtuale che può essere convertita in una moneta fisica come lo Yen giapponese.

La peculiarità di Twitch è, invece, la struttura completamente basata sul gioco. Le stesse richieste di raccolta dati sono presentate come sfide da completare in breve tempo, con il premio di poter sbloccare il proprio dispositivo.

Gli ultimi due sistemi, SmartRoadSense e Smartdatanet, voglio mostrare come le metodologie del Crowdsensing stanno interessando realtà sempre più estese. Smartdatanet, in particolare, dimostra il forte interesse di enti pubblici nello sviluppare piattaforme di Crowdsensing sempre più utili e avanzate.

1.3 Comparazione dei sistemi

I sistemi di Crowdsensing esistenti permettono ai cittadini e alle amministrazione di poter migliorare il benessere di una comunità. Va, però, precisato che ogni singola applicazione svolge un ruolo diverso a seconda del campo d'azione.

La seguente tabella mostra tutti gli esperimenti discussi in precedenza evidenziando l'area di applicazione e la dimensione dell'esperimento.

Nome	Area di Applicazione	Dimensione Esperimento
Matador	Crowdsensing	Pubblico
mCrowd	Crowdsensing	Pubblico
Medusa	Crowdsensing	Pubblico
Ohmage	Health Monitoring	Personale
PEIR	Environmental Impact Report	Personale
USense	Crowdsensing	Pubblico
Vita	Crowdsensing	Pubblico
WiFiScout	Crowdsensing	Pubblico
NAIST Photo	Crowdsensing	Pubblico
Alien VS Mobile User	Crowdsensing	Comunità
Twitch	Crowdsensing	Pubblico
SmartRoadSense	Crowdsensing	Pubblico
Smartdatanet	Crowdsensing	Pubblico

1.4 Direzioni Future

Come è facile immaginare, la natura di questo campo di ricerca è ancora molto giovane, quindi ci sono ancora diversi quesiti da risolvere, non solamente di natura tecnologica. Uno di questi potrebbe essere che non tutti gli utenti vogliono condividere la stessa quantità di informazioni con gli altri e trovare il giusto bilanciamento tra garanzia della privacy per ogni utente e benefici per una community può essere un problema non banale. Parallelamente a questo si pone anche il problema del garantire sicurezza nelle comunicazioni tra nodi mobili e server ove previsti.

Un altro problema resta quello di capire come può essere possibile raggruppare gli individui basandosi sui luoghi che visitano di recente o sul tempo che passano insieme. Una volta raggruppati, si potrebbe far loro svolgere determinate azioni in modo da rafforzare il proprio rapporto, oppure si potrebbero incentivare i gruppi a spostarsi per poter visitare posti sempre diversi in modo da avere una raccolta dati uniforme in ogni punto della Smart City.

Sempre tra i problemi ancora non risolti vi è quello legato all'incentivazione degli utenti in quanto non tutti gli individui sono disposti a condividere dati personali a favore della collettività e avere una ricompensa in cambio dei dati raccolti può aumentare la qualità e la quantità dei dati disponibili.

Un altro punto in cui sono possibili miglioramenti è l'ottimizzazione dei consumi energetici su smartphone e una possibile via percorribile per avere miglioramenti è avere un layer tra le applicazioni che richiedono dati ai sensori ed i sensori stessi. Questo layer potrebbe permettere una condivisione di dati tra diverse applicazioni, utilizzando una cache condivisa, in modo da ridurre l'effettiva accensione di alcuni sensori.

Capitolo 2: ParticipAct e MoST

In questo capitolo parleremo dell'esperimento di crowdsourcing svolto dall'Università degli Studi di Bologna chiamato ParticipAct [19] e di MoST, un middleware che fornisce le funzionalità orizzontali per applicazioni di mobile sensing sviluppato per piattaforma Android. L'obiettivo di questo esperimento è molto semplice: si cerca di sfruttare la collaborazione spontanea di utenti in modo da far loro fornire dati utili per l'elaborazione di analisi statistiche attraverso lo svolgimento di unità si sensing denominate Task.

Verrà prima illustrata la piattaforma ParticipAct introducendo il concetto di Task e l'architettura distribuita del sistema, per poi focalizzare l'attenzione su MoST.

2.1 ParticipAct

ParticipAct è un progetto di crowdsensing dell'Università di Bologna che permette ai partecipanti di contribuire ad una raccolta dati sia in maniera passiva che attiva. Questi dati, una volta raccolti e inviati al server, vengono analizzati su diverse scale, dal singolo individuo all'intera città, per ottenere delle informazioni di alto livello utilizzabili dalla smart city.

Attraverso l'elaborazione di questi dati è possibile trarre numerose conclusioni su svariati argomenti come l'individuazione delle aree di una città che vengono maggiormente utilizzate per le interazioni sociali, le linee di trasporto pubblico sovraccaricate, le informazioni sul traffico in una data posizione, i posti preferiti dai cittadini per svolgere attività fisica e molto altro ancora.

Attraverso queste informazioni i dirigenti della città potrebbero migliorare le zone di interesse in modo da far salire il benessere dei propri cittadini.

Le informazioni sulla mobilità possono essere utilizzate inoltre per fornire suggerimenti su come muoversi all'interno della città sfruttando le informazioni sulla mobilità abituale di ciascun individuo e le condizioni climatiche al fine di ridurre l'inquinamento.

2.1.1 Task

Nella struttura di ParticipAct era necessario un elemento che permettesse agli amministratori di poter interrogare gli utenti. Il Task è un compito di sensing che viene assegnato ad uno o più membri del progetto ed è caratterizzato da cinque categorie di informazioni:

- **Elementi descrittivi:** quali nome, descrizione, durata stimata e area geografica di interesse.
- **Azioni:** ad esempio una richiesta di foto, un questionario o l'attivazione di un sensore dello smartphone.
- **Stato:** indica a che punto nel proprio ciclo di vita il Task si trova.
- **Partecipanti:** una serie di utenti ai quali è stato richiesto di eseguire.
- **Dati:** i risultati delle azioni associate al Task, possono essere sia dati grezzi come i valori provenienti direttamente da un sensore oppure un'inferenza generata a partire da essi.

Per elementi descrittivi si intende l'id (univoco all'interno di ParticipAct), il nome, la descrizione, la durata stimata necessaria per un corretto svolgimento, la scadenza entro la quale il Task è accettabile, l'area geografica della quale si desidera ottenere informazioni ed il punteggio assegnato.

Ogni Task può avere diverse azioni associate che si differenziano in:

- **Sensing:** in questo caso viene richiesta l'attivazione di uno o più sensori, come GPS e accelerometro, presenti sul dispositivo.
- **Foto:** viene chiesto all'utente di scattare ed inviare una fotografia dell'elemento specificato.
- **Questionari:** una o più domande che possono essere a risposta aperta o chiusa.

Nel caso di risposta chiusa è possibile avere sia una domanda a risposta singola che a risposta multipla.

Un'azione può essere attiva, ossia che richiede l'esplicito intervento dell'utente per essere portata a termine (foto e questionari ne sono un esempio), o passiva (come nel caso di sensing).

Lo stato di un task viene utilizzato per gestirne il ciclo di vita (vedi Figura 15), i principali valori sono:

- **Available:** l'utente può ancora decidere se accettare o rifiutare il Task.
- **Running:** il Task è già stato accettato e la raccolta dei dati desiderati è in corso.
- **Refused:** l'utente ha deciso di rifiutare il Task.
- **Succeeded:** i dati richiesti sono stati raccolti con successo.
- **Failed:** l'utente ha accettato il Task ma la raccolta dei dati non è andata a buon fine.

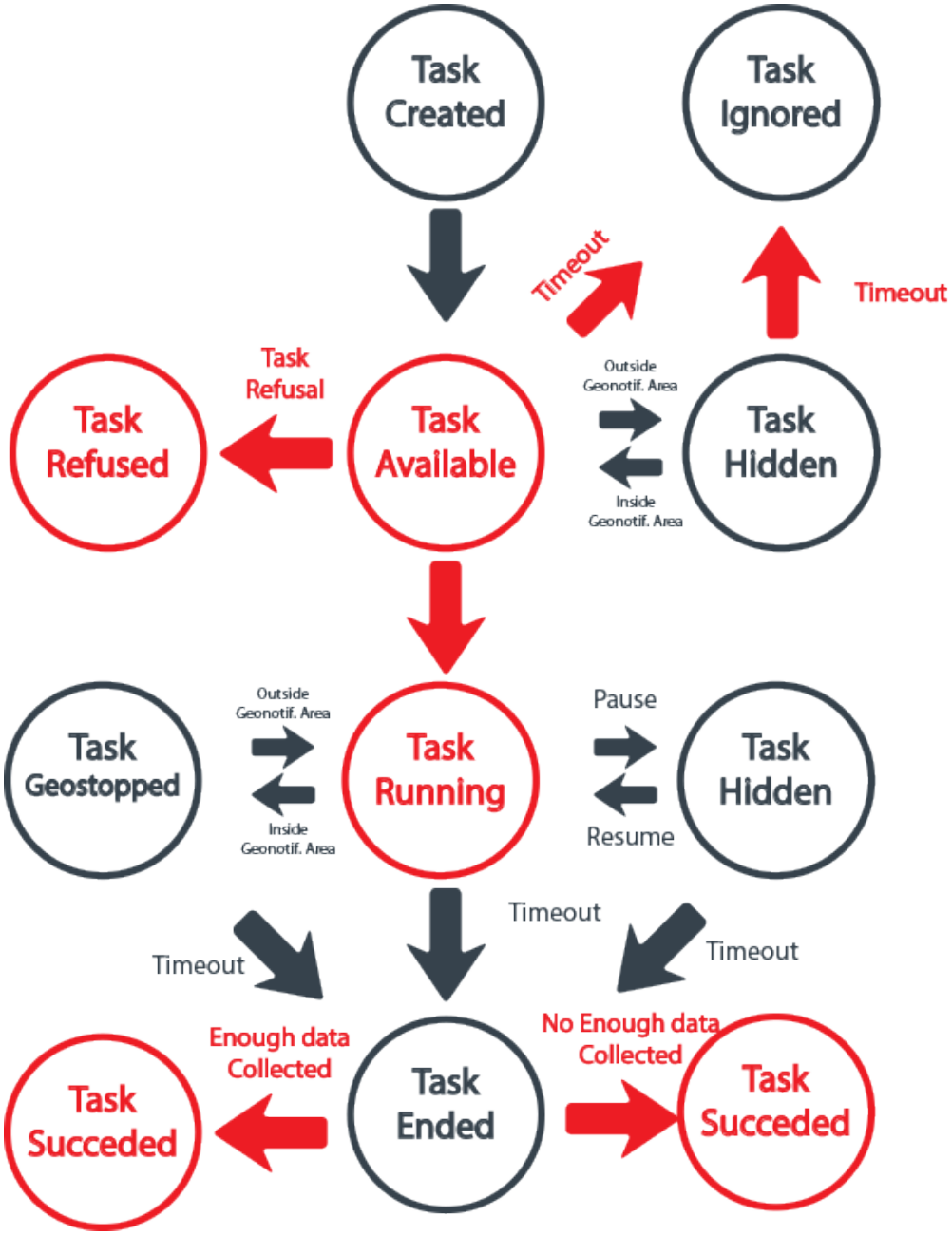

Figura 15: Ciclo di vita di un Task

Quando un Task viene creato è inizializzato con lo stato available e non cambia finché l'utente decide di eseguirlo (running) o rifiutarlo (refused). Se l'utente accetta il task e i dati raccolti sono considerati validi il task si porta nello stato succeeded altrimenti nello stato failed. I task che hanno anche informazioni di geolocalizzazione controllano

periodicamente la posizione attuale interrompendo la raccolta dei dati se è al di fuori dell'area di interesse (geostopped).

Quando un utente cambia lo stato di un Task interagendo con esso, ad esempio perché lo accetta questa informazione viene propagata al server che tiene quindi traccia dell'andamento della richiesta tra i diversi utenti a cui è stato assegnato. Non tutti i cambiamenti di stato sono però propagati, solamente quelli ritenuti significativi. Ad esempio se lato client un Task viene momentaneamente messo in pausa e successivamente ripreso questa informazione non è ritenuta di interesse e non viene quindi propagata. Gli unici cambiamenti di stato che vengono propagati sono quelli che portano il Task in stato available, refused, ignored, running succeeded o failed.

I dati sono il risultato dell'esecuzione del task e sono eterogenei. Possono ad esempio essere una fotografia o dei dati grezzi raccolti da uno dei sensori dello smartphone.

2.1.2 Architettura del Task

La piattaforma ParticipAct è concettualmente divisa in due moduli: uno dedito al sensing ed uno riguardante la gestione della smart city (vedi Figura 16).

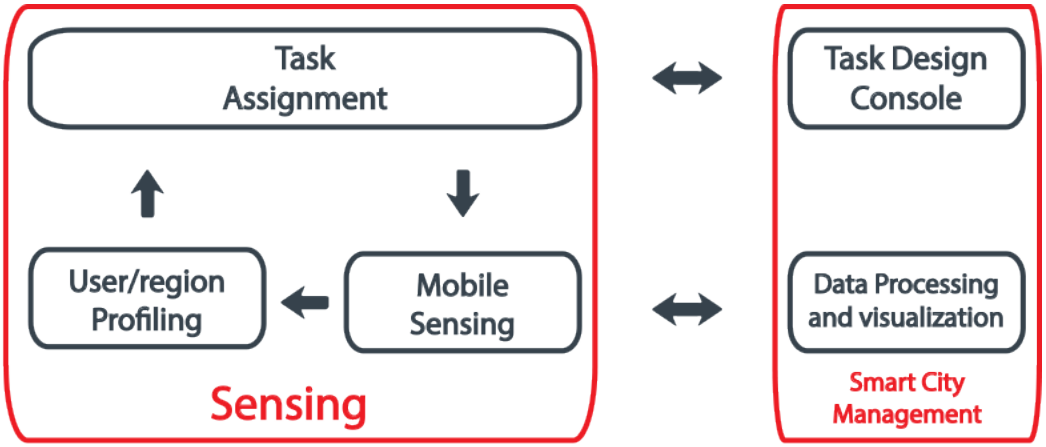

Figura 16: Struttura della piattaforma ParticipAct

Questi moduli astratti sono implementati in un'architettura distribuita (vedi Figura 17) realizzata seguendo il modello client-server e le responsabilità sono divise in tre componenti: un'applicazione mobile che effettuerà il sensing dei dati, un back end database per la persistenza e la gestione dei dati, e una console di gestione web per gli amministratori.

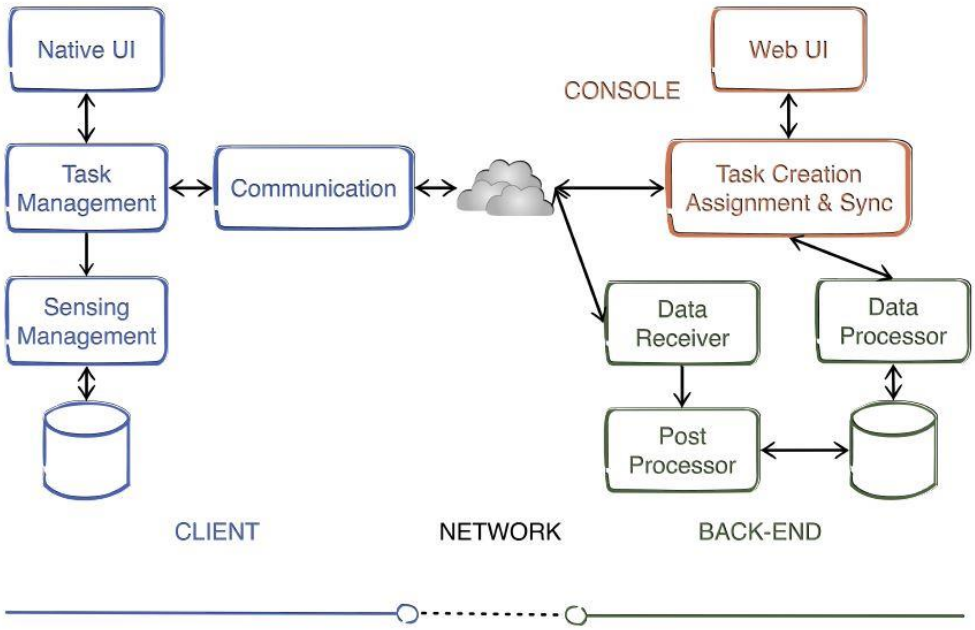

Figura 17: Architettura distribuita di ParticipAct

Come sarà spiegato in maniera dettagliata nei capitoli successivi l'introduzione della creazione di gruppi geolocalizzati e l'estensione della parte relativa alla gamification comporterà sia l'aggiunta di nuovi componenti che la modifica di quelli mostrati in figura. Tutto ciò viene fatto per incentivare ulteriormente gli individui a partecipare alla raccolta dati in gruppo in modo da avere informazioni sempre più precise sull'attendibilità e sulla qualità dei dati trasmessi.

Anche la Console sarà estesa permettendo agli amministratori di configurare e sfruttare i nuovi elementi introdotti. Il Back-End dovrà gestire i nuovi dati prodotti dagli utilizzatori e fornire risultati alla console.

Ciascun componente è realizzato da moduli a grana più fine che verrano ora illustrati.

Data Back-End

Suddiviso in tre moduli con diverse responsabilità:

- **Data Receiver:** l'interfacciamento con l'applicazione mobile, riceve il risultato del sensing svolto dagli utenti.
- **Post Processor:** pulisce i dati ricevuti e li prepara per essere salvati in maniera persistente nel database attraverso operazioni di interpolazione e integrazione.
- **Data Processor:** elabora inferenze di alto livello dai dati ricevuti al fine di identificare gli utenti che hanno la maggior probabilità di portare a termine un Task. Fornisce inoltre una stima del tempo necessario al completamento di un Task in base alle informazioni storiche e profili temporali e geografici degli utenti.

Il modulo di Data Processor fornisce agli amministratori del sistema una lista di utenti a cui assegnare un Task usando una delle seguenti politiche:

- Random.
- Attendance.
- Recency.
- Dbscan.

La politica random seleziona un insieme di utenti in maniera casuale. La politica attendance sceglie gli utenti in base al tempo totale trascorso nella zona di interesse per il Task, mentre la politica recency privilegia gli utenti che più recentemente si sono trovati nella zona di interesse. Per ultima la politica dbscan considera solamente gli utenti che hanno trascorso in passato un quantitativo di tempo significativo nell'area di interesse. Per ottenere questo risultato viene utilizzato l'algoritmo DBSCAN con i dati sulle posizioni passate degli utenti [20]. Tale algoritmo ha proprietà che lo rendono adatto per la selezione degli utenti a cui assegnare Task poiché non richiede la conoscenza a priori del numero di cluster, riesce ad individuare cluster di forma arbitraria, è robusto al rumore e agli outliers ed è ottimizzato per essere eseguito con database GIS. Una particolarità di questa ultima politica è che non fornisce un ranking di utenti come le precedenti, per cui gli utenti a cui assegnare il Task sono un sottoinsieme, scelto casualmente, di tutti gli utenti individuati da DBSCAN.

Indipendentemente dalla politica scelta il sistema scarta tutti gli utenti con una percentuale di batteria residua inferiore ad una soglia come si può vedere in Figura 18.

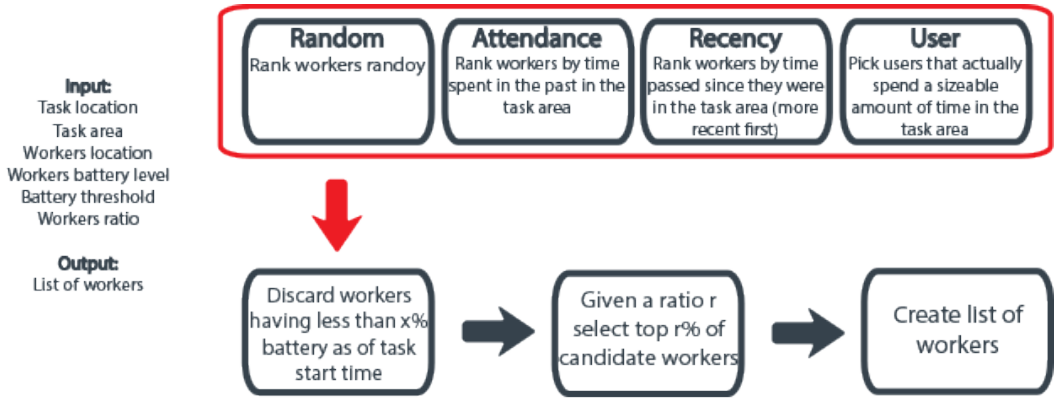

Figura 18: Flusso di creazione Task in ParticipAct

Console

Questo componente è lo strumento attraverso il quale gli amministratori interagiscono con la piattaforma utilizzando un'interfaccia web.

Il modulo di **Task Creation Assignment & Sync** permette agli amministratori di creare nuovi Task specificando la regione di interesse, la durata e il tipo di dato desiderato e assegnarli agli utenti con una delle politiche precedentemente descritte.

Inoltre è responsabile di notificare agli utenti scelti la presenza di un nuovo Task da eseguire e di mostrare agli amministratori i dati ricevuti.

Client

Questo componente è realizzato tramite un'applicazione mobile che fornisce le funzionalità orizzontali per la gestione di un Task (vedi Figura 19) come ricezione delle notifiche, l'interazione con l'utente, l'upload dei dati al server centrale e delega l'effettivo raccoglimento dei dati ad un framework sottostate chiamato MoST che verrà illustrato successivamente.

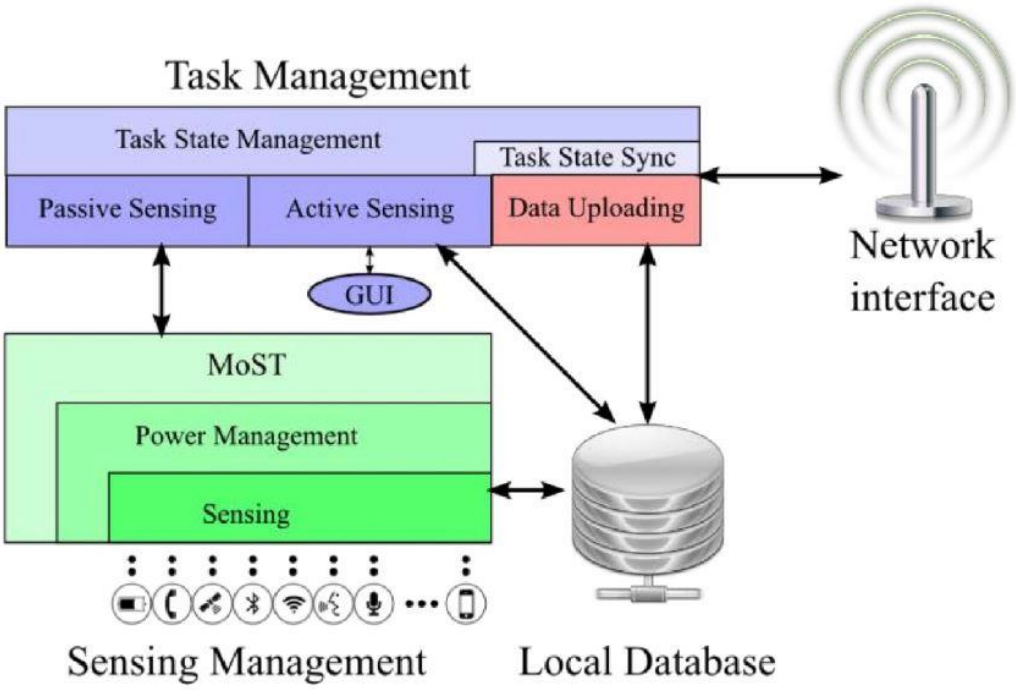

Figura 19: Architettura client mobile ParticipAct

I moduli che compongono questo componente sono:

- **Task Management:** viene allertato dal server per la presenza di nuovi Task disponibili e a sua volta notifica il server dei cambiamenti di stato del Task. Per l'effettiva raccolta dei dati delega il lavoro al modulo di Sensing Management, mostrando all'utente un'interfaccia grafica (GUI) nel caso sia richiesta una partecipazione attiva (come nel caso di richiesta di fotografia o questionario).
- **Sensing Management:** responsabile della raccolta dei dati richiesti. Nel caso dell'applicazione Android utilizza il framework MoST.
- **Communication (Network Interface):** riceve le richieste dal server centrale, attiva il modulo di Task Management e inoltra i dati raccolti dal modulo di Sensing Management al server.

2.2 Most

MoST [21] è un middleware che fornisce le funzionalità orizzontali per applicazioni di mobile sensing sviluppato per piattaforma Android. Per poterne apprezzare l'utilità vale la pena considerare tutti gli aspetti che uno sviluppatore dovrebbe ogni volta ripercorrere se non utilizzasse un framework con tali funzioni.

Innanzitutto senza un tale supporto, che come vedremo in seguito astrae i dettagli di ogni sensore, uno sviluppatore dovrebbe conoscere le API specifiche per ogni sensore interessato e gestire in maniera ottimale la concorrenza considerando che anche altre applicazioni potrebbero aver bisogno dello stesso sensore. Inoltre risulta molto più efficace gestire il consumo energetico se le effettive chiamate ai sensori hardware sono mediate da un controllore che possa gestire una cache di alto livello tra diverse applicazioni, senza dover ottimizzare i consumi per ogni singola applicazione.

MoST permette quindi agli sviluppatori di applicazioni mobile di concentrarsi solo sulla logica applicativa senza doversi preoccupare della gestione delle risorse, analogamente a quanto fa un container per applicazioni Java enterprise. Oltre ad esporre i dati grezzi MoST permette agli sviluppatori di ottenere direttamente inferenze di alto livello. La gestione delle risorse è aware dei vincoli dell'OS, ad esempio essendoci il vincolo di mutua esclusione nell'uso del microfono MoST interromperà automaticamente la raccolta di dati dal microfono nel momento in cui il telefono riceve una chiamata. Sempre per sottostare ai limiti del sistema operativo, che nel caso di Android si traduce in un calo di performances all'aumentare dell'instanziazione di nuovi oggetti a causa dell'attivazione del garbage collector, MoST utilizza un pool di risorse che vengono riutilizzate ogni qual volta possibile.

Architettura

In questa sezione si introdurrà solamente l'architettura del framework, per i dettagli implementativi si rimanda al capitolo 4 sezione 3. Prima di scendere nei dettagli dell'architettura di MoST vengono presentati i due componenti fondamentali di MoST: Input e Pipeline.

L'Input è un'astrazione per qualunque tipo di sensore, virtuale o fisico, gestito da MoST che fornisce dati grezzi, mentre la Pipeline è un componente di più alto livello che incapsula la logica applicativa di inferenza a partire dagli Input verso i quali ha mostrato interesse. Gli Input si possono suddividere in due categorie: Input che raccolgono dati continuamente e Input che raccolgono dati periodicamente in base a politiche energetiche.

Il framework è diviso in due componenti con responsabilità separate (vedi Figura 20): uno è dedito all'effettiva raccolta di dati mentre l'altro gestisce il primo considerando eventi scatenati dal sistema operativo, da utente o dal framework stesso e coordinando le risorse necessarie utilizzando un pool e politiche di risparmio energetico.

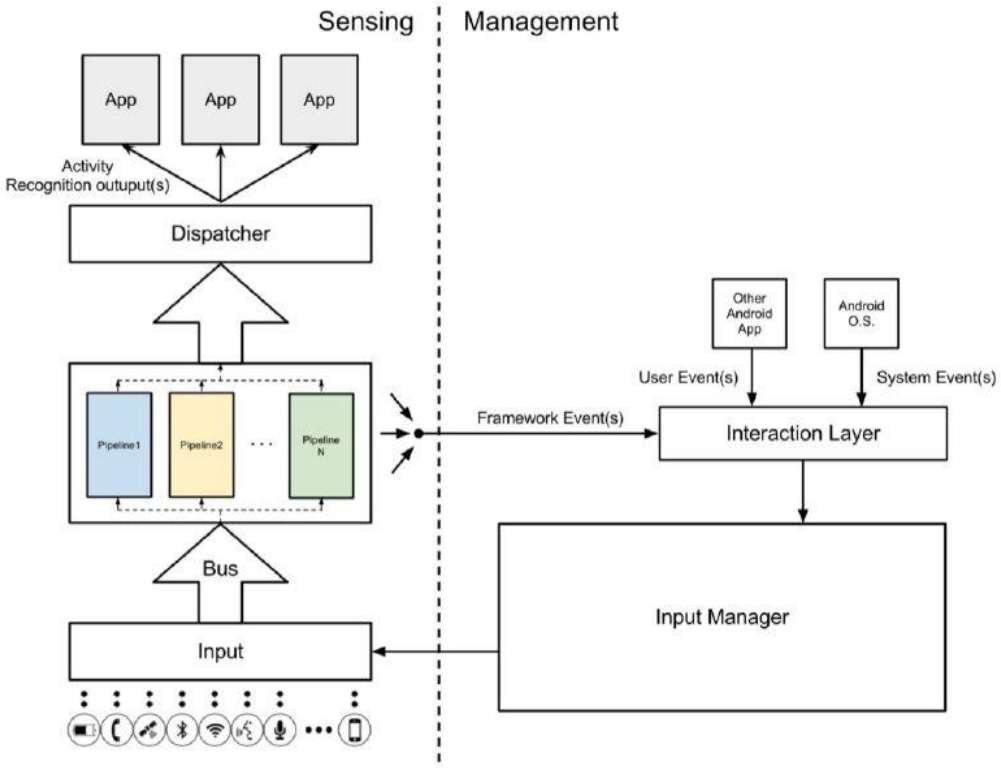

Figura 20: Architettura del framework MoST

I l sottosistema di sensing ha un'architettura multi layer nella quale troviamo alla base gli Input che vengono propagati attraverso un bus alle Pipeline interessate.

Questa Pipeline dopo le opportune elaborazioni consegneranno al Dispatcher, il punto di collegamento con le applicazioni che utilizzano il framework, i risultati delle inferenze.

La parte di gestione è a sua volta suddivisa in due moduli: l'Interaction Layer che è in ascolto di eventi dal sistema operativo e dal framework stesso per gestire situazioni di concorrenza e politiche di risparmio energetico, e L'Input Manager che gestisce il ciclo di vita degli Input per ottimizzare l'utilizzo di risorse.

Questo capitolo ha introdotto il progetto ParticipAct mostrandone l'architettura e i componenti attualmente presenti. Questa prima analisi ad alto livello ha permesso di capire che la nuova logica che attraverso la creazione di gruppi geolocalizzati affiancherà ed estenderà la già presente gamification per aumentare sia il livello di gradimento che il coinvolgimento degli utenti sarà distribuita nei vari componenti che compongono l'architettura.

Capitolo 3: Tecnologie Utilizzate

In questo capitolo analizzeremo alcune tecnologie che vengono utilizzate per sviluppo di piattaforme distribuite. L'obiettivo di questa analisi è quello di fornire delle nozioni molto generali su alcuni dei sistemi scelti per l'implementazione del progetto ParticipAct che verrà ampiamente discusso nel prossimo capitolo.

Nella trattazione verranno presentate nell'ordine:

- **l'architettura Restfull:** un modello architetturale per la comunicazione tra componenti distribuiti;
- **il framework Spring** [22]: tecnologia prevalentemente, ma non necessariamente, server-side basato sul concetto di container leggero;
- **il sistema Android** [23]: sistema composto da uno stack software che parte dal sistema operativo ed arriva a fornire API di alto livello per gli sviluppatori in contesti mobili.

3.1 Architettura RESTfull

La sigla RESTfull sta per REpresentational State Transfer e simboleggia uno stile architetturale di soluzione Resource Oriented. Nell'ultimo decennio questa tecnologia ha guadagnato un tal consenso da arrivare a rappresentare una più che valida alternativa ai sistemi Service Oriented. I sistemi RESTfull comunicano sfruttando il protocollo http/https e le sue primitive (GET, POST, PUT, DELETE).

È importante notare che REST non ha nulla a che vedere con le chiamate a procedura remota in quanto è uno stile architetturale Resource-Oriented che vuole enfatizzare lo scambio di messaggi descriventi risorse e non che abbiano il fine di scatenare un'azione.

Per capire meglio questo stile di comunicazione si provi ad analizzare il suo nome:

- Representational: le risorse, in REST, possono essere rappresentate in varie forme. Ad esempio, XML, JavaScript Object Notation (JSON) o anche HTML o qualunque forma sia ritenuta migliore dai progettisti di un sistema;
- State: quando si utilizza il paradigma REST si è focalizzati sullo stato delle risorse più che sulle azioni che si possono intraprendere su di esse;
- Transfer: indica il trasferimento di dati intesi come risorse auto descrittive da un'applicazione all'altra.

Nelle sottosezioni successive analizzeremo gli elementi essenziali di questa architettura. Si ricordi sempre che un requisito fondamentale del paradigma è che la comunicazione sia sempre senza stato. L'interazione tra cliente e servitore non deve prevedere il concetto di sessione ed ogni operazione sulle risorse deve essere auto contenuta.

3.1.1 Identificazione delle Risorse

A differenza dei Web Services SOAP-oriented che sono basati sul concetto di chiamata remota i Web Services RESTful sono basati sulle risorse, intese come qualsiasi elemento oggetto di elaborazione. Ciascuna risorsa deve essere identificata univocamente ed essendo in ambito web il meccanismo più naturale per individuare una risorsa è dato dal concetto di URI. Alcuni esempi validi sono:

- http://www.example.com/task/18
- http://www.example.com/user/83

3.1.2 Utilizzo esplicito dei metodi http

Secondo questo principio bisogna sfruttare i metodi predefiniti del protocollo HTTP, ovvero GET, POST, PUT e DELETE per elaborare le risorse stabilendo una mappatura uno ad uno tra le tipiche operazioni CRUD e i metodi HTTP:

- **POST** per Create: usata per la creazione di nuove risorse.
- **GET** per Read: usata per richiedere una risorsa.
- **PUT** per Update: usata per l'aggiornamento di una risorsa.
- **DELETE** per Delete: usata per richiedere l'eliminazione di una risorsa.

3.1.3 Risorse autodescrittive

Le risorse sono concettualmente separate dalle rappresentazioni restituite al client. Da un punto di vista virtuale possiamo utilizzare un qualunque formato noto per la rappresentazione da restituire, ma solitamente si utilizzano formati come JSON o XML in modo da avere una rappresentazione human-readable e Protocol Buffers (ProtoBuf) [24].

Il tipo di rappresentazione usata è inviata al client nella stessa risposta HTTP tramite un tipo MIME e un cliente a sua volta nel momento in cui invia la richiesta per una risorsa può utilizzare l'attributo Accept di una richiesta HTTP per specificare il tipo di rappresentazione desiderata.

3.1.4 Collegamenti tra risorse

Seguendo questo principio, noto anche come Hypermedia As The Engine Of Application State (HATEOAS), le risorse devono essere messe in relazione tra di loro tramite link ipertestuali.

Ad esempio la rappresentazione di un ordine deve contenere eventuali collegamenti agli articoli e al cliente correlati in modo tale che il client può accedere alle risorse correlate semplicemente seguendo i collegamenti contenuti nella rappresentazione della risorsa corrente.

3.1.5 Comunicazione senza stato

Ricalcando una delle caratteristiche principali del protocollo HTTP le interazioni tra client e server devono essere senza stato, ossia ciascuna richiesta non deve aver alcuna relazione con le richieste precedenti e successive.

Questa scelta deriva dal fatto che mantenere uno stato di sessione sul server ha dei costi sul server. In più in questo modo possiamo avere un cluster di servitori del tutto trasparenti al Client, che può decidere di usare server diversi per richieste successive.

3.2 Android

Non molto tempo fa eravamo abituati a pensare ad un telefono cellulare unicamente come un mezzo per poter effettuare chiamate o inviare SMS, ma negli anni la loro diffusione è diventata così ampia da diventare un mezzo di comunicazione indispensabile.

Fortunatamente, grazie al progressivo avanzamento della tecnologia, i programmatori non sono più costretti a conoscere il tipo l'hardware a cui è legato un sistema per poter sviluppare applicazioni software. Questo è dovuto ad un distaccamento dai linguaggi di basso livello come C e C++, a favore di linguaggi di più alto livello come Java.

Vennero introdotte nuove piattaforme, come ad esempio Symbian, per facilitare lo sviluppo di applicazioni. I limiti di questi sistemi, tuttavia, risiedevano nel fatto che richiedevano agli sviluppatori di scrivere del

codice in C/C++ e di far uso di API proprietarie molto spesso difficili da comprendere.

In seguito, con l'aumentare dell'interesse per lo sviluppo di applicazioni mobili, furono introdotte le MIDlets. Questi componenti venivano eseguiti su una macchina virtuale Java (JVM), con una conseguente astrazione dell'hardware specifico di cui era composto il telefono, permettendo lo sviluppo di programmi in grado di essere eseguiti su una larga varietà di dispositivi (tutti quelli che utilizzano la JVM).

Tuttavia, il prezzo che si dovette pagare fu che non era più possibile avere un accesso libero alle risorse hardware. Questo comportò l'impossibilità di sfruttare a pieno i vantaggi conseguenti dagli scenari di mobilità.

Nel 2003, infine, nacque la società Android Inc. rilevata successivamente da Google nel 2005. Il progetto aveva la finalità di sviluppare una piattaforma per dispositivi mobili che fossero "più consapevoli della posizione e delle preferenze del loro proprietario" (Andy Rubin). Il sistema Android fa parte di quella nuova ondata di piattaforme mobili in grado di supportare lo sviluppo applicativo su dispositivi hardware sempre più ricchi e potenti. Piattaforme come Microsoft Windows Phone o Apple iPhone fanno parte della stessa categoria di piattaforme ma, a differenza di Android, sono basate su un sistema operativo proprietario.

Lo stesso Andy Rubin, fondatore di Android Inc. definì la piattaforma come: "The first truly open and comprehensive platform for mobile devices. It include an operating system, user-interface and applications – all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation".

3.2.1 Il kernel Linux

Android è costituito da un Kernel basato sul kernel Linux 2.6 e 3.x (da Android 4.0 in poi), con middleware, Librerie e API scritte in C (oC++) e software in esecuzione su un framework di applicazioni che include librerie Java compatibili con librerie basate su Apache Harmony. Android utilizza la Dalvik virtual machine con un compilatore just-in-time per l'esecuzione di Dalvik dex-code (Dalvik Executable), che di solito viene tradotto da codice bytecode Java La piattaforma hardware principale di Android è l'architettura ARM. L'architettura x86 è supportata grazie al progetto Android x86 e Google TV utilizza una speciale versione x86 di Android.

Il kernel Linux di Android mette a disposizione modifiche all'architettura create da Google al di fuori del ciclo di sviluppo del kernel. Un tipico sistema Android non possiede un X Window System nativo, né supporta il set completo standard di librerie GNU, e nel caso del C++ vi è solo una implementazione parziale delle STL. Tutto ciò rende difficile il porting su Android di applicazioni grafiche o librerie sviluppate per GNU/Linux. Per semplificare lo sviluppo di applicazioni C/C++ sotto Android si usa SDL che tramite un piccolo Wrapper java permette l'utilizzo della JNI dando un'idea di utilizzo simile a un'applicazione nativa in C/C++.

Le applicazioni Android sono Java-based; in effetti le applicazioni scritte in codice nativo in C/C++ devono essere richiamate dal codice java, tutte le chiamate a sistema fatte in C (o C++) devono chiamare codice virtual machine Java di Android: infatti le API multimediali di SDL sotto Android richiamano metodi in Java; questo significa che il codice dell'applicazione C/C++ deve essere inserito all'interno di un progetto Java, il quale produce alla fine un pacchetto Android (APK).

Google ha contribuito ad implementare alcune funzionalità all'interno del kernel Linux, in particolare il wakelock, una funzione che permette alle applicazioni di "risvegliare" il device mentre è in sleep. Quest'ultima feature è però stata rifiutata dagli sviluppatori del kernel mainline, in parte perché hanno ritenuto che Google non abbia manifestato l'intenzione di mantenere il proprio codice. In Linux è stato incluso l'autosleep e le capacità wakelocks nel kernel 3.5, dopo molti precedenti tentativi di fusione. Le interfacce sono le stesse ma la realizzazione di Linux ha due diverse modalità di sospensione: a memoria (le sospensione tradizionale che utilizza Android), e su disco (ibernazione, come è noto sul desktop).

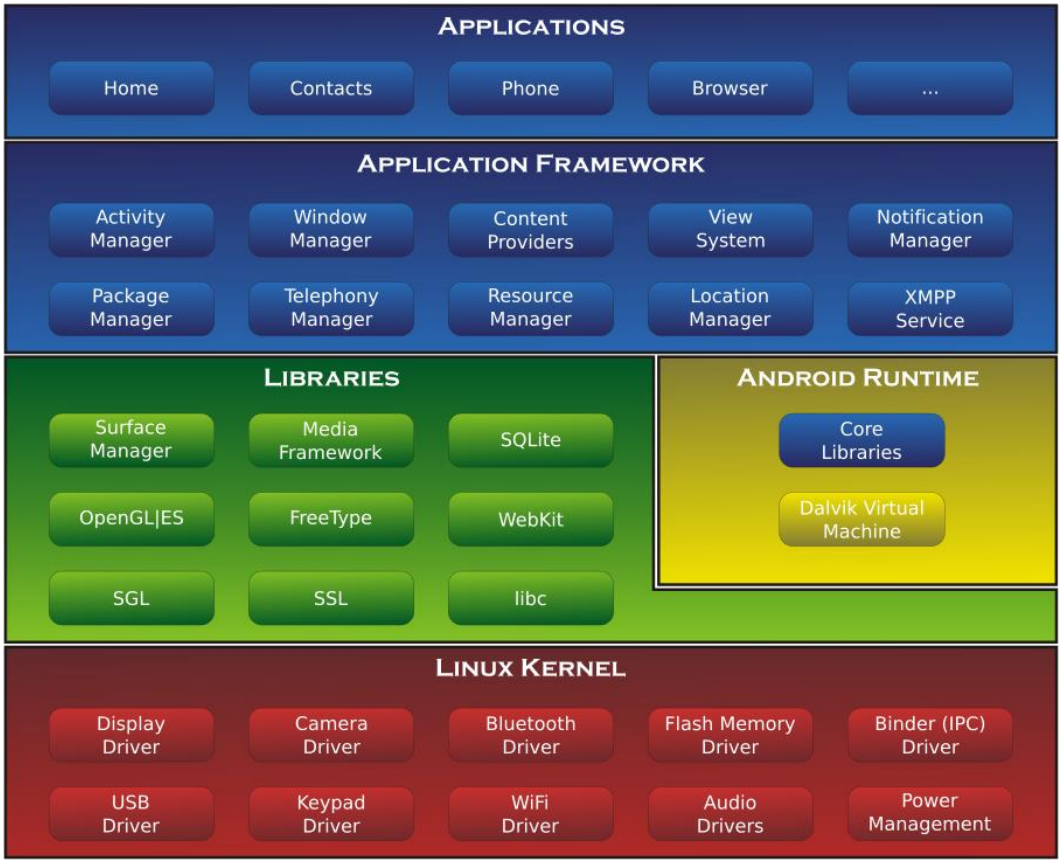

Figura 21: Architettura di Android

3.2.2 Ciclo di vita delle Applicazioni

Le applicazioni, in Android, consistono in componenti fortemente disaccoppiati che comunicano tra loro attraverso il paradigma a scambio di messaggi. I blocchi che compongono un'applicazione Android sono descritti attraverso un file chiamato Manifest, utilizzato anche per specificare alcuni metadati ed informazioni specifiche dell'applicazione.

I componenti fondamentali che permettono la creazione di un'applicazione in Android sono:

- Activities
- Services
- Content Providers
- Intents
- Broadcast Receivers

A differenza di molte piattaforme Android non permette un controllo, da parte dello sviluppatore, completo sul ciclo di vita delle applicazioni. I componenti applicativi devono essere in ascolto dei cambiamenti dello stato applicativo e reagire di conseguenza. Solitamente, ogni applicazione esegue in un proprio processo; ognuno dei quali è in esecuzione su un'istanza specifica di Dalvik VM. Il sistema Android gestisce in modo molto aggressivo le risorse facendo tutto il necessario affinché venga sempre garantita una stabile user-experience; in concreto, questo sta a significare che i processi (e le applicazioni che ospitano) possono essere terminate dal supporto per liberare risorse necessarie ad altre applicazioni più prioritarie.

Nel dettaglio, l'ordine secondo cui i processi vengono terminati è determinato dalla priorità delle applicazioni che ospitano. La priorità di un'applicazione è uguale a quella più alta tra i componenti che la

compongono. Se due applicazioni hanno la stessa priorità, allora il processo che ha avuto la priorità per più tempo viene terminato.

La priorità di un processo è anche influenzata dalle dipendenze tra i processi; ad esempio, se un'applicazione dipende da un Service o un Content Provider forniti da una seconda applicazione allora quest'ultima ha almeno lo stesso livello di priorità dell'applicazione che supporta.

3.2.3 Activity Stack

Le activity sono quelle applicazioni destinate a una interazione diretta con l'utente. Un esempio sono i videogiochi, le applicazioni per l'ufficio e i visualizzatori (reader) di E-book. Le activity vengono generalmente distribuite sotto forma di file .apk, vengono poi installate in diverse cartelle nella memoria del dispositivo (o in una card estraibile), infine viene creata una icona per l'utente, che gli permetterà di eseguirla in qualsiasi momento. È anche possibile disinstallare le activity mediante una utility integrata con Android. Le activity vengono create come oggetti di classe Activity da cui ereditano proprietà e metodi.

L'Activity ha un proprio ciclo di vita che viene gestito dal container Android in accordo al ciclo di vita applicativo che è stato mostrato in precedenza. Sebbene il runtime di Android ne controlli la terminazione e la gestione del ciclo di vita; attraverso il suo stato è possibile ricavare il livello di priorità dell'applicazione. Lo stato è determinato dalla posizione che essa occupa all'interno dell'Activity Stack, una collezione di Activity gestite secondo politica last-in-first-out. In Figura 22 viene mostrato il processo di creazione di una nuova Activity.

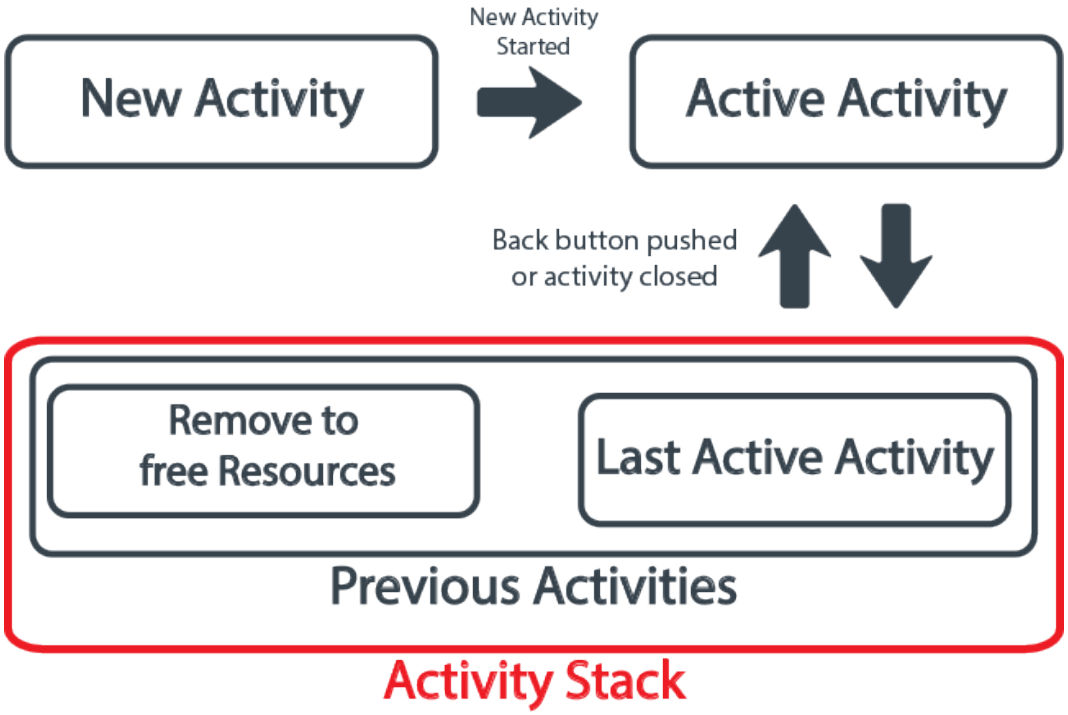

Figura 22: Processo di creazione di un Activity

Gli stati che caratterizzano il ciclo di vita di un'Activity sono quattro:

- **active:** quando è in cima allo stack. Android cerca di mantenere in vita questa Activity ad ogni costo garantendogli tutte le risorse di cui ha bisogno;
- **paused:** quando è visibile ma non ha il focus. In questo stato il componente viene considerato dal run-time di Android come se fosse attivo anche se non riceve input dall'utente. In casi estremi il sistema può terminare delle Activity in stato paused per permettere l'allocazione di risorse a quelle active.
- **stopped:** quando non è visibile. Il componente rimane in memoriaconservando tutto il suo stato applicativo ma diventa un candidato per essere terminato.
- **inactive:** dopo che viene terminata e prima che venga lanciata.

Le transizioni di stato sono un processo non deterministico e sono interamente governate dal gestore di memoria di Android.

3.2.4 Fragment

I fragments nel tempo si sono meritati un posto di spicco nel mondo della **programmazione Android** tanto da essere inseriti come parte integrante del framework nella versione 3.0.

La progettazione di un'interfaccia utente incentrata su questo meccanismo ha come primo obiettivo la modularità. Un fragment infatti occupa una **porzione di Activity**, ma non svolge il ruolo di semplice "raggruppamento di View", bensì appare come una sotto-applicazione dotata un suo ciclo di vita.

Ciò lo rende potenzialmente separabile dagli altri fragments della stessa interfaccia o riutilizzabile per conto proprio nonchè congiuntamente a nuove componenti.

Figura 23: Esempio di Fragments

Nella Figura 23 possiamo notare sulla sinistra una "pulsantiera" fatta di diversi "<Button>" ognuno dei quali riporta il nome di una grande metropoli, sulla destra invece appare una scheda di dettaglio. Come

presumibile, al click di un pulsante, la scheda sulla destra viene aggiornata con informazioni relative alla città prescelta.

A livello di layout, le due sezioni dell'Activity vengono rappresentate con dei **fragments**, rispettivamente detti LeftFragment e CentralFragment come evidenziato in figura dai rettangoli rossi. L'esempio non è dei più elaborati, ma mostra comunque tutti gli elementi fondamentali dell'utilizzo dei fragments: ciclo di vita, modularità e collaborazione reciproca.

Uno dei primi nodi da sciogliere nell'apprendimento dei fragments è proprio il **rapporto che intercorre tra essi e le classiche Activity**.

3.2.5 Fragment e Activity: rapporti e differenze

Un'interfaccia utente che mostra la presenza di più fragments nel layout continua pur sempre ad **avere una sola Activity** che svolge il ruolo di **contenitore**, cabina di regia della comunicazione tra fragments.

Un fragment, in quanto parte di un'Activity, non può avere un ciclo di vita separato da essa ma deve, in un certo senso, condividerne le sorti. Come sappiamo, le Activity attraversano vari stati: ACTIVE, PAUSED,STOPPED e INACTIVE.

Parallelamente, anche i fragments passano da uno stato all'altro ed ogni transazione significativa, viene notificata tramite eventi che il programmatore ha la possibilità di gestire mediante override dei metodi di callback.

Durante l'attivazione di un'Activity, i fragments vivono varie fasi denotate, tra gli altri, dai seguenti metodi:

- **onAttach(Activity a):** avviene il collegamento funzionale tra il fragment e l'activity che lo possiede. È il momento in cui il fragment può salvare il riferimento all'Activity passato come parametro formale. Ciò al fine di invocarne metodi come nel caso della comunicazione con gli altri fragments;
- **onCreateView:** si sta creando l'aspetto grafico del fragment. Avvengono qui tutte le operazioni di configurazione delle varie View e dei listener per la gestione degli eventi: praticamente ciò che si fa normalmente nel metodo onCreate delle Activity;
- **onActivityCreated:** il fragment viene notiziato del completamento dell'Activity. Da ora in poi è possibile invocarla tramite i metodi.

Per il resto il fragment segue il destino dell'Activity che lo contiene sia che essa stia transitando nello stato di PAUSE, di STOP o si stia procedendo alla sua distruzione.

3.2.6 Intents, Intent Filters e Broadcast Receiver

L'Intent è probabilmente uno dei concetti più importanti del sistema Android; infatti premettono l'interazione tra componenti (anche di applicazioni differenti) tramite uno stile di comunicazione message-passing. Questo permette di trasformare il dispositivo da una piattaforma formata da componenti indipendenti tra loro a un sistema interconnesso.

Uno dei casi d'uso più frequenti è certamente quello di far partire una nuova Activity; che può essere fatto in maniera esplicita (indicando il nome del componente) o implicita (richiedendo che venga eseguita una determinata azione per un certo insieme di dati). Inoltre, gli

Intents possono essere utilizzati per effettuare il broadcast di messaggi all'interno del sistema. L'applicazione, infatti, può registrare dei Broadcast Receiver che siano in ascolto di questi messaggi reagendo di conseguenza. In questo modo è possibile creare delle applicazioni event-driven basate su eventi interni, di sistema, o di applicazioni esterne.

L'utilizzo degli Intents, in Android, è un principio di design fondamentale perché permettono il disaccoppiamento massimo tra i componenti permettendo il riutilizzo di quest'ultimi.

Come detto in precedenza, nel caso in cui si voglia far partire un nuovo componente, si può utilizzare una modalità esplicita o implicita. Nel secondo caso, sicuramente di maggior interesse, è previsto che sia direttamente il sistema Android a trovare il miglior componente da istanziare.

Gli Intent possono anche essere utilizzati se si vuole fare un broadcast di messaggi anonimi tra componenti. Attraverso l'utilizzo di Broadcast Receiver è possibile ascoltare e reagire a determinati eventi di interesse. I Broadcast Intents sono usati per notificare le applicazioni di eventi applicativi o di sistema abilitando il paradigma di interazione event-driven tra componenti. In questo modo è possibile sviluppare delle applicazioni più aperte perché, producendo eventi, si abilita l'interazione con applicazioni esterne. Inoltre, registrando dei Broadcast Receiver, l'applicazione può reagire ad eventi di sistema che tipicamente la piattaforma Android produce come ad esempio eventi di disconnessione o connessione alla rete.

3.2.7 Services

I Services sono dei componenti Android pensati per effettuare delle operazioni di lunga durata e gestire operazioni che non necessitano dell'interfaccia utente. Questi possono venir controllati da altri componenti applicativi e possono esser fatti partire, messi in pausa o stoppati. Quando sono in esecuzione hanno una priorità più alta rispetto alle Activity che sono nello stato stopped, in modo da ridurre la probabilità di terminare la loro esecuzione da parte del sistema. L'unica ragione per cui Android può terminare un Service è perché ha bisogno di fornire delle risorse ai dei componenti che sono in foreground. Quando accade ciò è comunque possibile configurare il Service affinché venga rilanciato automaticamente.

Anche se sono stati pensati per essere eseguiti in assenza di User Interface, è necessario ricordare che i Service eseguono all'interno del main thread applicativo.

Questo sta a significare che una corretta gestione è a carico dello sviluppatore, che deve evitare di bloccare il thread principale e di conseguenza la UI.

3.3 Spring

Per la realizzazione del server di ParticipAct è stato utilizzato Spring [25], un framework Java per applicazioni enterprise che fornisce un'infrastruttura di supporto permettendo agli sviluppatori di concentrarsi sulla logica di business. Gli sviluppatori possono quindi scrivere dei Plain Old Java Object (POJO) avendo già tutte le funzionalità orizzontali implementate dal framework.

Uno degli aspetti che ha caratterizzato Spring fin dalle prime release è l'Inversion of Control (IoC) o Dependency Injection, principio secondo il quale è il container a gestire il ciclo di vita dei componenti.

È quindi il container che si occupa di soddisfare tutte le dipendenze prima di instanziare un'oggetto, lasciando come unica responsabilità allo sviluppatore quella di definire le dipendenze di un oggetto, senza preoccuparsi di risolverle.

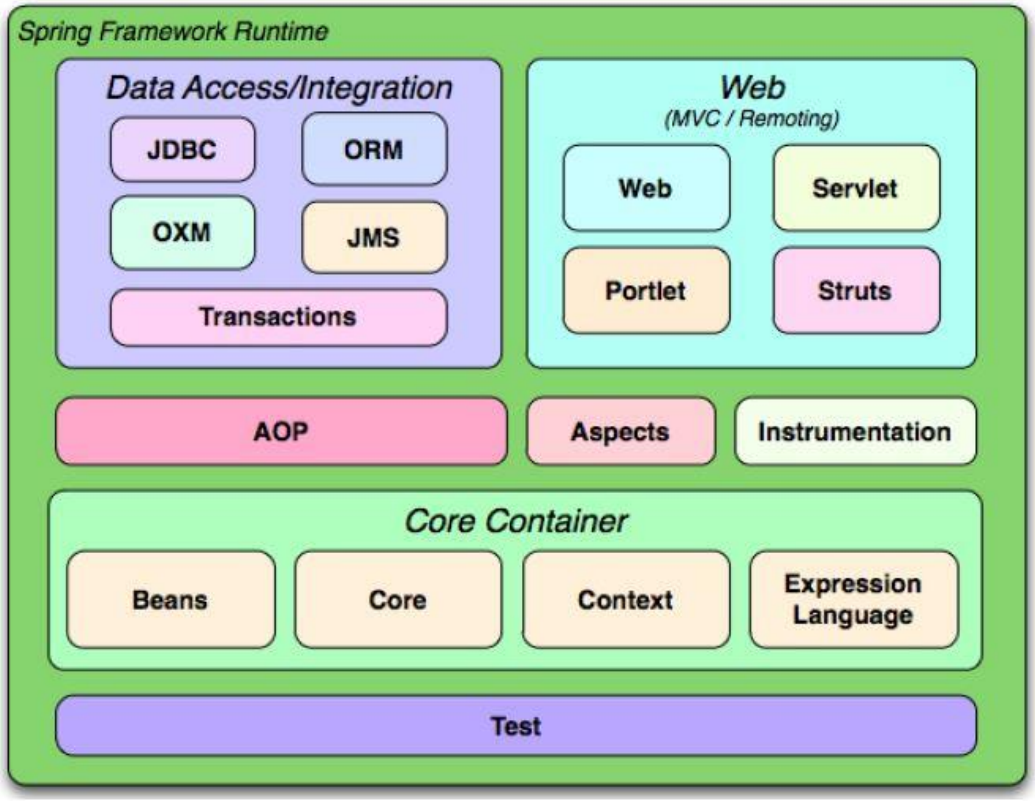

Figura 24: Architettura del framework Spring

Nel 2000 Rod Johnson inizia lo sviluppo di Spring mentre lavora a Londra come consulente libero professionista, ma è durante la scrittura di "Expert One-on-One Java EE Design and Development" che espande il proprio codice al fine di esprimere la sua visione di semplificare e rendere più consistente il modo di inter operare delle varie parti che compongono una applicazione Java EE.

Prima del 2001 i modelli di programmazione dominanti per le applicazioni web erano basati sulle Java Servlet API e gli Enterprise JavaBeans. Entrambe le specifiche furono create da Sun

Microsystems in collaborazione con altri vendor e godevano di grande popolarità all'interno della comunità Java. Le applicazioni che non erano basate sul web si appoggiavano ad altri strumenti e paradigmi di programmazione. Rod Johnson ha avuto il merito di creare un framework basato su principi ottimali largamente accettati e lo ha reso disponibile per tutti i tipi di applicazione, non solo web.

Nel febbraio 2003 un piccolo gruppo di sviluppatori ha creato un progetto su Sourceforge al fine di estendere il framework e dopo circa un anno ha rilasciato la prima versione 1.0. Sebbene sia stato largamente adottato, Spring venne pesantemente criticato per il fatto che il progetto si poneva come obiettivo la semplice integrazione con gli standard Java EE senza un documento di specifica controllato da un comitato ufficiale.

Spring rese popolari alcune tecniche prima d'allora poco note come l'Inversion of Control e il paradigma di Programmazione orientata agli aspetti. Il 2005 ha visto un enorme aumento di consensi in concomitanza di un nuovo importante rilascio. Inoltre il forum ufficiale ha notevolmente aiutato ad accrescere la popolarità del framework e si è imposto quale fonte primaria di informazione e di supporto agli utenti.

3.3.1 IoC container

L'IoC container è la parte di Spring che si occupa di instanziare e configurare gli oggetti che vengono inseriti in esso, che prendono il nome di beans.

I beans vengono configurati attraverso dei metadati che possono essere dei file xml o delle Java annotations. Di default i metadati vengono letti solo dagli xml, per abilitare l'uso delle annotations c'è bisogno di configurare l'ApplicationContext.

La parte di IoC e di DI viene implementata attraverso la BeanFactory e l'ApplicationContext. Dato che l'ApplicationContext è un superset della BeanFactory, se ne consiglia l'uso, e da qui in avanti faremo riferimento solo all'ApplicationContext.
Ci sono vari tipi di ApplicationContext forniti da Spring, a seconda dell'applicazione che si deve sviluppare. Per esempio per applicazioni stand alone ci sono il ClassPathXmlApplicationContext e il FileSystemXmlApplicationContext, mentre per le applicazioni enterprise c'è il WebApplicationContext, che viene instanziato attraverso un serlvet listener.
Tutti gli ApplicationContext hanno bisogno dei metadati di configurazione e quindi per instanziarne uno abbiamo bisogno prima di scrivere un file xml.

I vantaggi nell'usare Spring saltano subito all'occhio:

- I beans della nostra applicazione non implementano nessuna interfaccia o classe astratta e le dipendenze dall'IoC container sono nulle.
- Se si decide di cambiare un'implementazione o di aggiungerne delle altre, basta cambiare il file xml, senza toccare il codice, questo grazie all'uso delle interfacce.
- Sempre grazie all'uso interfacce è facile creare dei test per la nostra applicazione.

3.3.2 Data access/integration

Del modulo Data Access/Integration è stato utilizzata la componente ORM che fornisce un layer per le API di object-relational mapping. In particolare è stato utilizzato Java Persistence Api (JPA) [26] con Hibernate [27] e PostgreSQL [28].

Le uniche responsabilità lasciate allo sviluppatore sono definire una data source e mappare le entity e le relative proprietà usando delle

annotazioni. Poi Spring provvederà a configurare l'EntityManager e ad iniettarlo ove richiesto.

3.3.3 Web

Di questo modulo si è utilizzato Spring MVC sia per gestire l'interfaccia web per l'amministratore che per l'interfacciamento REST con i client. È stato inoltre usato a n c h e Spring Web Flow per creare dei flussi di lavoro eseguibili dagli amministratori, ad esempio per la creazione ed assegnamento di un Task o per la creazione di un badge.

Il nucleo di Spring MVC è una servlet di front end chiamata DispatcherServlet (o Front controller in Figura 25) che intercetta tutte le richieste web e utilizzando l'url della richiesta individua il controller che gestirà la richiesta consultando un Handler Mapping. Il controller è un semplice POJO annotato con l'annotazione @Controller e uno o più metodi annotati con @RequestMapping ognuno dei quali scritto per una richiesta differente. Solitamente le richieste che arrivano dai client prevedono l'invio una risposta e il controller è il responsabile per reperire i dati desiderati, ma questi dati grezzi non sono sufficienti per qui il controller specificherà anche un nome logico di una view che verrà utilizzato sempre dalla DispatcherServlet per effettuare un look-up presso un View Resolver ottenendo così la view concreta per il rendering dei dati specificati dal controller.

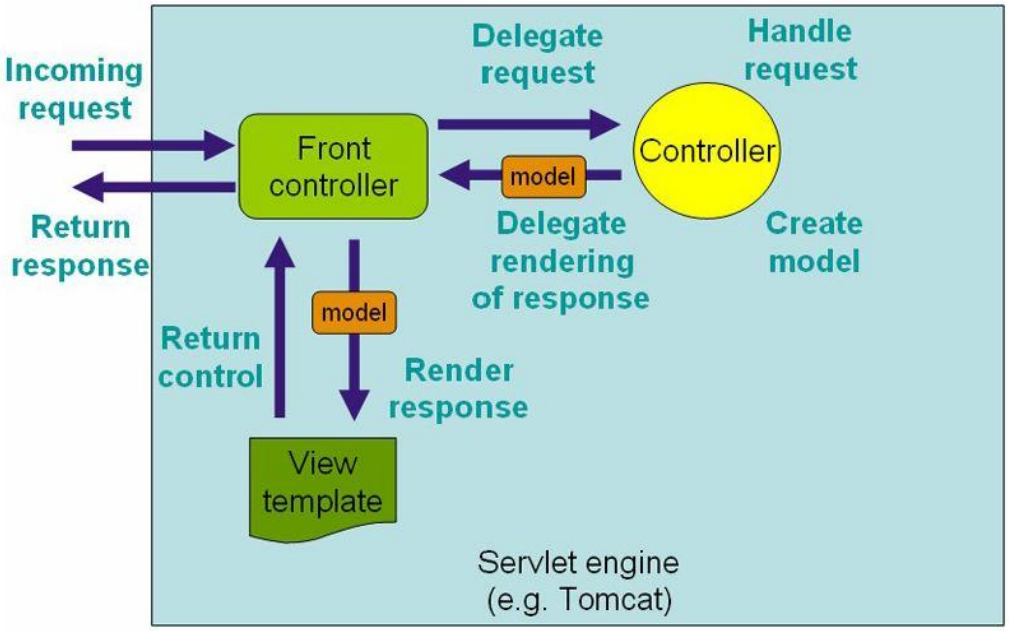

Figura 25: Funzionamento di Spring MVC

Il supporto per REST in Spring è bastato su Spring MVC al quale sono stati aggiunte alcune funzionalità tra cui:

- Annotazione @PathVariable: permette la gestione di url parametrizzati.
- Annotazione @ResponseBody: permette al controller di restituire direttamente i dati.
- Conversione automatica del model in: XML, JSON, Atom e RSS.
- Annotazione @RequestBody: per estratte dati in automatico dal body della request.

Spring Web Flow è un'estensione di Spring MVC che permette di definire dei flussi di lavoro intesi come una sequenza di passi che possono essere eseguiti in contesti diversi. In Spring Web Flow un flusso è composto da diversi stati e tipicamente nell'ingresso in uno stato viene mostrata una view all'utente. In questa view l'utente può

scatenare una serie di eventi, tra cui un evento di transizione in uno nuovo stato.

3.3.4 L'uso di MVC

Per comprendere al meglio il framework è necessario introdurre il pattern teorico che esso implementa ovvero il modello MVC. MVC rappresenta un acronimo per Model View Controller ovvero le tre componenti principali di un'applicazione web. Grazie a questo pattern i compiti vengono separati verso questi componenti:

- i Model (in italiano Modelli) si occupano di accedere ai dati necessari alla logica di business implementata nell'applicazione (nel caso di un'applicazione per una biblioteca potrebbero essere le classi Libro, Autore, Scaffale);
- le View (in italiano Viste) si occupano di creare l'interfaccia utilizzabile dall'utente e che espone i dati da esso richiesti (nel caso bibliotecario potrebbero essere le pagine HTML del catalogo, le form di ricerca oppure i PDF contenenti ricerche o appunti);
- i Controller (in italiano Controllori) si occupano di implementare la vera logica di business dell'applicazione integrando le due componenti precedenti, ricevendo gli input dell'utente, gestendo i modelli per la ricerca dei dati e la creazione di viste da restituire all'utente (nel nostro caso potrebbero essere il motore di ricerca interno oppure il sistema di login al sito Internet della biblioteca).

Spring MVC implementa perfettamente questo approccio mantenendo i concetti della nomenclatura del pattern. All'interno di una applicazione Spring MVC avremo quindi:

- i Model sono rappresentati dalle classi che a loro volta rappresentano gli oggetti gestiti e le classi di accesso al database;
- le View sono rappresentate dai vari file JSP (che vengono compilati in HTML) e da eventuali classi per l'esportazione in formati diversi da HTML (PDF, XLS, CSV…);
- i Controller sono rappresentati da classi (chiamate appositamante Controller) che rimangono "in ascolto" su un determinato URL e, grazie ai Model e alle View, si occupano di gestire la richiesta dell'utente.
- Secondo la documentazione ufficiale Spring MVC presenta inoltre molti altri vantaggi oltre alla netta separazione tra le funzionalità:
- è adattabile, flessibile e non intrusivo grazie alla presenza di comode e chiare Java Annotations;
- permette di scrivere codice riusabile;
- possibilità di essere esteso tramite adattatori e validatori scritti ad hoc per le nostre esigenze;
- url dinamici, SEO-friendly e personalizzabili;
- gestione integrata dell'internazionalizzazione e dei temi;
- libreria JSP sviluppata ad hoc per facilitare alcune operazioni ripetitive;
- nuovi scope per i bean (request e session) che permettono di adattare i container base di Spring anche al mondo web.

Capitolo 4: Il progetto ParticipAct

In questo capitolo verranno forniti i dettagli implementativi dei componenti della piattaforma ParticipAct alla luce delle scelte tecnologiche discusse nel capitolo precedente. I componenti che verranno discussi sono ancora privi della logica per la creazione dei gruppi e dell'estensione della gamification ai gruppi. Di queste ulteriori modifiche ne parleremo a partire dal prossimo capitolo.

Si inizierà con i dettagli del server centrale analizzandolo a partire dal layer più alto, che è la presentazione, scendendo lo stack fino ad arrivare alla persistenza. Terminata la sezione dedicata al server si passerà ai componenti mobili della piattaforma.

Verranno discussi sia il client ParticipAct che si interfaccia con il server per la gestione dei Task, sia il framework MoST che è il vero responsabile delle azioni di sensing.

4.1 Server

Il server del progetto ParticipAct è stato realizzato attraverso il framework Spring precedentemente descritto, organizzando i componenti in un'architettura multi layer (vedi Figura 26).

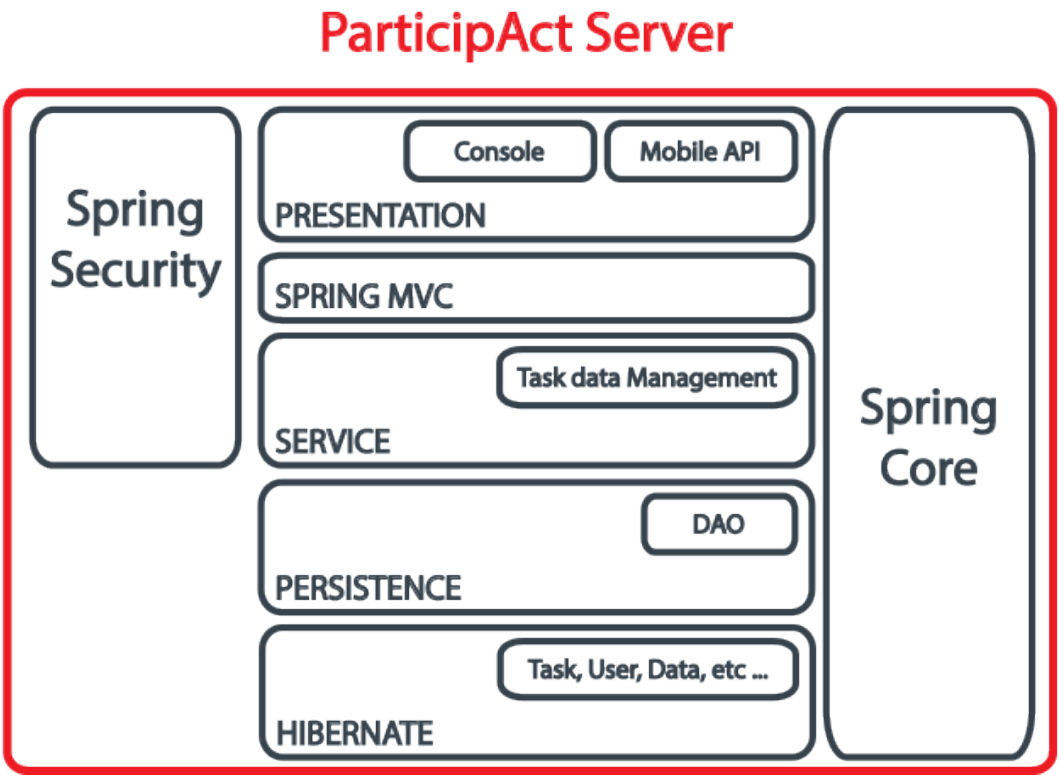

Figura 26: Server ParticipAct

Questa architettura permette ad ogni livello di utilizzare i servizi offerti dal livello sottostante attraverso un accoppiamento debole. Tipicamente infatti i componenti di un livello esprimono una dipendenza da un componente di un layer sottostante attraverso un interfaccia. Sarà poi responsabilità di Spring Core reperire i componenti concreti che realizzano le funzionalità desiderate e renderli reperibili a chi ne fa richiesta. Questa organizzazione permette di effettuare modifiche layer per layer senza impattare sui componenti degli altri livelli, garantendo dei componenti facilmente manutenibili ed intercambiabili.

In cima allo stack troviamo il livello di presentation utilizzato per fornire agli amministratori della piattaforma uno strumento di monitoraggio e gestione chiamato console. Tale strumento è realizzato attraverso l'utilizzo di JavaServer Pages (JSP).

Sempre in questo livello sono presenti le Mobile API REST invocabili dal client che può disinteressarsi dell'effettivo reperimento delle informazioni richieste.

Immediatamente sotto al livello di presentation è presente il livello di Spring MVC che serve ad orchestrare il layer superiore. In questo livello sono presenti infatti i controller Spring utilizzati per popolare e recuperare dati dalle JSP e i controller che mappano le API fornite al client in metodi dei componenti contenenti la logica di business del server.

Il livello contenente la logica di business è il livello service. I componenti di questo livello utilizzano i dati forniti dai layer sottostanti all'interno di transizioni gestite da Spring per effettuare elaborazioni che vengono esposte ai livelli superiori.

Nella parte finale dello stack troviamo il layer di persistenza e quello di Hibernate che vengono utilizzati per la gestione dei dati presenti nel database seguendo il pattern repository e sfruttando la tecnologia Object-Relational Mapping (ORM) fornita dal framework Hibernate.

Spring Security è usato dai layer superiori dello stack per garantire l'autenticazione e l'autorizzazione nell'accesso alle risorse.

4.1.1 Presentation

Al livello di presentation sono presenti le view utilizzate dagli amministratori scritte utilizzando la tecnologia JSP e l'end-point usato dall'applicazione mobile per comunicare con il server che utilizza la rappresentazione JSON o ProtoBuf in base alla richiesta pervenuta (vedi Figura 27).

Figura 27: Livello di presentazione del server

La Console è divisa in diversi moduli:

- **Dashboard:** permette all'amministratore di effettuare alcune operazioni di gestione, come la consultazione dei log ricevuti dai client e l'invio di notifiche push ai dispositivi mobili.
- **Account Management:** utilizzato per gestire sia gli utenti di ParticipAct che gli amministratori.
- **Task Management:** componente responsabile della creazione di Task, dell'assegnamento agli utenti e visualizzazione dei dati ottenuti.

Le Mobile API sono suddivise in:

- **Task Sync:** utilizzate dai client mobile per aggiornare sul server lo stato dei Task.
- **Data Receiver:** invocate per effettuare l'upload dei dati raccolti dagli smartphone.
- **News Provider:** forniscono al client le novità sul progetto per essere mostrate ai partecipanti.

4.1.2 Model View Controller

Subito sotto c'è il livello di Spring MVC utilizzato per fare il dispatch di tutte le richieste ricevute sia dall'applicazione web che dai client mobile. Si possono suddividere due macro categorie di richieste: quelle provenienti dalla web app per gli amministratori, e quelle REST provenienti dall'applicazione mobile (Figura 28). In questo livello sono definiti anche i form, con i relativi pattern di validazione, utilizzabili dalle view per l'immissione di dati.

Figura 28: Livello MVC del server

Le richieste provenienti dalla web app per amministratori sono gestite da una classe di controller chiamati web controller, mentre le richieste provenienti dai client mobile sono gestiti dai rest controller.

Un semplice esempio di controller web che non prevede l'elaborazione di input inseriti dagli utenti è il controller che è mappato nell'url *protected/dashboard* qui riportato:

```
@RequestMapping(value = "/protected/dashboard", method = RequestMethod.GET) (1)
        @PreAuthorize("isFullyAuthenticated()")                              (2)
        public ModelAndView home(ModelAndView mav, HttpSession session) {
        mav.setViewName("protected/dashboard");                              (3)
        return mav;
}
```

Attraverso le annotazioni offerte da Spring è specificato l'url e il metodo HTTP della richiesta (1) che gli amministratori invieranno quando vogliono visualizzare il pannello di controllo principale della dashboard. È anche specificato il ruolo che il richiedente (specificato nell'header HTTP della richiesta) deve avere per poter effettuare la richiesta (2), nell'esempio questo controller risponderà solamente se l'utente che ne fa richiesta è un amministratore. Per effettuare questo controllo è stata utilizzata una Spring Expression Language (SpEL) [29] che restituisce il valore true se l'utente che fa richiesta è correttamente autenticato. Se l'utente è correntemente autenticato viene eseguito il metodo che specifica il nome logico della view che deve essere restituita all'utente (3).

Altri controller web degni di nota a sono:

- **AccountController:** per la visualizzazione e la modifica dei dati degli utenti iscritti al progetto.
- **TaskAddController:** gestisce il flusso di creazione di un Task recuperando e validando i dati che gli amministratori inseriscono e notificando gli utenti ai quali il task è stato assegnato.
- **TaskManageController:** recupera dai layer sottostanti i dati raccolti dagli utenti per i Task, li elabora e prepara per essere visualizzati in forma aggregata dagli amministratori.

I controller rest differiscono dai controller web perché restituiscono direttamente una rappresentazione della risorsa richiesta, senza popolare una view.

Un esempio è il controller TaskController che permette ai client di richiedere i Task disponibili:

```
@RequestMapping(value = "/rest/taskflat", method = RequestMethod.GET)          (1)
        public @ResponseBody TaskFlatList getTaskFlat(ModelAndView modelAndView,
        Principal principal, @RequestParam("state") String state{
        //Recupero dai layer sottostanti dei Task
        //con lo stato richiesto per l'utente che
        //ne ha fatto richiesta
        TaskFlatList result = findTaskForUserAndStatus(principal.getName,
        state);
        return result;                                                          (2)
}
```

Questo metodo gestirà tutte le richieste all'indirizzo */rest/taskflat?state=query* (1) e restituirà direttamente (2) una rappresentazione in JSON dei Task con stato uguale a query per l'utente che ha inviato la richiesta.

Altri controller rest presenti sono:

- **ResultDataController:** permette di inviare richieste mettendo nel body http una rappresentazione ProtoBuf dei dati raccolti. Il controller estrarrà i dati contenuti nella richiesta e dopo averli validati li inserirà nel database.
- **TwitterController:** invocato dai client per ottenere news sul progetto.

4.1.3 Service

La logica di business è scritta nel layer di service. Un service è un componente legato alla logica di business che ha il duplice compito di definire le transazioni e di offrire ai livelli superiori un business service.

Questo layer è organizzato in moduli e ognuno di essi espone un'interfaccia e fornisce un'implementazione concreta. Questa scelta dipende dalla volontà di voler sfruttare al meglio uno dei principi fondamentali di Spring: la dependency injection, che permette ad un componente di esprimere una dipendenza nei confronti di un'interfaccia e delegare a Spring core il reperimento del componente che implementa tale interfaccia. Attraverso questa tecnica i layer superiori restano indipendenti alle implementazioni e continuano a funzionare anche nel caso di cambiamenti di queste ultime.

Nel progetto ParticipAct alcuni Service sono degni di nota:

- **Account:** permette di trovare, modificare e cancellare gli utenti presenti in ParticipAct.
- **Data:** responsabile dell'elaborazione di tutti i dati raccolti dagli smartphone. Dopo una pulizia e integrazione li memorizza nel database.
- **Task:** utilizzato per creare nuovi Task e assegnarlo agli utenti.
- **Task Report:** permette agli amministratori di ottenere e visualizzare attraverso la console i dati raccolti dagli utenti per ogni Task.

4.1.4 Gestione del Layer Persistence

Per l'accesso ai dati si è scelto di implementare all'interno del layer persistence il pattern repository.

È importante ricordare che l'ultimo livello presente è Hibernate che viene utilizzato per la persistenza dei dati all'interno di un database.

4.1.5 Protezione e Sicurezza dei dati sensibili

Per non esporre dati sensibili si è scelto di utilizzare Spring Security nei layer superiori dell'architettura. Per l'autenticazione si è scelto di usare HTTP Basic [30] gestendo le autorizzazioni per ogni richiesta web o metodo invocato tramite ruoli differenti per gli utenti e gli amministratori di ParticipAct.

4.2 Client

Per la realizzazione dell'app mobile è stata scelta un'architettura a livelli e si è preferito sviluppare l'app per sistemi Android. La Figura 29 mostra la stratificazione dell'applicazione.

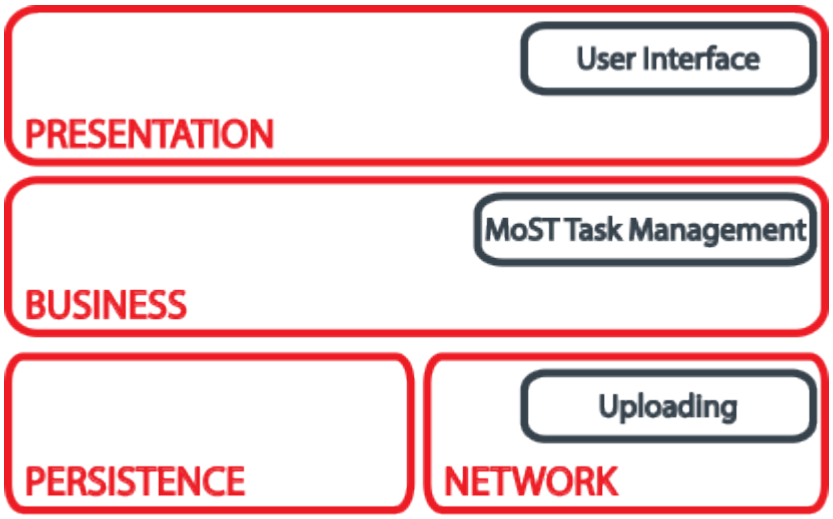

Figura 29: Architettura del Client

La scelta di organizzare l'architettura del Client ParticipAct a livelli è dovuta alla volontà di avere componenti con responsabilità ben definite e riutilizzabili al meglio.

Il livello di **presentation** è responsabile di gestire l'interazione con gli utenti, mostrando un interfaccia grafica che permette di accettare/rifiutare nuovi Task oltre a gestire quelli attualmente in esecuzione.

I dati da visualizzare vengono ottenuti dal livello di **business** che ha il compito di gestire il ciclo di vita di tutti i Task. Per fare ciò utilizzerà

un framework esterno chiamato MoST attraverso una comunicazione a scambio di messaggi basato su intent.

Per salvare i dati in un **database** privato viene utilizzato un approccio ORM che permette agli sviluppatori di disinteressarsi della mappatura degli oggetti Java in tabelle SQL ed ottenere questo mapping in maniera automatica.

Il modulo di **network** realizza un client REST per le API fornite dal server ParticipAct e fornisce una serie di servizi aggiuntivi come il marshalling ed unmarshalling per la rappresentazione necessaria alla comunicazione con il server, la gestione della cache e degli errori HTTP.

4.2.1 Layer Presentation

Per gestire la comunicazioni con gli utenti il layer di Presentation è stato costruito attraverso l'utilizzo di activities e fragments.

All'avvio dell'applicazione viene caricata l'activity principale: la DashboardActivity che modifica la propria user interface quando l'utente naviga il menù laterale.

L'activity principale che viene lanciata all'avvio dell'applicazione presenta un menù laterale che permette all'utente di navigare tra le diverse sezioni, una Toolbar superiore che fornisce informazioni sulla sezione attualmente visualizzata e un corpo centrale che cambia il proprio stato durante la navigazione dell'utente caricando il fragment richiesto dagli utenti.

Ogni volta che si vuole cambiare il fragment visualizzato viene prima di tutto pulita la navigazione, poi recuperato il Fragment richiesto dall'utente e visualizzato.

Grazie all'accoppiamento debole fornito dalle interfacce Java il Fragment non dipende direttamente da DashboardActivity e continuerà a funzionare senza alcuna modifica necessaria all'interno di tutte le activity che implementano l'interfaccia FragmentSwitcher.

Altre activity presenti nell'applicazione servono ad assistere l'utente durante una raccolta di dati che prevedono un'interazione attiva come QuestionnaireActivity. Queste activity vengono avviate con il metodo startActivity() quando l'utente decide di svolgere un Task.

4.2.2 Layer Business

Questo layer è il responsabile della gestione del ciclo di vita dei Task. Il service TaskService comunica con il framework MoST (più precisamente con un suo servizio chiamato MoSTService) attraverso un meccanismo a scambio di messaggi basato sugli Intent, richiedendo l'avvio e la sospensione delle Pipeline di interesse per i Task che devono essere eseguiti. Inoltre questo service comunica costantemente con MoST attraverso un segnale di ping che serve a tenere l'applicazione ParticipAct aggiornata sulle Pipeline attualmente attive e a segnalare possibili malfunzionamenti.

La comunicazione tra i due services avviene anche al momento del completamento dell'attività di sensing, quando ParticipAct richiede a MoST i dati raccolti per poterli caricare nel server, e una volta ottenuta conferma del caricamento richiede a MoST di eliminare tali dati per liberare risorse sul dispositivo.

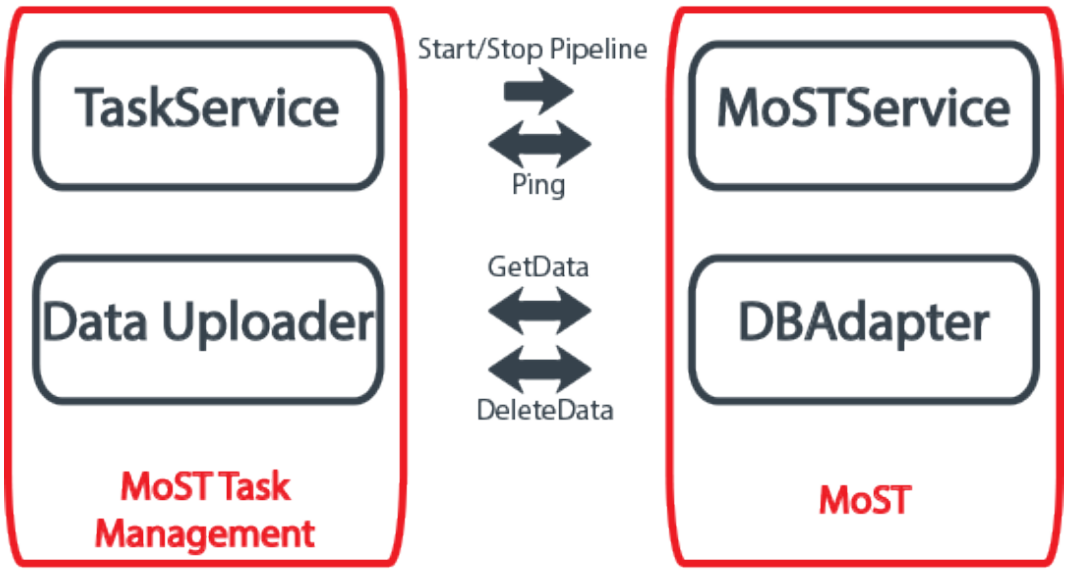

Figura 30: Struttura del layer di Business

4.2.3 Layer Network

Il layer di comunicazione è responsabile di tenere sincronizzato lo stato del Task tra il server centrale e l'applicazione mobile, notificando le decisioni dell'utente, le sue interazioni e i dati raccolti.

L'applicazione utilizza RoboSpice [31], un framework per il networking in Android che fornisce supporto per REST e cache con API asincrone scritte per essere usate nel ciclo di vita dei componenti Android.

Il framework permette allo sviluppatore di creare una Request specificando l'URL ed eventuali parametri da inviare al server, poi questa richiesta verrà gestita dal framework che si preoccupa del marshalling e dell'unmarshalling dei valori, della gestione dei thread, della cache e di restituire il risultato nel main thread dell'applicazione.

4.2.4 Layer Persistence

Il livello di persistenza è responsabile della gestione ottimale e sicura della memorizzazione dei dati raccolti dall'attività di sensing che vengono salvati in un database privato SQLite.

L'applicazione fa uso di OrmLite che fornisce alcuni classi helper come OrmLiteSqliteOpenHelper utilizzata per la creazione del database e l'aggiornamento a versioni successive. Per mappare il model in tabelle del database sono fornite una serie di annotazioni che permettono di specificare sia il nome della tabella per una classe che una serie di metadati per ogni field.

4.3 MoST

Dopo aver visto gli elementi che compongono un'applicazione Android e, in dettaglio, la struttura del Client ParticipAct, procediamo ora con la descrizione della realizzazione di MoST.

4.3.1 La base del Sistema: Gli Input

Gli Input sono alla base del sistema e sono un'astrazione dei dati provenienti dai sensori o da altre fonti, infatti, grazie al sistema dei broadcast receiver, è possibile ricevere qualunque dato anche di livello applicativo e utilizzarlo come input di MoST.

Gli input attualmente implementati in MoST sono:

Raw Data	Inferences
• Accelerometer • App in use • Battery level • Bluetooth scan • Cell tower ID and signal strength • Gyroscope • Geolocation • Installed apps • Network traffic statistics • Noise level • WiFi scan	• Physical activity (standing stull, walking, running, biking, being in a vehicle) • Walked distance • Sound recognition (human voice, noise, silence) • Length of conversations

Alcuni di questi input sono presenti in due varianti: una periodica che produce dati ogni intervallo di tempo, e una continua. Input con variante periodica sono Accelerometer, Geolocation e Phisical activity.

4.3.2 Input Manager

Tutti gli input sono gestiti da un InputManager che ne controlla il ciclo di vita (vedi Figura 31), instanziandoli dinamicamente con un pattern factory e inserendoli in un pool per poter essere riutilizzati. Una volta raccolti i dati questi vengono inoltrati alle pipeline che ne hanno fatto richiesta tramite un bus.

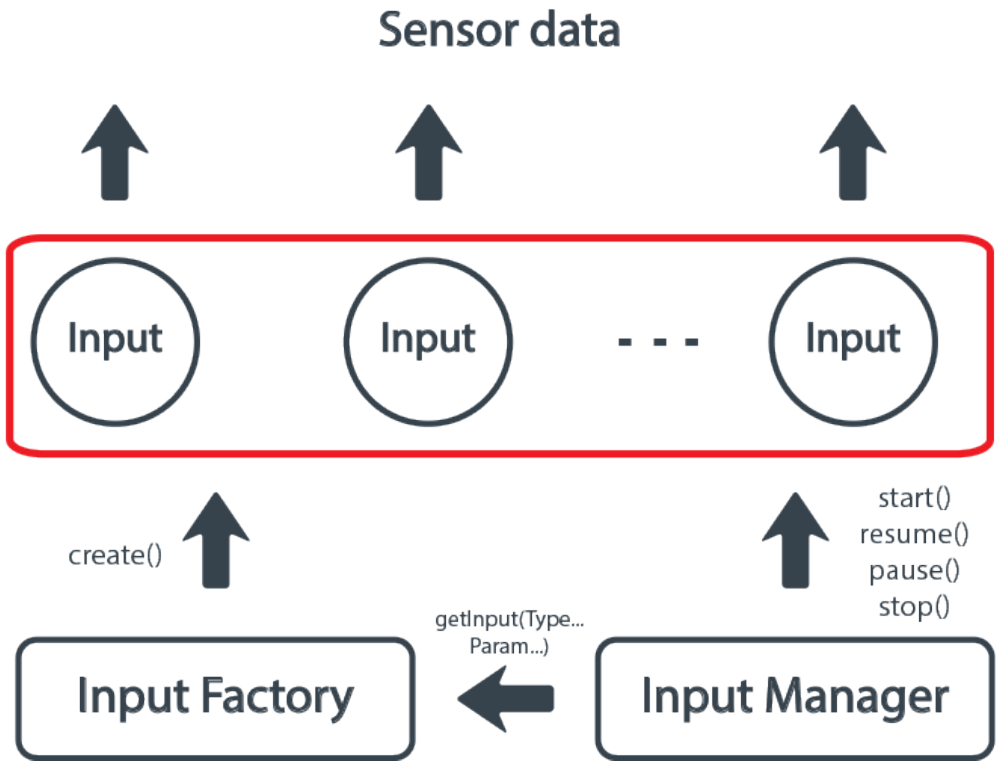

Figura 31: Input manager e ciclo di vita di un input

4.3.3 Bus

MoST utilizza un bus per mettere in collegamento gli Input con le Pipeline che ne hanno fatto richiesta (vedi Figura 32). I dati, poiché sono eterogenei, sono incapsulati in DataBundle che sono riciclati in un pool per ottimizzare l'uso di risorse. Ogni DataBundle ha associato

un intero che rappresenta il numero di Pipeline che necessitano del dato, ogni volta che una pipeline ha finito di elaborare il dato contenuto nel DataBundle decrementa questo valore e quando il valore sarà zero il DataBundle tornerà ad essere disponibile nel pool.

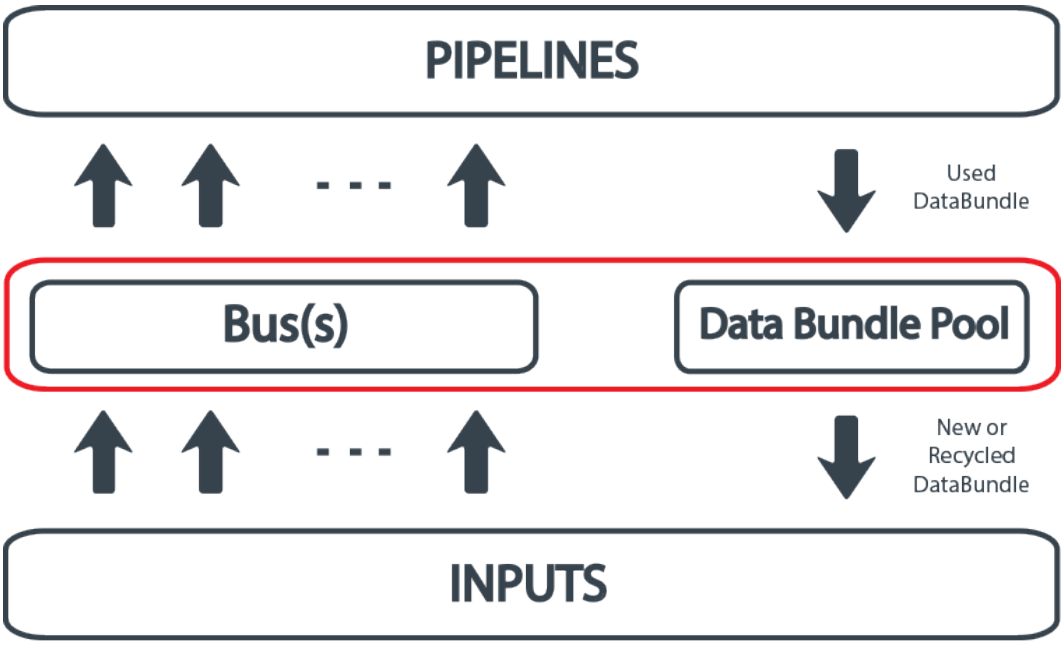

Figura 32: Bus e DataBundle

4.3.4 Pipeline

Una Pipeline è un componente in cui lo sviluppatore deve aggiungere la logica di inferenza per ottenere informazioni di alto livello a partire dagli input verso i quali ha mostrato interesse.

Analogamente a quanto avviene per gli Input anche qui è presente un manager chiamato PipelineManager che utilizza il pattern factory per instanziare le pipeline (vedi Figura 33).

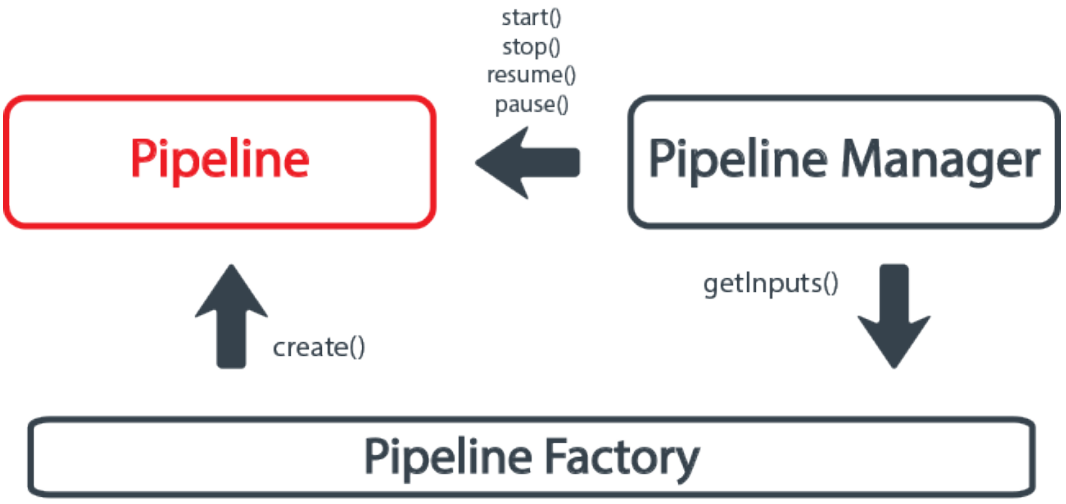

Figura 33: PipelineManager e ciclo di vita di una Pipeline

Per aggiungere una nuova pipeline al framework sarà necessario estendere la classe astratta Pipeline e implementare i seguenti metodi:

- onActivate().
- getInput().
- onData(DataBundle data).

Come facilmente intuibile onActivate viene invocato all'attivazione della Pipeline, in getInput si elencano gli input necessari al funzionamento della pipeline e onData è il metodo che viene richiamato ogni volta è disponibile un nuovo dato dagli Input.

Sarà inoltre necessario registrare la nuova pipeline presso il PipelineManager.

4.3.5 Management

Quando ParticipAct richiede l'esecuzione di un Task di sensing a MoST viene avviato il service MoSTService che si occupa di contattare il PipelineManager per instanziare le pipeline e di conseguenza gli Input necessari per la corretta esecuzione del Task.

Quando poi la pipeline non è più necessaria il sistema di management delegherà all'InputManager e al PipelineManager la disattivazione dei componenti.

L'Interaction Layer notifica a MoST gli eventi ricevuti dal sistema per un corretta gestione delle risorse che necessitano di accesso in mutua esclusione come il microfono che deve essere rilasciato da MoST nell'istante in cui all'utente arriva una chiamata.

A seguito degli eventi ricevuti l'interaction layer notifica l'InputArbiter che sospende o riattiva gli input secondo una politica nella quale quattro componenti esprimono il permesso di abilitare l'input con un valore booleano. I componenti che esprimono il voto sono:

- Sensing: richiesta attività di sensing.
- User: eventi creati da utente.
- Event: eventi di sistema.
- Power: politica di risparmio energetico.

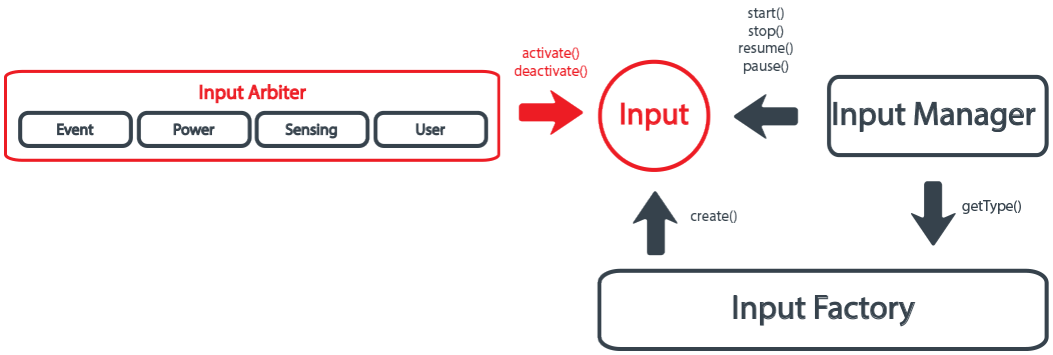

Figura 34: Il ruolo dell'Input Arbiter

Ora che è stata analizzata nel dettaglio l'architettura distribuita di ParticipAct, entrando sia nei componenti collocati nel server sia in quelli nel client mobile è chiaro che le novità di gamification che si introdurranno avranno impatto sui diversi layer sia nel server che nel

client. L'unico componente che non verrà modificato è il framework MoST.

Nel server sarà necessario estendere il layer di presentation per fornire strumenti di gestione agli amministratori e le nuove Mobile API richiamate dal client. Di conseguenza anche il layer di MVC verrà ampliato per gestire le nuove richieste HTTP. L'effettiva logica verrà scritta al livello di business e verranno utilizzati nuovi repository scritti a livello di persistence.

Lato client saranno scritti nuovi Fragment a livello presentation, uno per ogni nuova sezione che si introdurrà, aggiunti alcuni componenti a livello business per l'elaborazione degli elementi che si introdurranno e esteso il modulo di network per recuperare le informazioni necessarie. Il modulo di persistence non verrà invece modificato.

Capitolo 5: Gruppi Geolocalizzati

In questo capitolo verranno forniti i dettagli su come è stata sviluppata la parte del progetto ParticipAct dedicata al calcolo dei Gruppi Geolocalizzati. I componenti che verranno discussi presenteranno l'intera logica per il calcolo dei gruppi.

Si inizierà parlando dei vantaggi nell'utilizzare gruppi geolocalizzati in sistemi che sfruttano il crowdsensing mettendo in evidenza le differenze tra il precedente utilizzo della geolocalizzazione e l'utilizzo della stessa per il calcolo dei gruppi.

Successivamente verrà presentato l'algoritmo utilizzato per il calcolo dei gruppi mettendo in evidenza altre applicazioni dello stesso algoritmo in scenari con diverse piattaforme.

Infine verranno presentate le modifiche effettuate alla logica del server ParticipAct per poter ospitare le nuove funzionalità.

5.1 Vantaggi dei Gruppi Geolocalizzati

Un gruppo è un insieme di persone che interagiscono le une con le altre, in modo ordinato, sulla base di aspettative condivise. Ognuno di essi è composto da un insieme di persone i cui status e i cui ruoli sono in comune.

Gli esseri umani sono portati a cooperare, competere, analizzare, produrre idee, progettare e decidere in gruppo. I gruppi sono, quindi, una parte vitale della struttura sociale e si formano e si trasformano costantemente; non è necessario che siano autodefiniti, infatti molto spesso sono definiti dall'esterno.

Le caratteristiche di un gruppo riguardano per lo più gli atteggiamenti degli individui che ne fanno parte. Spesso i membri interagiscono tra loro influenzandosi a vicenda, rispettando delle regole proprie e, soprattutto, sono dipendenti l'uno dall'altro.

Gli individui, solitamente, decidono di riunirsi in un gruppo per raggiungere obiettivi comuni. Alcuni esempi possono essere visti nel gruppo dei ragazzi che abitualmente si riunisce per organizzare una partita di calcetto o nel gruppo di amici che il sabato sera va nel pub di fiducia per poter vedere la partita della squadra del cuore.
È facile notare, quindi, che gli individui che fanno parte di un gruppo tendono a frequentare gli stessi posti negli stessi orari in modo da passare del tempo insieme.

Alla base di questo lavoro di tesi c'è proprio questa idea. Perché non rendersi conto dei gruppi che i partecipanti al progetto ParticipAct costituiscono abitualmente e perché non organizzare qualcosa che tutti insieme possano svolgere?

Tutto ciò porta sicuramente a numerosi vantaggi dal punto di vista del completamento di un'azione in quanto un singolo individuo può trovare molta difficoltà nel completare un compito, mentre se questo compito viene diviso all'interno di un gruppo risulta essere molto più semplice da portare a termine. Altro vantaggio va visto nell'incremento delle amicizie nella scala sociale perché facendo fare qualcosa ai membri di un gruppo il loro rapporto di amicizia si fortifica e il gruppo continua ad esistere perché legato ad azioni che vanno portate a termine.

5.2 Scelte Progettuali

Il progetto ParticipAct, come già detto, è un progetto di crowdsensing sviluppato dall'Università di Bologna e la raccolta dei dati viene fatta attraverso smartphone distribuiti agli studenti. Uno dei dati fondamentali che viene raccolto è la posizione dello smartphone e quindi dello studente a cui è stato assegnato.

5.2.1 Raccolta dei DataLocation

I *DataLocation* sono un particolare tipo di dato che ParticipAct usa per catturare informazioni riguardanti la posizione degli individui sulla superficie terrestre. I campi di maggiore interesse per questo lavoro di tesi sono latitudine e longitudine e vengono usati per il calcolo della distanza tra due punti distinti. Altri campi che vengono presi in considerazione riguardano l'accuratezza del dato raccolto e il momento in cui il dato è stato raccolto sullo smartphone. La raccolta dei *DataLocation* avviene periodicamente con cadenze di un minuto e quindi gli aggiornamenti sulla posizione sono molto precisi e dettagliati.

Ogni smartphone, come già detto, possiede nativamente un sensore GPS e una connessione ad Internet e proprio questi due mezzi vengono utilizzati per determinare con una certa accuratezza la posizione sulla superficie terrestre. Successivamente alla raccolta del *DataLocation* l'applicazione Android provvede ad inviarlo al server in modo da poter essere analizzato.

Figura 35: Architettura per il calcolo della Posizione

La Figura 35 mostra una semplice architettura per la raccolta di dati sulla posizione attraverso uno smartphone e per la loro conservazione in un Location Server.

5.2.2 Utilizzo dei DataLocation per il calcolo dei Gruppi

Il *DataLocation* è un tipo di dato che contiene molte informazioni per la descrizione di una punto sulla superficie terrestre e quindi è stato scelto come punto di partenza per il calcolo di gruppi geolocalizzati. Per snellire i tempi di calcolo si è preferito tenere in considerazione solo alcuni campi di questo dato.

Figura 36: Dati di Interesse

Nella Figura 36 vengono mostrati i campi presi in considerazione:

- **LATITUDINE:** rappresenta la coordinata geografica pari all'altezza del polo celeste sull'orizzonte;
- **LONGITUDINE:** rappresenta la coordinata geografica che indica la distanza angolare in senso Est o Ovest dal meridiano di Greenwich;
- **ACCURATEZZA:** rappresenta il grado di precisione del dato raccolto;
- **MOMENTO DELLA RACCOLTA:** rappresenta il momento in cui lo smartphone richiede al GPS Provider o al Network Provider la posizione attuale.

Un campo che merita una trattazione più dettagliata è sicuramente l'accuratezza in quanto i punti che non hanno un grado di accuratezza accettabile vengono scartati a priori dal calcolo dei gruppi.

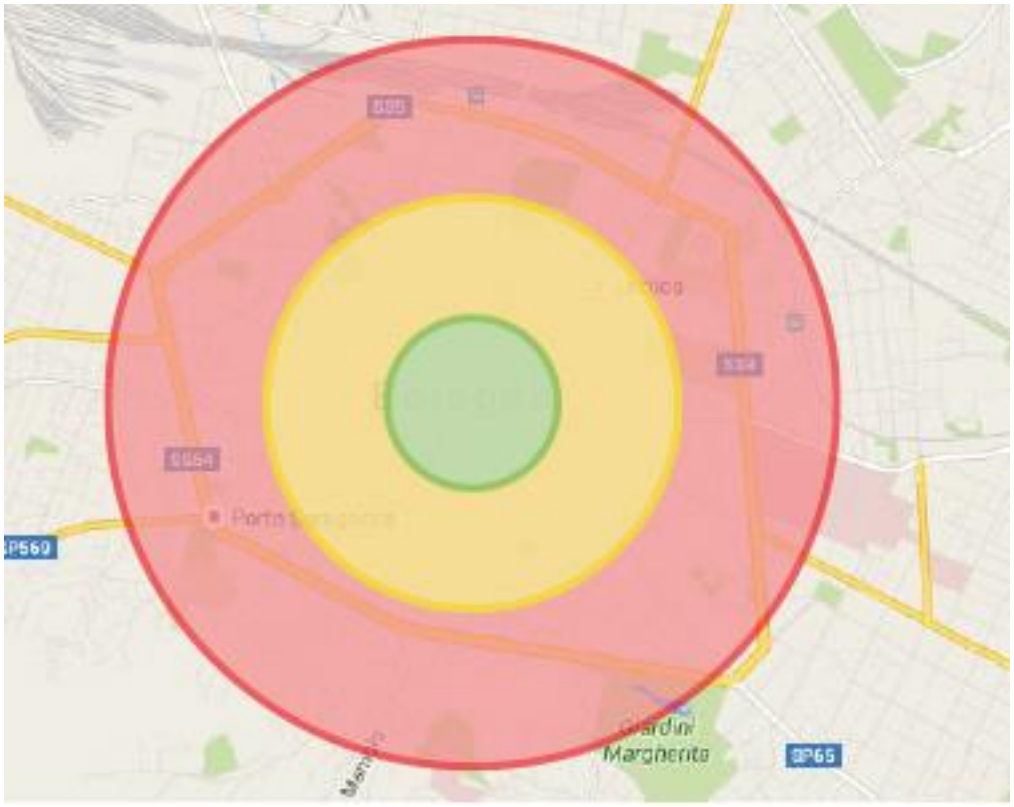

Figura 37: Livelli di Accuratezza

La Figura 37 mostra come l'accuratezza è misurabile facilmente in 3 livelli:

- **ALTA ACCURATEZZA:** è presente nella zona in verde e indica che il punto di interesse si trova all'interno di un raggio massimo di 100 m;
- **MEDIA ACCURATEZZA:** è presente nella zona in giallo e indica che il punto di interesse si trova tra un raggio di 100 m e un raggio di 500 m;
- **BASSA ACCURATEZZA:** è presente nella zona in rosso e indica che il punto di interesse si trova oltre un raggio di 500 m.

Per lo sviluppo di questo lavoro di tesi è stato scelto di non considerare i punti con bassa accuratezza in quanto il loro contributo al calcolo era minimo.

5.3 Calcolo dei Gruppi

Una volta vista l'importanza che i gruppi hanno nella vita sociale di ogni individuo e i vantaggi che oggetti come i *DataLocation* potessero fornire al calcolo, rimane da individuare il giusto algoritmo per effettuare il calcolo.

Gli studi sul calcolo dei gruppi sono già stati affrontati in passato e uno di essi ha portato allo sviluppo dell'algoritmo **k-CLIQUE** [32].

5.3.1 Presentazione del k-CLIQUE

Il k-CLIQUE è un algoritmo utilizzato dall'Università di Cambridge per il calcolo di gruppi di individui che passano insieme determinate ore della giornata.

Questo algoritmo si basa sul principio che l'essere umano tende ad avere sempre più spesso rapporti con altre persone. Questi rapporti sono caratterizzati principalmente da due caratteristiche:

- **Durata:** indica il tempo che le due persone trascorrono insieme, ad esempio se in una giornata l'individuo A ha incontrato l'individuo B una prima volta dalle 11:00 alle 12:00 e una seconda volta dalle 16:00 alle 17:00, i due individui hanno passato insieme 2 ore della giornata.
- **Numero di incontri:** indica il numero di incontri che due individui hanno avuto in una giornata, ad esempio se un individuo C incontra un individuo D una prima volta dalle 8:00 alle 8:30, una seconda volta dalle 15:30 alle 16:30 e una terza volta dalle 20:00 alle 22:00, il numero di incontri sarà pari a 3.

In base a questi due parametri si possono individuare 4 gruppi fondamentali in cui si distribuiscono i vari contatti di un singolo essere umano.

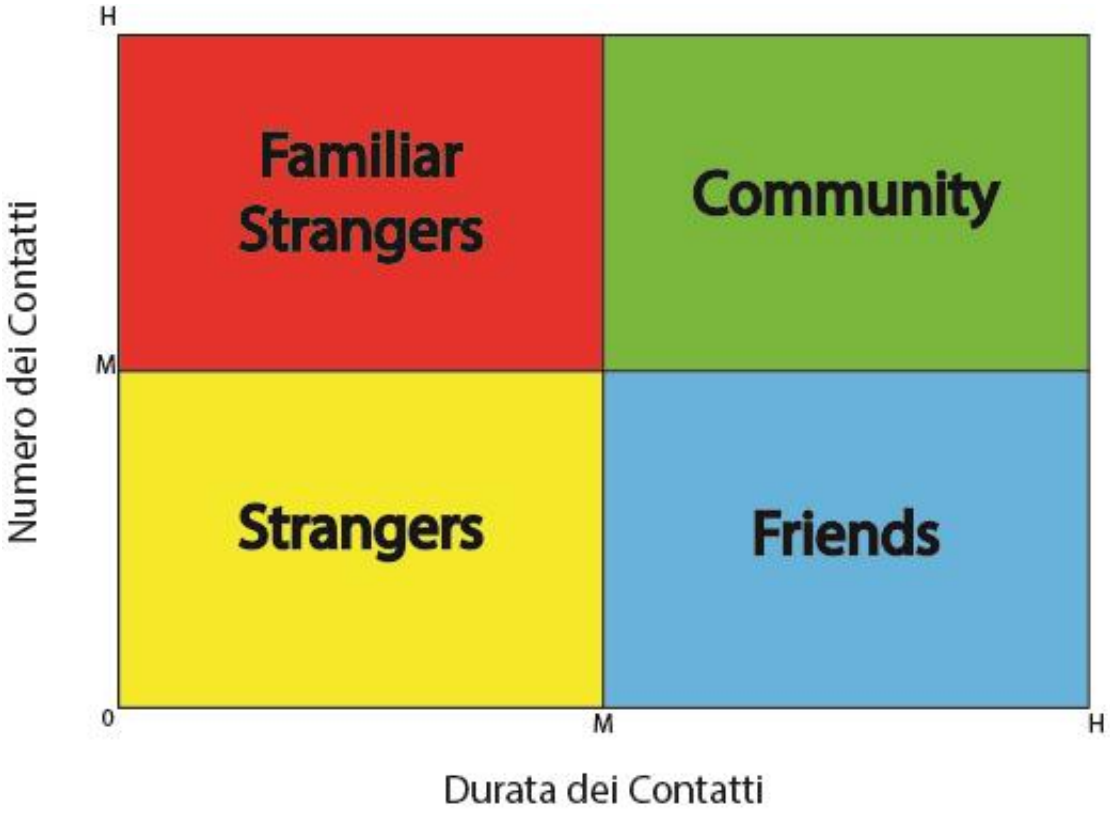

Figura 38: Gruppi individuati attraverso il k-CLIQUE

La Figura 38 mostra i 4 gruppi che vengono calcolati per ogni individuo, ognuno di essi indica il rapporto che lo stesso ha con gli altri. In particolare:

- **Strangers:** indica l'insieme degli individui che hanno pochi rapporti con l'individuo selezionato. Questo gruppo è caratterizzato da un basso numero di contatti e da una bassa durata di questi ultimi. Esempi di componenti di questo gruppo sono da trovare nelle persone che sporadicamente incontriamo per strada, ma che non conosciamo;
- **Familiar Strangers:** indica l'insieme degli individui che hanno un numero medio-basso di rapporti con l'individuo selezionato. Questo gruppo è caratterizzato da un alto numero di contatti e da una bassa durata di questi ultimi. Esempi di componenti di questo gruppo sono da trovare nelle persone che giornalmente incontriamo, ma con cui non passiamo molto tempo insieme;
- **Community:** indica l'insieme degli individui che hanno molti rapporti con l'individuo selezionato. Questo gruppo è caratterizzato da un alto numero di contatti e da una alta durata di questi ultimi. Esempi di componenti di questo gruppo sono da trovare nelle persone che incontriamo più volte nella stessa giornata e con cui passiamo molto tempo insieme;
- **Friends:** indica l'insieme degli individui che hanno rapporti di alta durata con l'individuo selezionato. Questo gruppo è caratterizzato da un basso numero di contatti e da una alta durata di questi ultimi. Esempi di componenti di questo gruppo sono da trovare nelle persone che passano molto tempo insieme e si separano poche volte durante l'arco della giornata. Il basso numero di contatti è dovuto al fatto che i componenti di questo gruppo stanno per molto tempo insieme, a volte hanno un solo contatto di durata l'intera giornata (è il caso dei coinquilini).

5.3.2 Applicazione del k-CLIQUE a Cambridge

Un primo test dell'algoritmo *k-CLIQUE* è stato fatto nell'Università di Cambridge dove sono stati distribuiti 54 smartphone tra gli studenti del primo e del secondo anno per 11 giorni.

A differenza del progetto ParticipAct a Cambridge hanno utilizzato come base per la raccolta dei contatti il sensore Bluetooth sviluppando un'applicazione che permettesse agli smartphone di connettersi tra loro nel momento in cui si trovassero ad una distanza minore di 5m. Una volta avvenuta la connessione si provvedeva ad incrementare il numero dei contatti di una unità e la durata del contatto di un tempo in secondi pari alla durata della connessione tra gli smartphone.

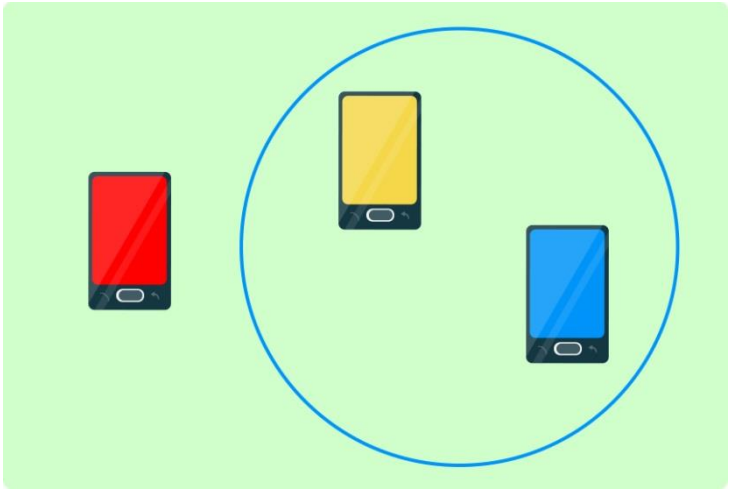

Figura 39: Area di Connessione

La Figura 39 mostra un tipico esempio di una connessione tra smartphone. Lo smartphone blu e quello giallo entrano in connessione tra di loro essendo ad una distanza minore di 5m, mentre lo smartphone rosso non ha contatti con gli altri smartphone in quanto la sua distanza da entrambi gli altri due dispositivi è superiore ai 5m.

5.3.3 Il k-CLIQUE in ParticipAct

L'applicazione ParticipAct, come già detto, raccoglie dati sulla posizione degli smartphone periodicamente. Si è scelto quindi di applicare l'algoritmo k-CLIQUE basandosi sulle posizioni raccolte.

Il procedimento che viene adottato è leggermente diverso da quello di Cambridge nella valutazione del contatto. Il calcolo è caratterizzato da una data di inizio e da una data di fine che rappresentano le date entro le quali considerare i *DataLocation.* Ogni utente, durante il calcolo, viene rappresentato attraverso la classe *it.unibo.paserver.groups.GroupUser.java* che provvede a mantenere in memoria due parametri fondamentali per ogni utente che abbia interazioni con l'utente selezionato:

- **duration***:* rappresenta il tempo in ore che i due individui trascorrono insieme;
- **numberofcontacts***:* rappresenta il numero degli incontri tra due utenti.

Oltre ai due parametri fondamentali per il calcolo, ogni *GroupUser* provvede a tener traccia dei gruppi **FRIENDS**, **COMMUNITY**, **FAMILIAR STRANGERS** *e* **STRANGERS** dell'utente.

Nel primo passo del calcolo il sistema provvede a recuperare i *DataLocation* dal database sottostante. Successivamente, per ogni utente, viene stimata la posizione media in ogni ora della giornata e viene calcolata la distanza dell'individuo selezionato da ogni altro individuo del progetto attraverso la funzione *getDistance(double latA, double lonA, double latB, double lonB)* che provvede a calcolare la distanza tra due punti sulla superficie terrestre.

```
public static double getDistance(double latA, double lonA, double latB,
        double lonB) {

    double lat_alfa, lat_beta;
    double lon_alfa, lon_beta;
    double fi;
    double p, d;

    lat_alfa = pigreco * latA / 180;
    lat_beta = pigreco * latB / 180;
    lon_alfa = pigreco * lonA / 180;
    lon_beta = pigreco * lonB / 180;

    fi = Math.abs(lon_alfa - lon_beta);

    p = Math.acos(Math.sin(lat_beta) * Math.sin(lat_alfa)
            + Math.cos(lat_beta) * Math.cos(lat_alfa) * Math.cos(fi));

    d = p * R * inM;
    return d;
}
```

Figura 40: Funzione per il Calcolo della Distanza tra due punti

Dopo il calcolo della distanza il sistema decide se considerare positivamente o negativamente un contatto tra due utenti nell'ora presa in esame in base al parametro **DISTANCE** presente nella classe *it.unibo.paserver.groups.GroupUtils.java* nella seguente maniera:

- Se *getDistance(…)* > **DISTANCE***:* il sistema valuta il contatto tra i due individui negativamente;
- Se *getDistance(…)* <= **DISTANCE***:* il sistema valuta il contatto tra i due individui positivamente;.

Se il sistema valuta positivamente il contatto si prosegue con l'aggiornamento dei gruppi dell'utente in questo modo:

- Se i due utenti hanno avuto un contatto anche nell'ora precedente viene incrementato unicamente il parametro **duration***;*
- Se i due utenti non hanno avuto un contatto nell'ora precedente vengono incrementati sia il parametro **duration** che il parametro **numberofcontacts**.

Dopo l'incremento di uno o di entrambi i parametri, il sistema provvede all'aggiornamento dei gruppi utente considerando i parametri **CONTACTSINDAY** e **INFAMILY** presenti nella classe *it.unibo.paserver.groups.GroupUtils.java* in questo modo:

- Se **duration** > **INFAMILY** e **numberofcontacts** > **CONTACTSINDAY** l'utente viene spostato nel gruppo **COMMUNITY**;
- Se **duration** > **INFAMILY** e **numberofcontacts** < **CONTACTSINDAY** l'utente viene spostato nel gruppo **FRIENDS**;
- Se **duration** < **INFAMILY** e **numberofcontacts** > **CONTACTSINDAY** l'utente viene spostato nel gruppo **FAMILIAR STRANGERS**;
- Se **duration** < **INFAMILY** e **numberofcontacts** < **CONTACTSINDAY** l'utente viene spostato nel gruppo **STRANGERS**.

Si prosegue in questo modo fino all'esaurimento dei *DataLocation* nel periodo considerato. Al termine del calcolo, il sistema provvede in automatico a rendere permanenti i gruppi nel database.

La Figura 41 mostra le interazioni tra i componenti durante il calcolo dei gruppi.

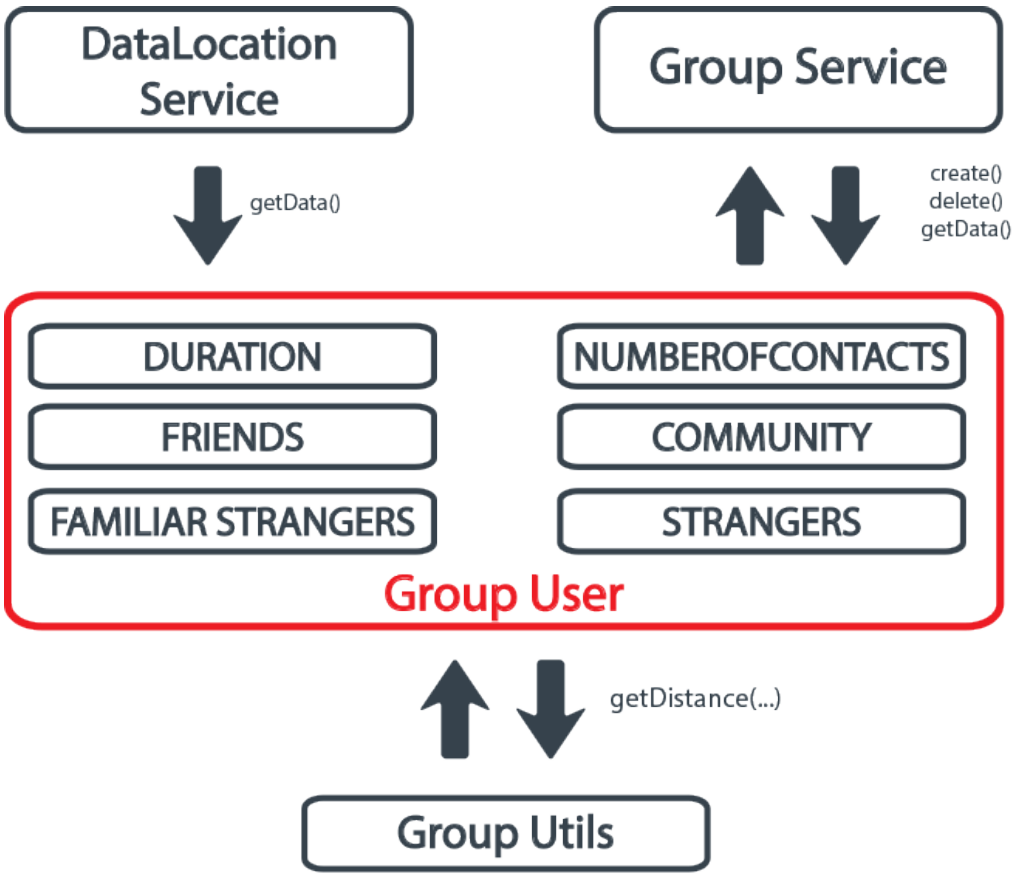

Figura 41: Calcolo dei Gruppi

A causa della grande quantità di dati raccolti attraverso gli smartphone il calcolo viene eseguito lato Server in modo da non appesantire i Client. In seguito vedremo più dettagliatamente le modifiche effettuate sul Server per poter ospitare la nuova funzionalità.

5.4 Modifiche effettuate al Server

Per poter implementare il calcolo dei gruppi si è scelto di estendere il Server ParticipAct aggiungendo le nuove funzionalità.

Nella nuova *Console*(vedi Figura 42) sono stati inseriti due nuovi moduli:

- **Group Management:** permette agli amministratori di creare facilmente nuovi gruppi sulla base dei *DataLocation;*
- **Task Group Management:** permette agli amministratori di assegnare Task ai gruppi e di visualizzare i risultati ottenuti.

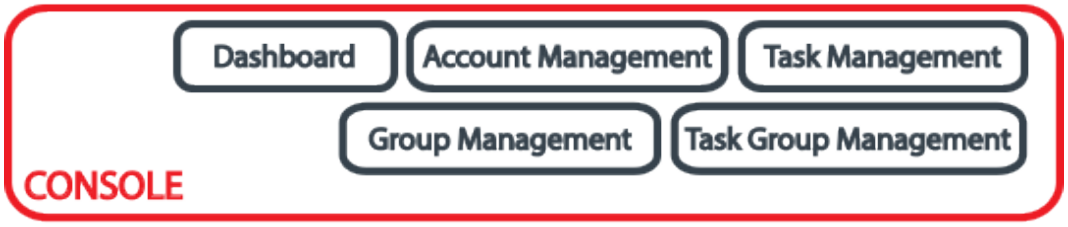

Figura 42: Nuova Console

I due nuovi moduli dialogano con il Layer di *PERSISTENCE* per poter rendere l'analisi persistente (vedi Figura 43).

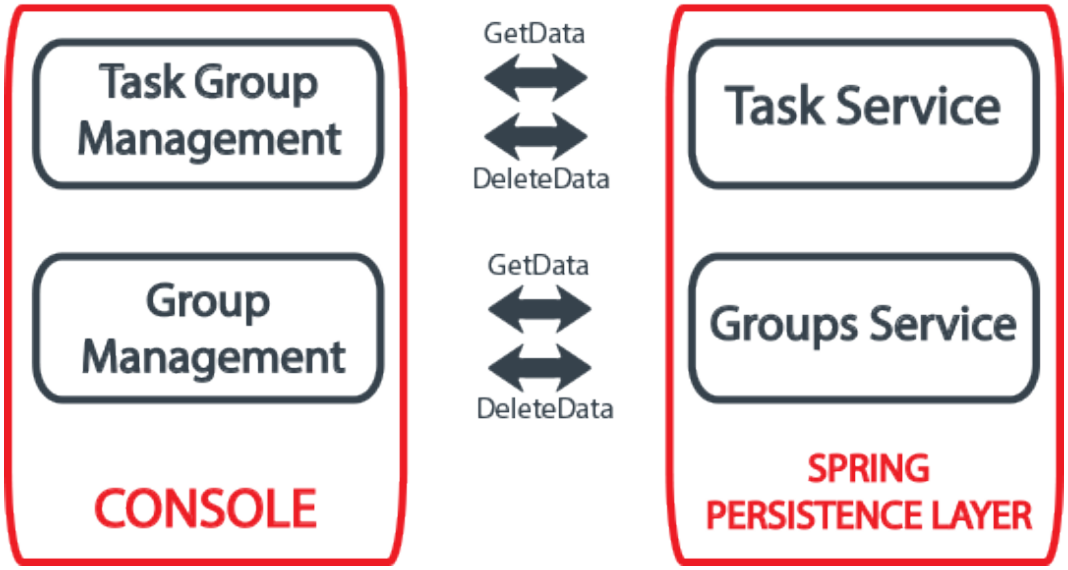

Figura 43: Interazione dei nuovi Moduli

Un'ultima modifica è stata effettuata alla *Dashboard per* aggiungere il tasto *Groups* che permette di utilizzare la funzione di calcolo gruppi (vedi Figura 44).

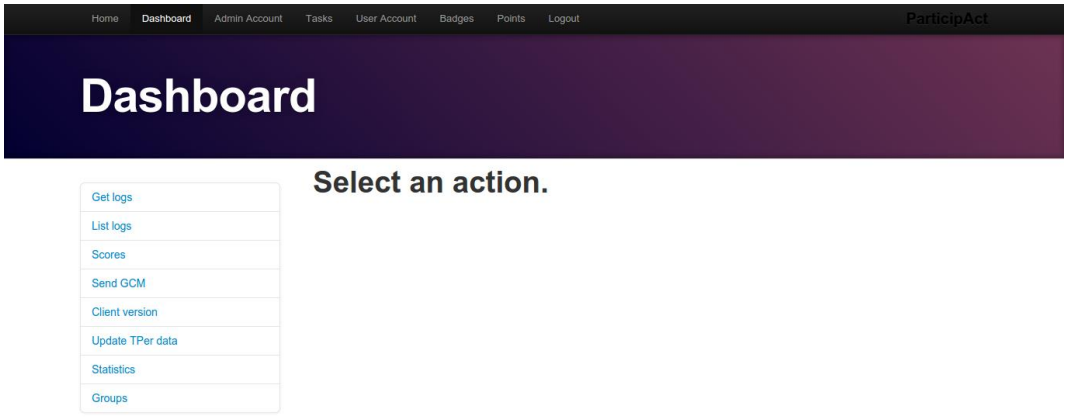

Figura 44: Nuova Dashboard

5.4.1 Il Calcolo dei Gruppi

La nuova *Dashboard*, oltre a tutte le precedenti opzioni, include il calcolo dei gruppi. Una volta entrati nella sezione *Groups* sarà possibile inserire due date che rappresentano il giorno in cui si desidera partire con il calcolo e il giorno in cui si desidera terminare il calcolo.

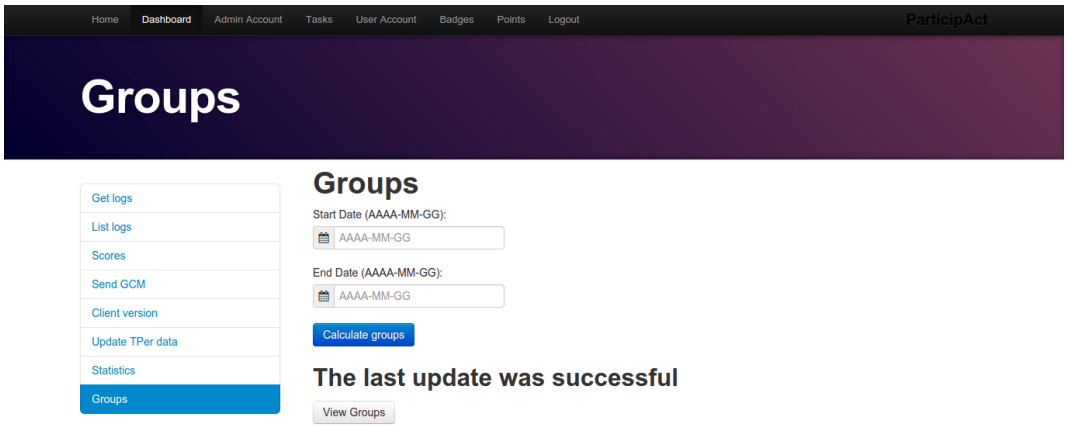

Figura 45: Pagina per il calcolo dei Gruppi

La pagina presentata nella Figura 45 è stata realizzata tramite una jsp chiamata *protected/groups/main.jsp*. Nella jsp, oltre ai due campi per l'inserimento delle date, sono presenti due *button*:

- *Calculate Groups:* permette di proseguire con il calcolo dopo aver inserito le due date.
- *View Groups:* permette di visualizzare i gruppi dopo la creazione.

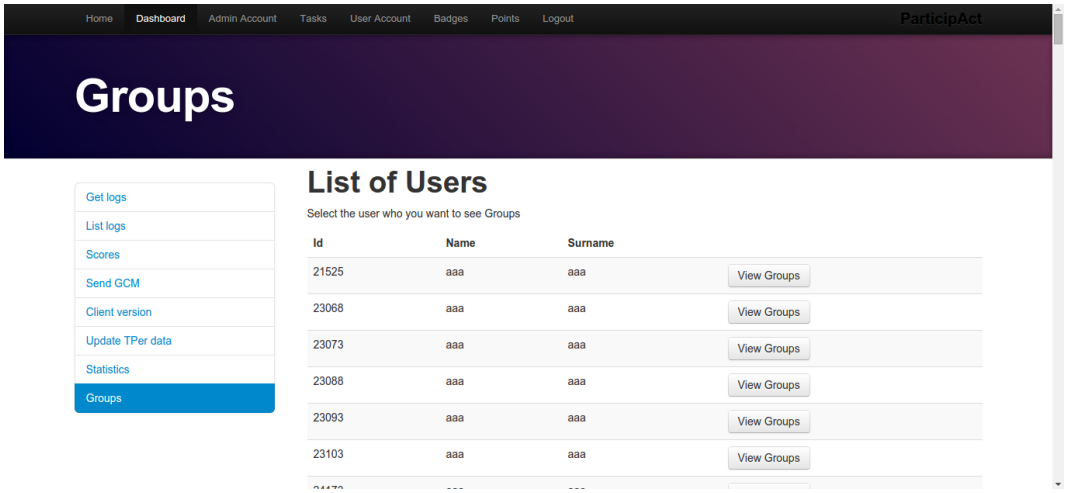

Figura 46: Visualizzazione dei Gruppi

La Figura 46 mostra la jsp denominata *protected/groups/listOfUser.jsp* che permette la visualizzazione di tutti gli utenti che partecipano al progetto. L'elenco è stato realizzato attraverso una tabella che, per ogni utente, permette di visualizzare *id*, *nome*, *cognome e* un *button* "*View Groups"* per la visualizzazione dei gruppi dell'utente selezionato.

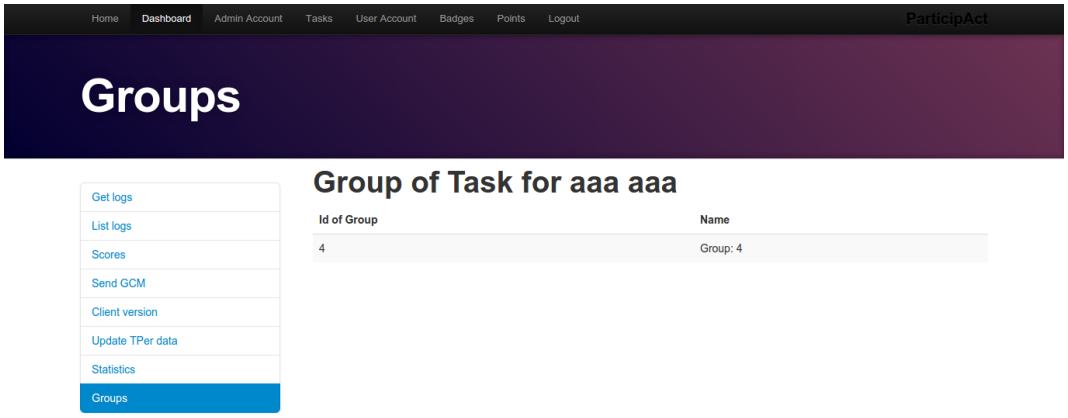

Figura 47: Gruppi per Singolo Utente

La jsp di nome *protected/groups/listOfUser.jsp*, mostrata in Figura 47, permette agli amministratori di visualizzare i gruppi a cui ogni utente appartiene. Il corpo della jsp è formato da una tabella che mostra l'*id e* il *nome* del gruppo.

È importante ricordare che la navigazione di questa sezione è regolata da un apposito Controller chiamato *GroupsController*. La logica per il calcolo dei gruppi è, invece, contenuta nel package *it.unibo.paserver.groups*. Classi degne di una presentazione sono:

- **GroupsUtils:** è la classe che contiene alcuni parametri di configurazione per il calcolo e le funzioni necessarie per il calcolo della distanza;
- **GroupUser:** è la classe che rappresenta un utente dopo il calcolo e contiene al proprio interno i gruppi friends, community, strangers e familiar strangers dello stesso utente;
- **GroupCalculate:** è la classe che si occupa del vero e proprio calcolo dei gruppi sfruttando istanze delle interfacce *IGroupUserDataManager*, *IGroupUserElementDataManager* e *IGroupUserInizialized*.

Capitolo 6: Task Comunitario

Il calcolo di gruppi geolocalizzati permette di individuare gruppi di persone che abitualmente si frequentano passando del tempo insieme. Ad esempio possiamo trovare gli amici che il sabato sera escono per andare a vedere la partita della propria squadra preferita o persone che abitualmente si ritrovano in Università per studiare assieme.

Una volta trovati questi gruppi resta il problema di rafforzare il rapporto che esiste tra i componenti. Un modo per fare ciò è quello di riuscire a far fare qualcosa ai membri del gruppo.
Il progetto ParticipAct, attraverso il modello dei Task, si sposa perfettamente con questa esigenza permettendo l'implementazione di un Task Comunitario, ovvero dove ogni membro del gruppo può completare una piccola parte per poi raggiungere tutti insieme il completamento con successo del Task.

6.1 Descrizione del Task

Il nuovo Task nasce come mezzo per rafforzare le interconnessioni in un gruppo. Per fare ciò si rende necessario il soddisfacimento di determinate caratteristiche:

- Bisogna cercare di coinvolgere tutti i componenti del gruppo e non solo i membri più attivi;
- Bisogna premiare sia i componenti attivi che completano i singoli step e sia i componenti passivi del gruppo.

Per il soddisfacimento della prima caratteristica si è scelto di utilizzare un workflow che inviasse a tutti i membri l'attuale step del task in modo da notificarli sull'avanzamento del lavoro di gruppo, mentre per il soddisfacimento della seconda caratteristica si è deciso di apportare

determinate modifiche alla gamification in modo da accogliere una nuova metodologia per l'attribuzione di punti.

Come attività base per lo svolgimento del Task comunitario si è scelto di porre una domanda al gruppo e di suddividere la risposta in parti più piccole che rappresentano i singoli step del Task. Ad ogni step deve essere comunicato ai membri del gruppo l'attuale stato di avanzamento del Task e le risposte precedentemente date dai propri amici. Un esempio potrebbe essere composto dalla domanda "*Quali sono le prime 3 classificate del mondiale di calcio del 2006 in ordine dalla prima alla terza?*" e dalle risposte "*Italia*", "*Francia*" e "*Germania*"; in questo esempio creeremmo un Task con 3 step.

6.1.1 Gli Step

Un gruppo risulta essere composto da un numero di persone ed ognuna di esse può essere più o meno attiva. La caratteristica fondamentale del gruppo è, però, quella di fare un po' a testa per poter raggiungere un obiettivo comune.
All'interno del task comunitario si è scelto di coinvolgere tutti i componenti del gruppo suddividendo il compito da portare a termine in tanti piccoli step.

Figura 48: Step del Task

La Figura 48 mostra il funzionamento del nuovo Task: man mano che un componente di un gruppo termina uno step il gruppo può avanzare

allo step successivo. Una volta concluso l'n-esimo step il task viene ritenuto concluso con successo. Il numero di step non è definito apriori, ma può essere scelto durante la creazione del task da parte degli amministratori.

6.1.2 Il Workflow

Una volta effettuata la scelta della divisione del task in singoli step dipendenti tra loro, si è preferito scegliere un workflow che permettesse a tutti gli utenti di essere partecipi allo svolgimento delle attività.

Il workflow è stato realizzato in modo tale da essere facilmente esteso in caso di future modifiche e il suo funzionamento è illustrato nella Figura 49.

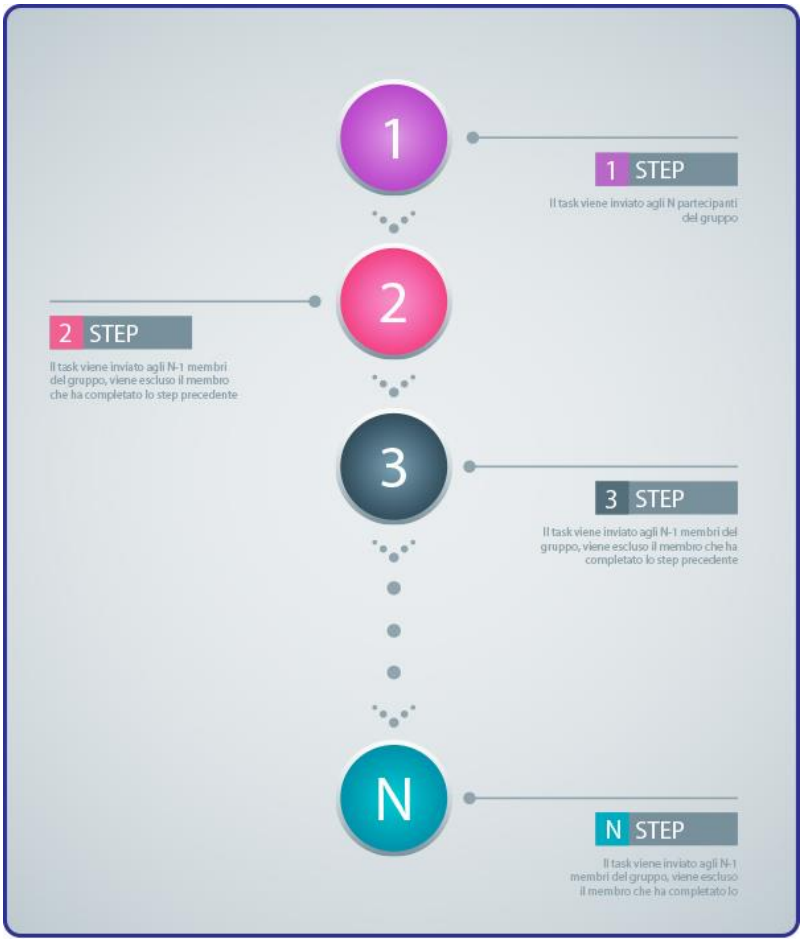

Figura 49: Workflow base del Task comunitario

Il funzionamento del workflow illustrato nella Figura 46 è molto semplice:

1. il primo step viene inviato agli N membri del gruppo;
2. alla ricezione della risposta da parte di un membro del gruppo, lo step corrente viene disabilitato per gli atri membri;
3. viene inviato il prossimo step a tutti i membri del gruppo escluso il risolutore del precedente step;
4. alla ricezione della nuova risposta si prosegue tornando al punto 2 se ci sono ulteriori step oppure si ritiene il task concluso se non ci sono ulteriori step.

Come si può notare il workflow scelto permette di inviare il task comunitario anche a gruppi di sole 2 persone. Sono stati volutamente esclusi i gruppi di una sola persona in quanto questo concetto va contro l'idea di gruppo.

La logica del Workflow per l'invio degli step del Task è stata realizzata nella classe *it/unibo/paserver/pipelinetask/BaseStrategy.java* che implementa l'interfaccia *it/unibo/paserver/pipelinetask/PipelineTaskStrategy.java.* La Figura 50 mostra il metodo per l'assegnamento degli utenti al Task per poter inviare gli step ai vari utenti.

```
@Override
public List<Long> getUserToSendTask(List<Long> usersOfGroup,
        List<Long> userCompleteTask) {
    ArrayList<Long> result = new ArrayList<Long>();

    for (Long l : usersOfGroup)
        if (!userCompleteTask.contains(l))
            result.add(l);

    return result;
}
```

Figura 50: Funzione per l'Assegnamento degli Utenti al Task

Va ricordato che ogni singolo step è stato implementato come un singolo task dipendente da altri task; è quindi possibile decidere il momento di invio dello step, il momento di conclusione dello step, la durata dello step e tutte le altre caratteristiche appartenenti ad un singolo task.

6.2 Scelta dei Gruppi per il Task

L'algoritmo *k-CLIQUE* permette di individuare per ogni utente 4 gruppi di persone (*FRIENDS, COMMUNITY, FAMILIAR STRANGERS e STRANGERS)* in base ai rapporti e al tempo che passa con esse. Questi gruppi, però, risultano essere troppo estesi per poter essere utilizzati nel nuovo Task. Si rendono, quindi, necessari alcuni accorgimenti per poterli utilizzare.

Alla base dell'idea di gruppo c'è il concetto di amicizia, si è quindi scelto di assegnare ad ogni gruppo utente un valore secondo un criterio di amicizia. La seguente lista mostra i gruppi utente a partire da quello maggiormente preso in considerazione fino a quello considerato minimamente:

1. *FRIENDS;*
2. *COMMUNITY;*
3. *FAMILIAR STRANGERS;*
4. *STRANGERS.*

Per la costituzione dei gruppi per il Task di ogni singolo utente vengono considerati, quindi, prima i componenti del gruppo *FRIENDS*, poi quelli dei gruppo *COMMUNITY*, successivamente quelli del gruppo *FAMILIAR STRANGERS* e solo alla fine i componenti del gruppo *STRANGERS*.

Applicando questi accorgimenti sui dati raccolti in 6 mesi si è notato subito un numero medio di gruppi che si aggira intorno ai 35 (vedi Figura 51).

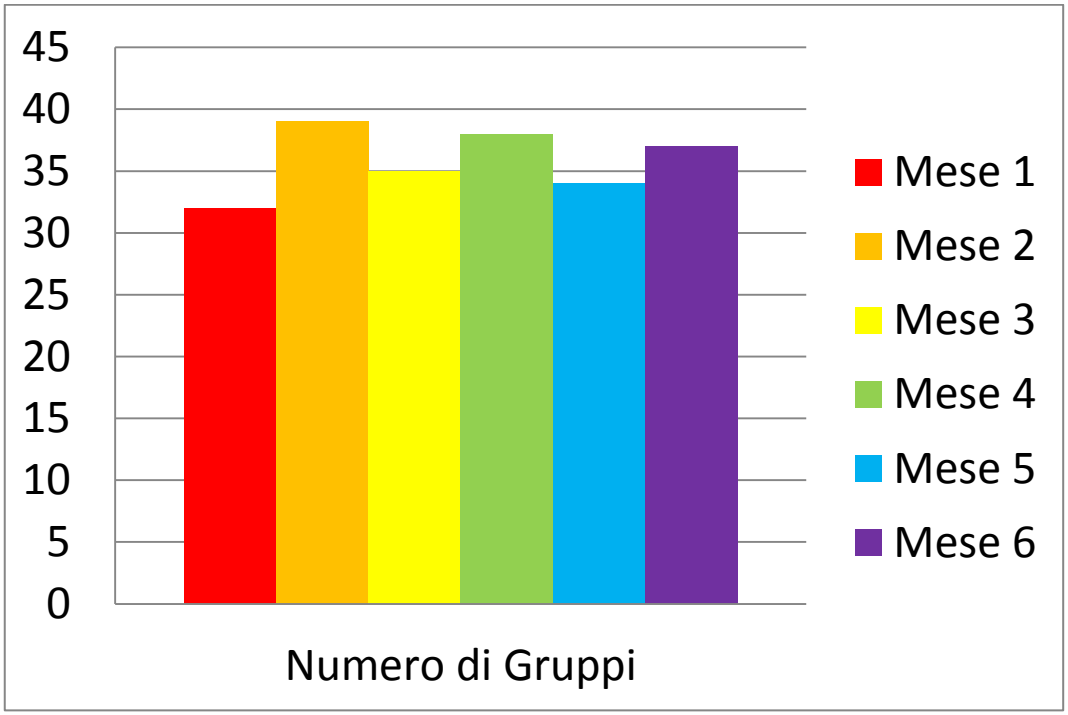

Figura 51: Numero di Gruppi calcolati in 6 mesi

6.3 Creazione del Task Comunitario

L'attuale webflow per la creazione di un Task non era sufficiente per ospitare il nuovo tipo di Task, si è, quindi, deciso di estenderlo in modo da permettere agli amministratori di creare facilmente i Task comunitari.

La Figura 52 mostra le estensioni effettuate sul Webflow. In particolare sono state aggiunte le seguenti funzioni:

- **Pipeline Action:** permette di aggiungere una action di gruppo al Task;
- **Assign Task to Groups:** permette di assegnare il Task ai Gruppi precedentemente creati.

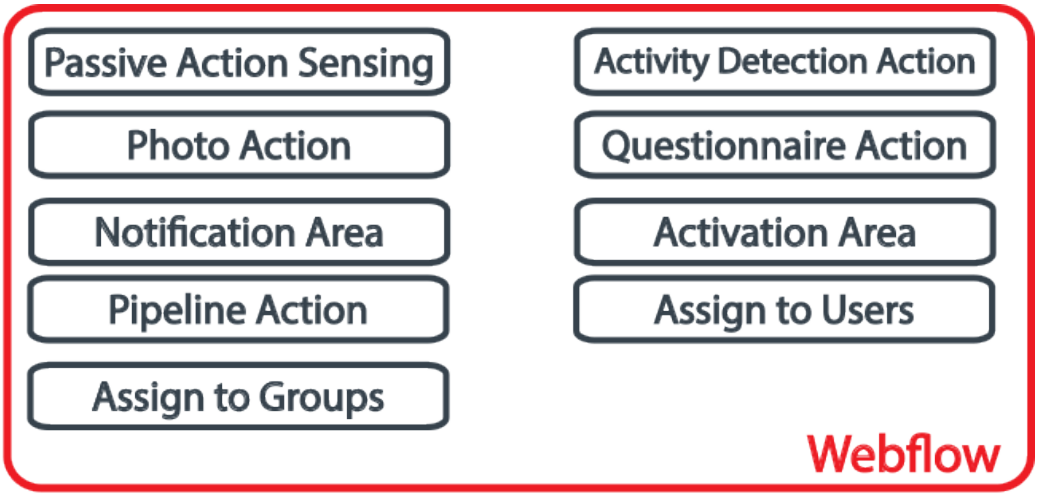

Figura 52: Modifiche al Webflow

Successivamente si è provveduto ad aggiungere le nuove jsp per la creazione del nuovo Task. Il primo passo è stato aggiungere un'apposita jsp denominata *protected/task/pipelinetask.jsp* per il management dei task comunitari (la Figura 53 mostra la nuova pagina di gestione). Il nuovo task viene mostrato in una tabella con i seguenti dati:

- *id:* identificatore univoco del task;
- *startdate:* data di inizio del task;
- *name:* nome dato al task dagli amministratori al momento della creazione;
- *state:* stato corrente del task (permette agli amministratori di avere un riscontro visivo sui task in esecuzione).

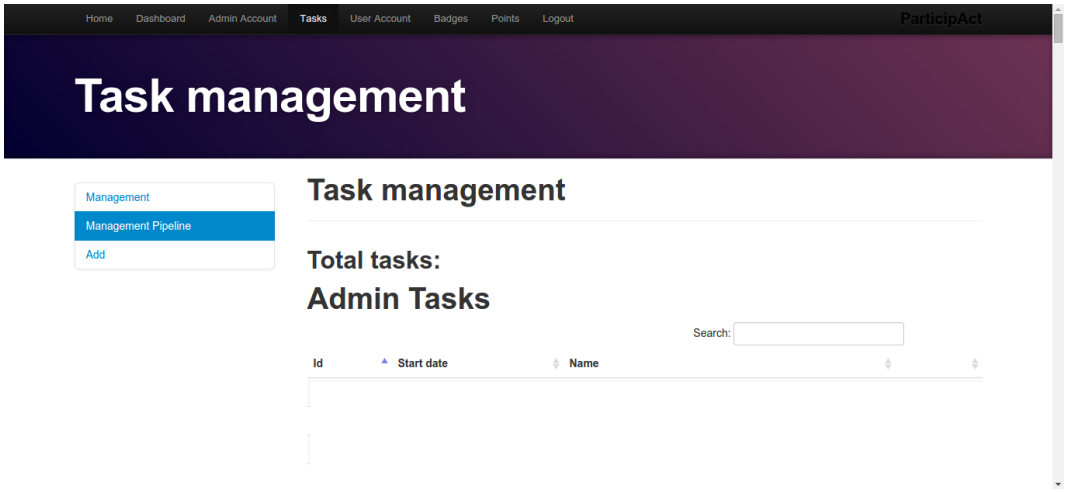

Figura 53: Management Task Comunitari

Il passo successivo è stato apportare modifiche alla procedura per la creazione di un Task. La prima parte, dove gli amministratori devono specificare le date di inizio e fine del Task, la durata e il nome sono rimaste invariate; sono state, invece, fatte delle modifiche nella sezione dove è possibile aggiungere le action al Task. Queste modifiche consistono nell'aggiunta del *button "Add a Pipeline Action"* e del *button "Assign Task to group"* alla jsp denominata *protected/task/add/selectActionType.jsp*. Il primo permette *button* di aggiungere un'azione di gruppo al Task, mentre, il secondo permette di assegnare il Task ai gruppi di utenti creati in precedenza. La Figura 54 mostra la nuova schermata per l'aggiunta delle *action*.

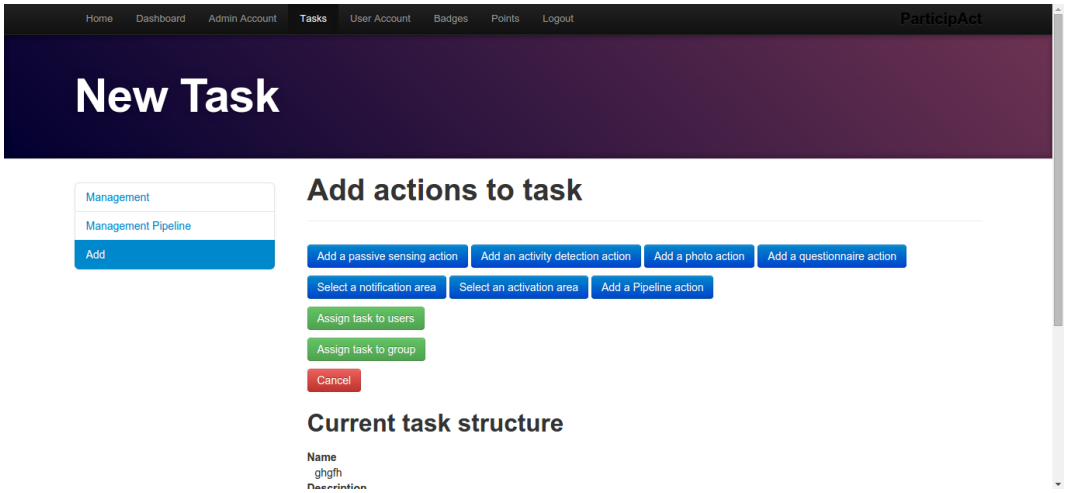

Figura 54: Modifiche della sezione Action

Dopo aver selezionato l'aggiunta di un'azione comunitaria il nuovo webflow permette di inserire il *titolo* e la *domanda* da porre ai membri del gruppo. La Figura 55. mostra la jsp chiamata *protected/task/add/actionPipelineTask.jsp* per l'inserimento del titolo e della domanda attraverso due apposite *textbox*.

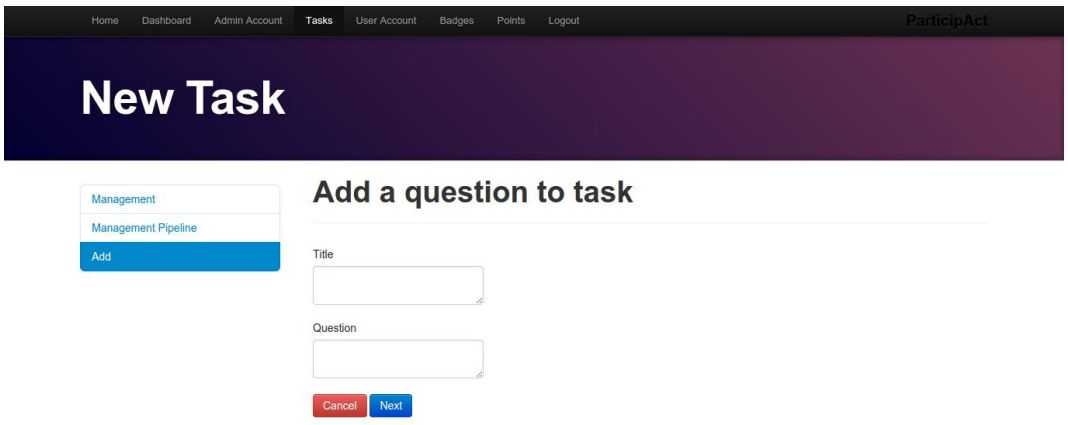

Figura 55: Inserimento Titolo e Domanda

Dopo aver inserito il titolo e la domanda il sistema prevede una sezione dove è possibile inserire le risposte che i membri del gruppo dovranno fornire per ogni singolo step.

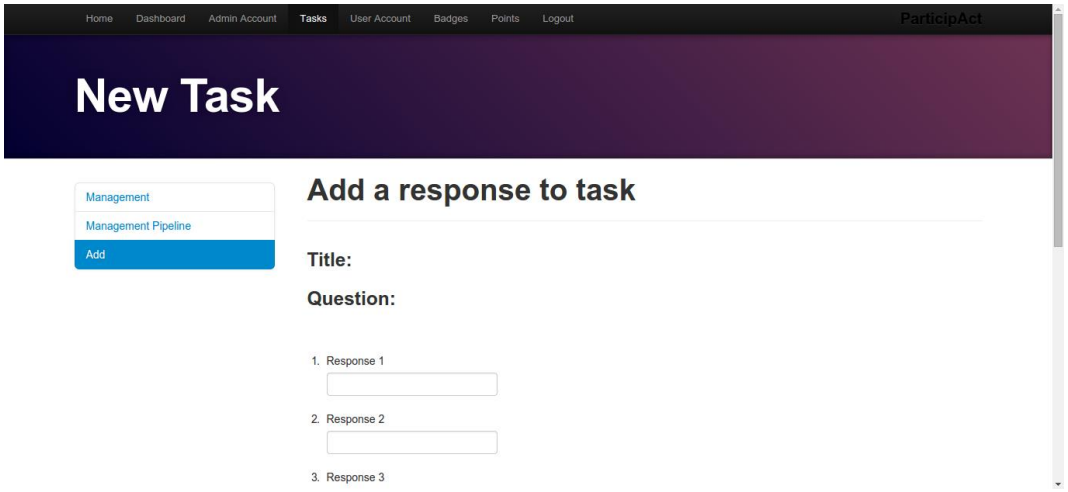

Figura 56: Inserimento delle Risposte

Per l'inserimento delle risposte di ogni step è stata creata la jsp *protected/task/add/actionPipelineTaskChooseResponse.jsp* visibile nella Figura 56. Va ricordato che i campi che vengono lasciati vuoti non vengono riconosciuti come step dal sistema, ma vengono esclusi dalla pipeline.

L'ultima modifica effettuata al webflow riguarda la parte per l'assegnazione di una nuova strategia di gamification dedicata ai gruppi di cui parleremo in seguito. La Figura 57 mostra la scelta della strategia di gruppo.

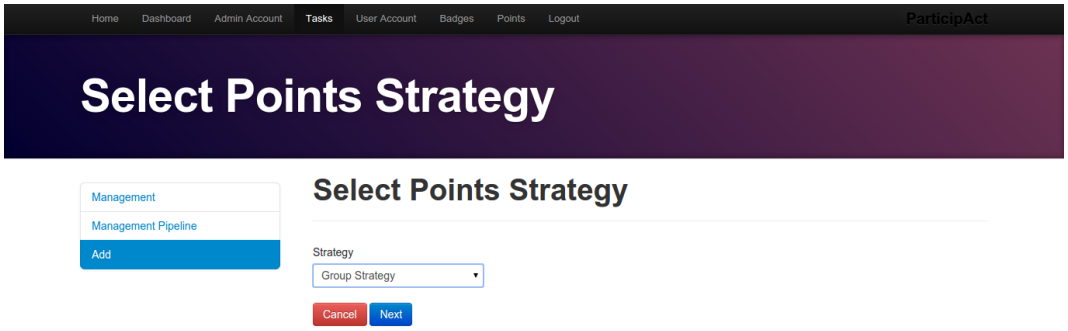

Figura 57: Scelta della strategia di gruppo per la Gamification

6.4 Strategia di Gamification per i Gruppi

La gamification è uno degli aspetti fondamentali per riuscire ad aumentare il tempo di utilizzo di un'applicazione. Consiste principalmente nel mettere in competizione i vari utenti attraverso delle classifiche in modo che la volontà di primeggiare spinga l'utente ad utilizzare sempre più l'applicazione per guadagnare più punti possibili.

Nel progetto ParticipAct la parte relativa alla Gamification è già stata sviluppata, ma si sono rese obbligatorie alcune modifiche per poter creare una nuova strategia di gamification adatta ad assegnare punti ad un gruppo di persone e non ad un singolo individuo. La logica di gamification per i gruppi è contenuta nella classe *it/unibo/paserver/gamificationlogic/GroupStrategy.java* che implementa l'interfaccia *it/unibo/paserver/gamificationlogic/PointsStrategy.java*.

Come già detto in precedenza, in un gruppo ci sono utenti maggiormente attivi che, quindi, devono essere maggiormente gratificati rispetto ad utenti che a mala pena partecipano alle uscite di gruppo.
Si è quindi scelto di attribuire agli utenti che completano il task un punteggio maggiore e agli altri utenti un punteggio minore. In questo modo tutto gli utenti del gruppo possono salire nella classifica generale, ma gli utenti che completano il task la scalano più velocemente.

I punti che mediamente vengono attribuiti per ogni step sono:

TIPO INDIVIDUO	PUNTI
UTENTE ATTIVO	25
UTENTE PASSIVO	5

6.5 Il Task sul Client

Il Client presenta nativamente una struttura capace di ospitare il nuovo Task per cui si è deciso di implementare i singoli step del Task Comunitario come singoli Task. La figura 58 mostra il dialogo tra Client e Server durante lo svolgimento del Task.

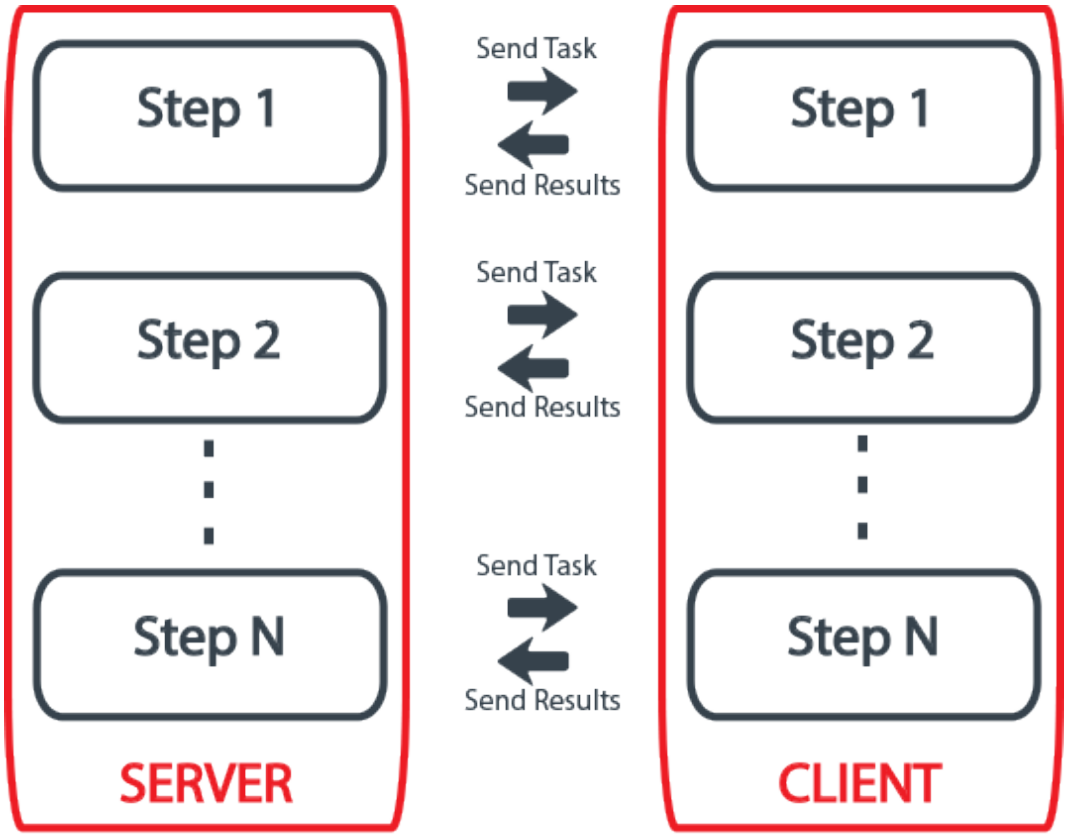

Figura 58: Dialogo tra Server e Client durante il Task

La parte più importante di questo lavoro rimane, però, quella di coinvolgere l'utente nella risoluzione del nuovo Task. Ogni utente deve sentirsi parte attiva del gruppo sia nel caso in cui completi attivamente uno step del Task e sia nel caso in cui partecipi passivamente alla risoluzione dello step.

Per realizzare ciò si rendono necessarie due azioni importanti:

- avvisare il gruppo dell'attuale step in esecuzione, in modo che tutti i membri possano conoscere l'avanzamento del gruppo;
- porre in modo chiaro la domanda agli utenti che desiderano partecipare attivamente alla risoluzione del Task.

Per quanto riguarda il primo punto si è scelto di inserire lo step corrente all'interno del *Titolo* del Task in modo che l'utente possa accorgersi immediatamente di esso.

La Figura 59 mostra il Task Comunitario nel momento in cui diventa disponibile al gruppo specificando l'attuale step di progressione.

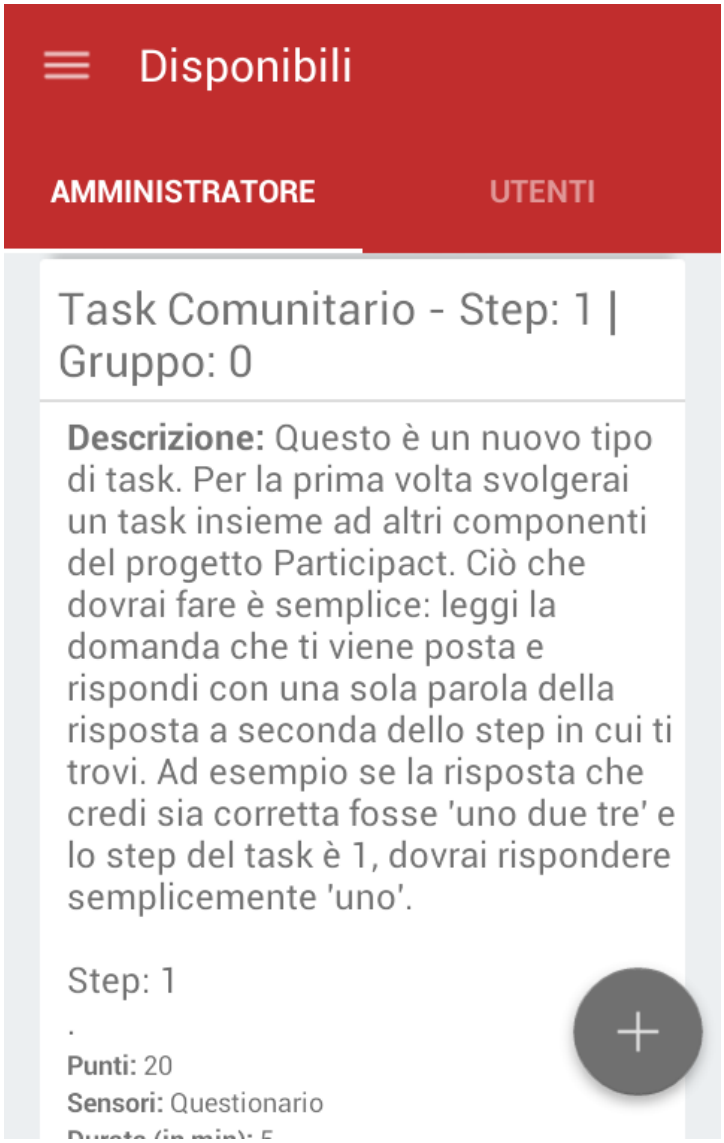

Figura 59: Disponibilità Task Comunitario

Dopo aver completato uno step, l'applicazione provvederà a notificare il Server dell'avvenuto completamento in modo che il prossimo step diventi subito disponibile per gli altri membri del gruppo.

Le risposte ai precedenti step vengono mostrate all'utente solamente nel momento in cui decide di *compilare* l'attuale step. Ad esempio se la domanda al Task è "*Quali sono le prime 3 classificate del mondiale di calcio del 2006 in ordine dalla prima alla terza?*" e le risposte corrette sono "*Italia*", "*Francia*" e "*Germania*", all'utente che deciderà di compilare il terzo step verranno comunicate le precedenti risposte come mostrato in Figura 60.

Figura 60: Visualizzazione degli Step

Capitolo 7: Risultati Sperimentali

Dopo aver illustrato le funzionalità introdotte al fine di aumentare la partecipazione attiva degli utenti verranno illustrati in questo capitolo i risultati ottenuti. Si inizierà analizzando i dati specifici degli elementi introdotti, ossia il calcolo dei gruppi, per poi concludere con una visione globale dei dati raccolti attraverso il nuovo Task comunitario.

7.1 Piano di Test

Nell'analizzare le nuove funzioni relative ai gruppi è stato preso in considerazione un periodo della durata di circa un mese, dove gli utenti hanno raccolto un'elevata quantità di *DataLocation*, mentre per l'analisi delle risposte del nuovo tipo di Task è stato considerato un periodo di circa 10 giorni dalla partenza del Task in modo da fornire a tutti il tempo necessario per completare i vari step.

I dati considerati per il calcolo dei gruppi consistono in circa 7 milioni di tuple (il numero elevato è giustificato dalla volontà di avere una precisione elevata per il calcolo).
Per il Task inviato ai nuovi gruppi, invece, è stata applicata una deadline di 24 ore e una durata di 10 minuti.

Sono stati inviati 4 Task con diverso numero di step per mettere in evidenza la capacità dei gruppi di completare il nuovo Task. In particolare è stato scelto di inviare un primo Task con 3 step, un secondo Task con 2 step, un terzo Task con 4 step e un quanto Task con 5 step.

Nelle prossime sezioni verranno illustrati prima i risultati ottenuti attraverso il calcolo dei gruppi e, successivamente, i risultati di ogni singolo Task.

7.2 Gruppi

Il primo aspetto che andremo ad analizzare riguarda i risultati del calcolo dei gruppi geolocalizzati. Questo calcolo permette di formare per ogni utente del progetto ParticipAct quattro gruppi: *Friends*, *Community*, *Familiar Strangers* e *Strangers*. Ogni gruppo possiede le caratteristiche descritte nei precedenti capitoli.

Per l'analisi si è scelto di considerare un periodo di circa un mese in modo tale da riuscire ad analizzare un numero elevato di dati. Nel periodo considerato sono stati analizzati circa 7 milioni di *DataLocation* e sono stati riscontrati, dopo aver escluso circa la metà dei dati di partenza in quanto poco significativi, un numero di contatti tra gli utenti pari a 41570. Grazie a questi contatti siamo riusciti a formare per ogni membro del progetto i propri gruppi di appartenenza. Per mostrare la significatività di questi dati analizzeremo i gruppi di un singolo utente.

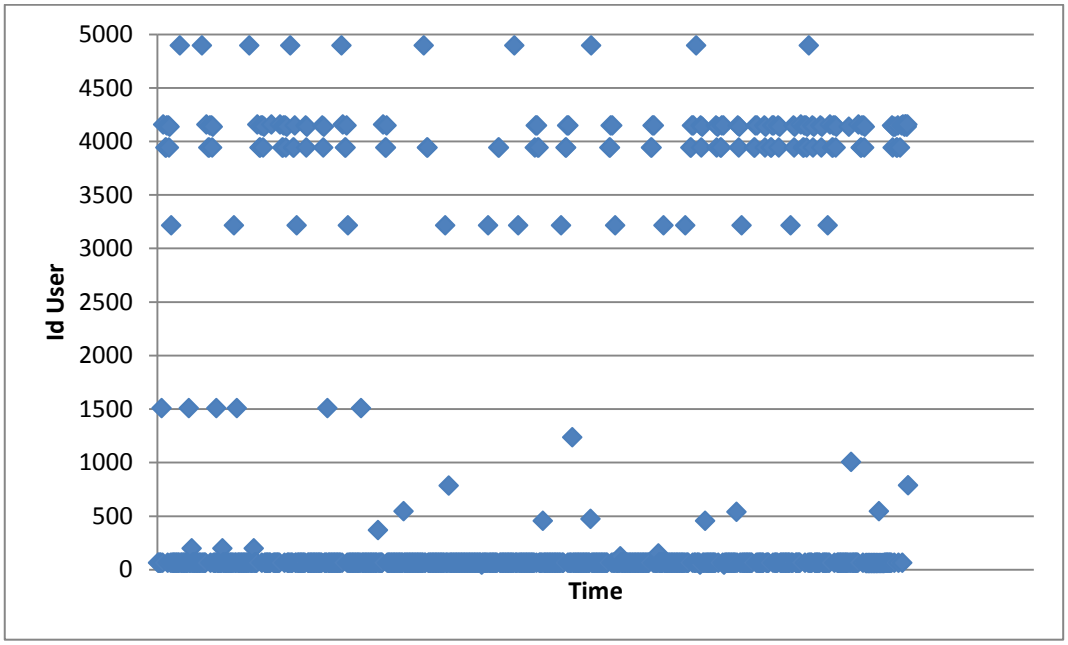

Figura 61: Contatti per un Singolo Utente

Nella Figura 61 sono mostrati i contatti di un singolo utente; in ascissa viene mostrato il periodo considerato, mentre, in ordinata viene mostrato l'ID dell'utente con cui ha avuto un contatto. Ogni punto del grafico rappresenta un singolo contatto tra due utenti. Si può subito notare la presenza di linee quasi continue, che rappresentano un numero di contatti molto elevato tra i due utenti, e di zone del grafico dove la densità dei punti è quasi nulla, che rappresentano contatti sporadici tra vari utenti.

Dopo l'applicazione dell'algoritmo *k-CLIQUE* sono stati separati gli utenti in gruppi come mostrato nella Figura 62.

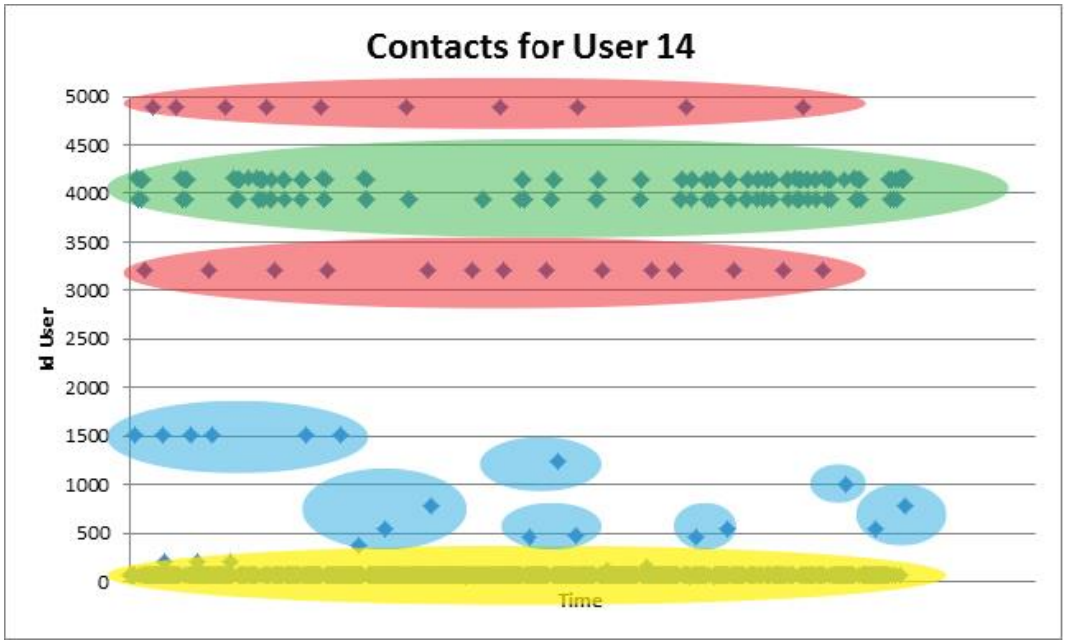

Figura 62: Risultati dopo l'applicazione dell'algoritmo k-CLIQUE

Nella Figura 62 sono state evidenziate le aree di maggiore interesse:

- area **GIALLA**: i contatti che sono stati riconosciuti nel gruppo **FRIENDS** (quindi le persone con cui un utente ha avuto molti contatti che durano per più di un'ora);
- area **VERDE**: i contatti che sono stati riconosciuti nel gruppo **COMMUNITY** (quindi le persone con cui un utente ha avuto un discreto numero di contatti dilazionati nel tempo);
- area **ROSSA**: i contatti che sono stati riconosciuti nel gruppo **FAMILIAR STRANGERS** (quindi le persone con cui un utente ha avuto un discreto numero di contatti che durano per più di un'ora);
- area **CELESTE**: i contatti che sono stati riconosciuti nel gruppo **STRANGERS** (quindi le persone con cui un utente ha avuto pochi contatti).

Per completezza ora descriveremo le singole aree in dettaglio, in modo da mostrare la significatività dei risultati ottenuti. Il precedente grafico verrà analizzato dal basso verso l'alto.

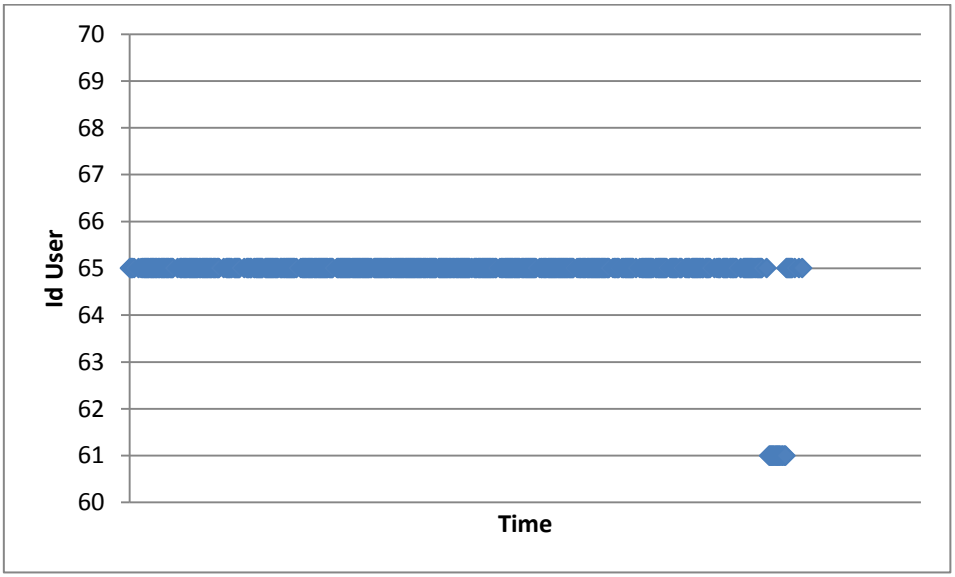

Figura 63: Analisi dei gruppi 1

Dalla Figura 63 possiamo evidenziare due aspetti:

- L'utente 65 è stato riconosciuto come componente del gruppo *FRIENDS*, infatti si può notare come nell'arco di circa un mese i due utenti abbiano contatti periodicamente.
- L'utente 61, pur avendo contatti continuativi per un periodo, non è stato riconosciuto come componente del gruppo *FRIENDS*, ma come componente del gruppo *COMMUNITY*, perché l'arco di tempo dove si sono verificati i contatti è troppo breve rispetto al periodo preso in esame.

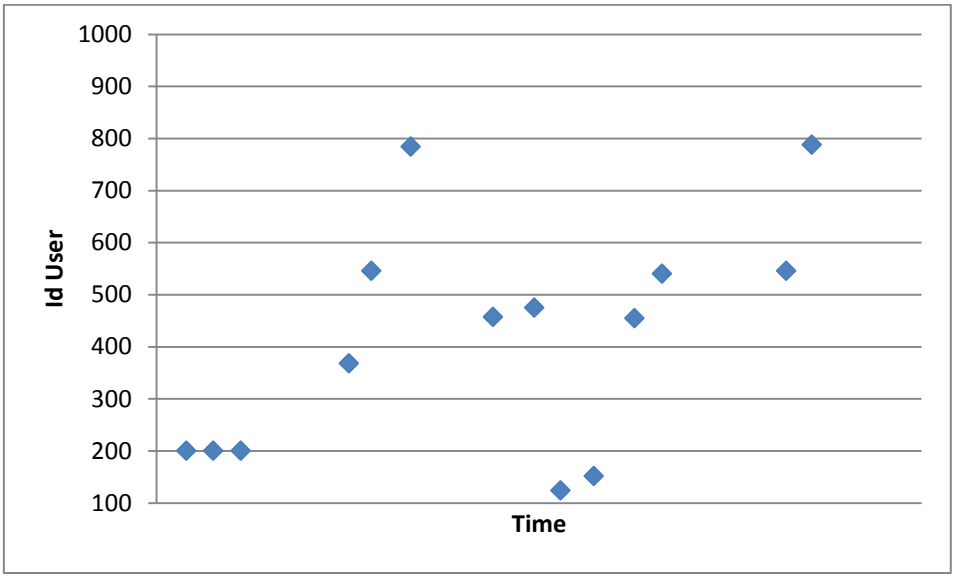

Figura 64: Analisi dei gruppi 2

Dalla Figura 64 si va evidenziato che gli utenti presenti in quest'area sono stati riconosciuti come componenti del gruppo *STRANGERS* in quanto i loro contatti sono sempre singoli e dilazionati nel tempo.

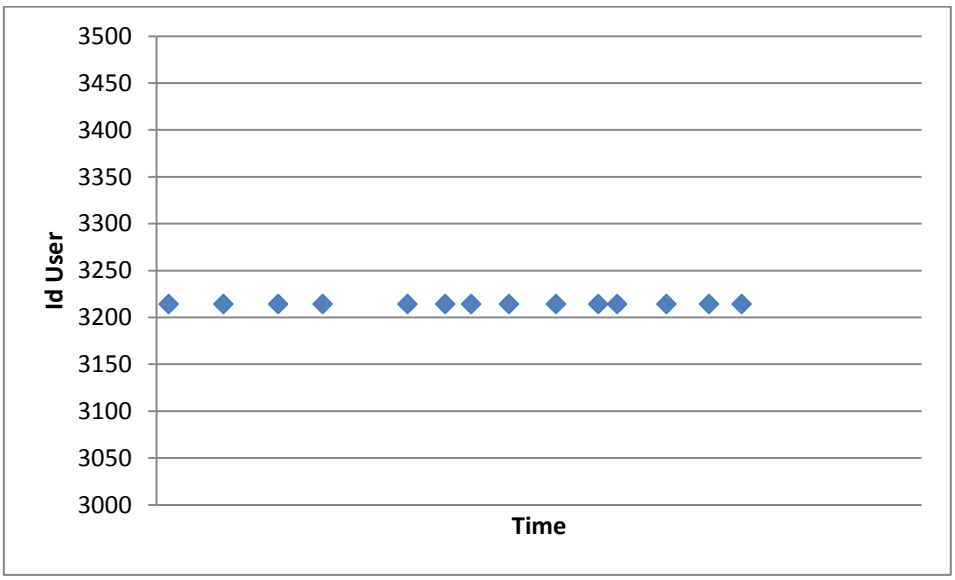

Figura 65: Analisi dei gruppi 3

L'utente presente nella Figura 65 è stato riconosciuto come componente del gruppo *FAMILIAR STRANGERS* in quanto i contatti

sono dilazionati nel tempo e non hanno una frequenza elevata tale da essere considerato come membro del gruppo *COMMUNITY*.

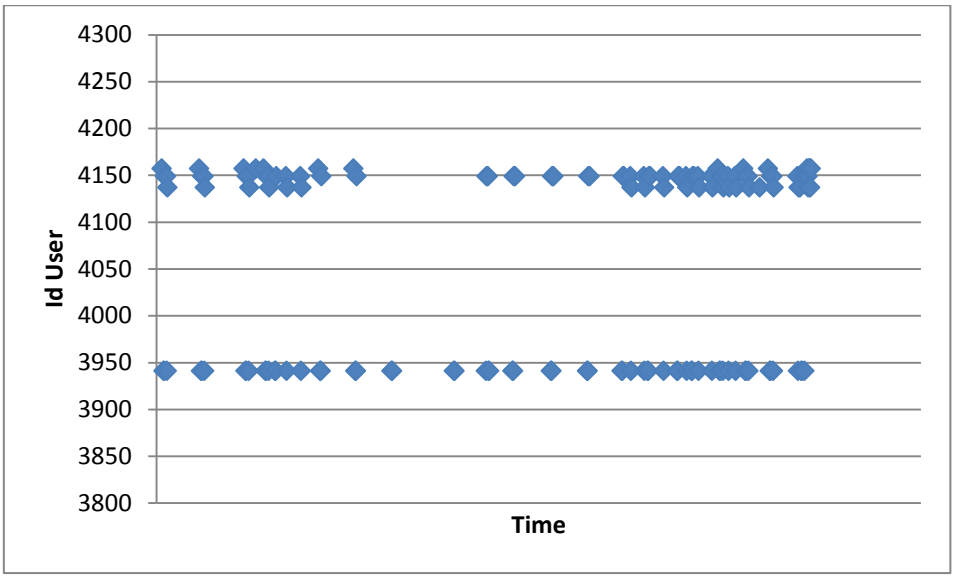

Figura 66: Analisi dei gruppi 4

La Figura 66 evidenzia utenti che sono stati riconosciuti come componenti del gruppo *COMMUNITY*, infatti si può notare come i punti del grafico sono molto più fitti di quelli in Figura 58, quindi hanno avuto molti contatti nel tempo e alcuni di essi sono continuativi.

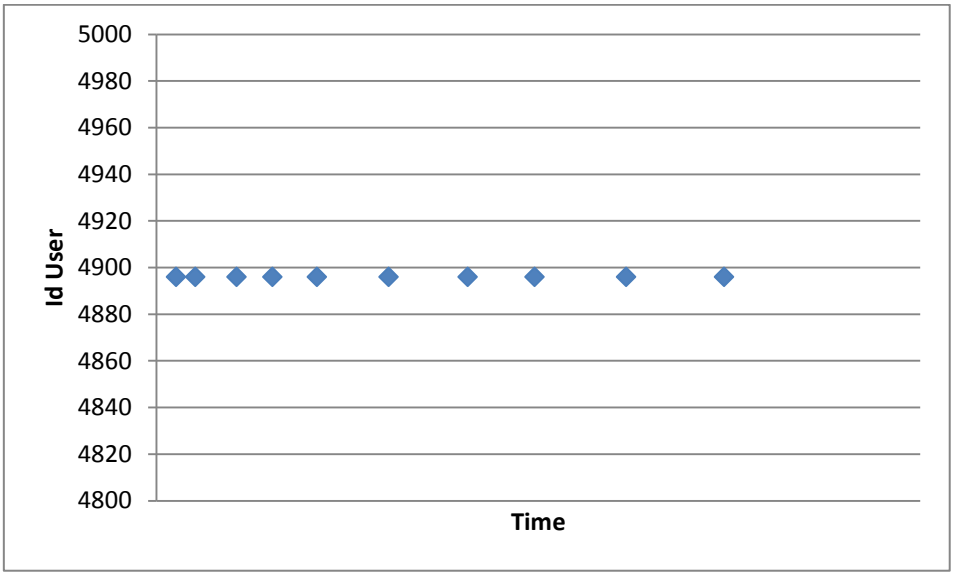

Figura 67: Analisi dei Gruppi 5

L'utente presente nella Figura 67 è stato riconosciuto come componente del gruppo *FAMILIAR STRANGERS* in quanto i contatti sono dilazionati nel tempo e non hanno una frequenza elevata tale da essere considerato come membro del gruppo *COMMUNITY*.

7.2.1 Legame tra Gruppi e Task Comunitario

I calcolo dei gruppi e il nuovo Task hanno, fin da subito, dato risultati significativi al progetto ParticipAct. Gli utenti hanno mostrato grande interesse verso il lavoro di gruppo completando più della metà degli step previsti.
La risoluzione degli step del Task comunitario ha mostrato un legame tra gli utenti che hanno completato il singolo step e i gruppi utente di appartenenza. La Figura 68 mostra il legame tra gli utenti che hanno completato i singoli step e i gruppi utente di appartenenza. In particolare:

- Il 46% dei partecipanti faceva parte di un gruppo *FRIENDS*;
- Il 31% dei partecipanti faceva parte di un gruppo *COMMUNITY*;
- Il 15% dei partecipanti faceva parte di un gruppo *FAMILIAR STRANGERS*;
- L'8% dei partecipanti faceva parte di un gruppo *STRANGERS*.

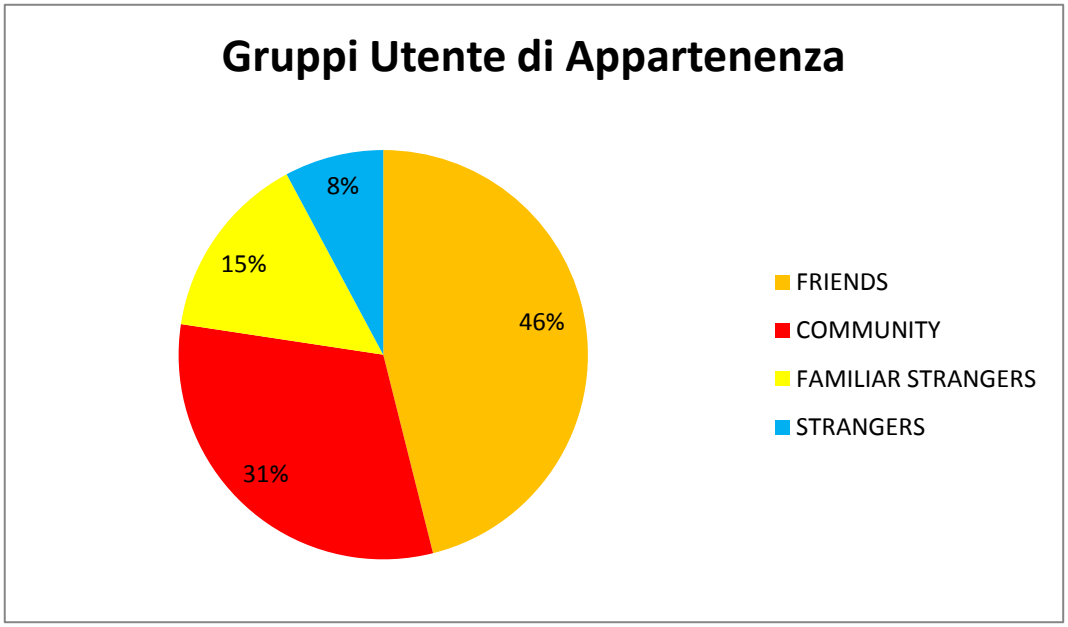

Figura 68: Gruppi Utente di Appartenenza

7.3 Risultati del nuovo Task

Per lo svolgimento del nuovo tipo di Task è stato deciso di calcolare i gruppi in un periodo di circa un mese. Dopo l'analisi sono stati coinvolti 37 gruppi di utenti per lo svolgimento di 4 Task.

Va ricordato che nel periodo precedente alle novità introdotte i Task venivano:

- Per il 38,05% accettati;
- Per il 7,10% rifiutati;
- Per il 54,85% ignorati.

La Figura 69 mette in evidenza le percentuali di accettazione prima dello sviluppo dei gruppi e del Task Comunitario.

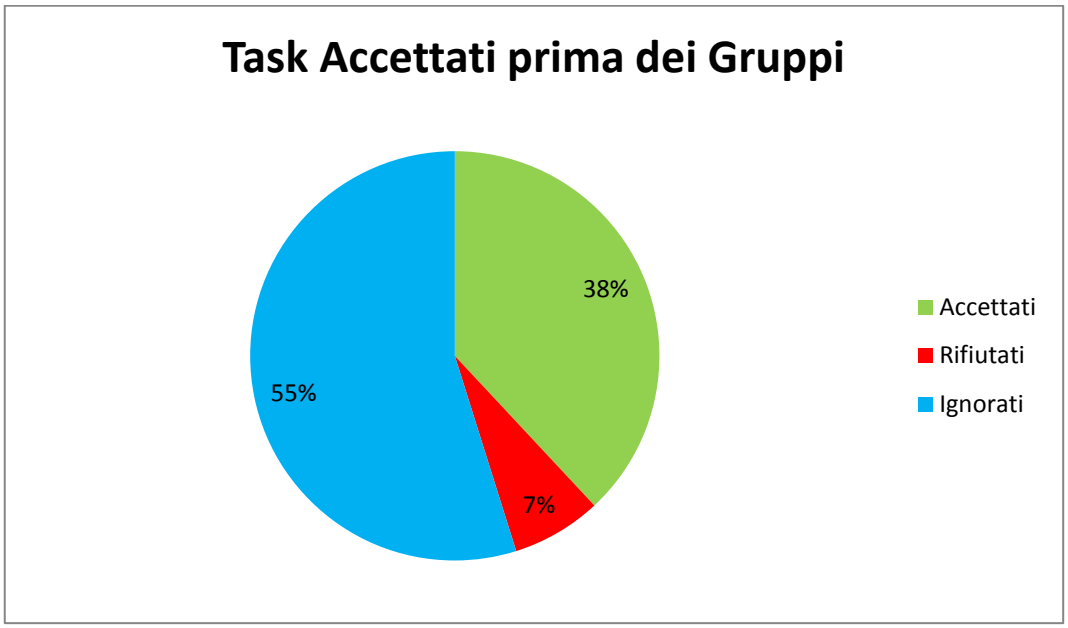

Figura 69: Percentuale di Accettazione prima del Nuovo Task

Il nuovo tipo Task è stato testato con l'invio di 4 domande a 37 gruppi. I risultati ottenuti hanno mostrato un miglioramento rispetto alle precedenti percentuali. In particolare:

- Per il 51% accettati;
- Per il 4% rifiutati;
- Per il 45% ignorati.

La Figura 70 mostra le percentuali di accettazione del nuovo Task. Si può notare un incremento di circa 12 punti percentuali dei Task accettati a fronte di una diminuzione di circa 3 punti percentuali dei Task rifiutati e di circa 10 punti percentuali dei Task ignorati.

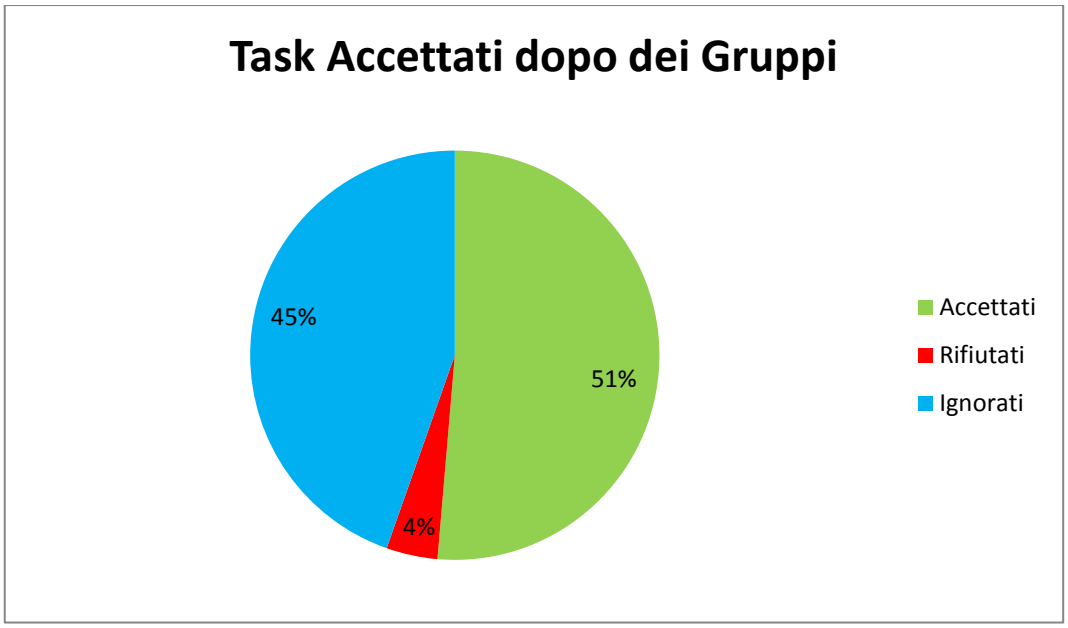

Figura 70: Percentuale di Accettazione dopo del Nuovo Task

Nel complesso le nuove tecniche introdotte, il raggruppamento degli utenti e il nuovo Task comunitario, hanno portato ad un incremento della partecipazione in punti percentuali.

Considerando, però, il numero assoluto degli utenti partecipanti al Task possiamo notare che l'aumento della percentuale è dovuto al fatto che non tutti gli utenti sono presenti nei gruppi in quanto molti studenti che partecipano al progetto non sono residenti nel comune di Bologna o non hanno raccolto una quantità di *DataLocation* da essere presa in considerazione. Nello specifico dobbiamo paragonare il 38% calcolato su 200 studenti (quindi 76 studenti) con il 51% dei 169 studenti che sono stati coinvolti nei gruppi (quindi 85 studenti). La Figura 71 mostra come il reale aumento è di circa 10 studenti che in percentuale rappresenta il 5% sul totale dei partecipanti al progetto.

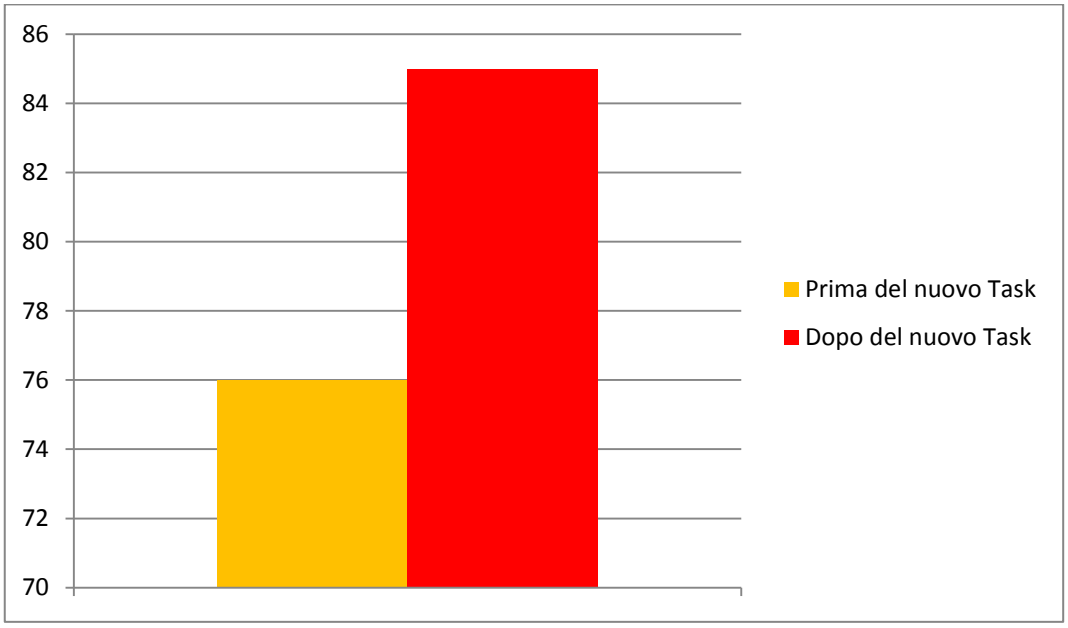

Figura 71: Comparazione sul Numero Assoluto

Di seguito vengono mostrate le domande poste nei 4 Task inviati con le relative risposte. Ogni Task presenta un numero diverso di step, ma, per ognuno di essi, è stata scelta una deadline di circa 24 ore e una durata di circa 10 minuti.

Le domande che sono state poste con le relative risposte sono:

1. *Quali sono le prime 3 classificate del mondiale di calcio del 2006 in ordine dalla prima alla terza?*
    a. *Italia*
    b. *Francia*
    c. *Germania*
2. *Chi è l'attuale presidente della Repubblica Italiana? Specificare prima il nome e dopo il cognome.*
    a. *Sergio*
    b. *Mattarella*
3. *Quali sono le prime 4 note musicali?*
    a. *Do*
    b. *Re*
    c. *Mi*
    d. *Fa*
4. *Qual è il titolo della canzone nota anche come Volare, scritta nel 1958 da Franco Migliacci e Domenico Modugno e da quest'ultimo originariamente interpretato?*
    a. *Nel*
    b. *Blu*
    c. *Dipinto*
    d. *Di*
    e. *Blu*

7.3.1 Primo Task

Al primo Task, che poneva la domanda: *"Quali sono le prime 3 classificate del mondiale di calcio del 2006 in ordine dalla prima alla terza?"* e prevedeva le risposte *"Italia", "Francia" e "Germania"*, hanno partecipato 37 gruppi.

La domanda posta metteva, quindi, gli utenti di fronte ad un quesito di genere sportivo, ma permetteva a tutti di rispondere facilmente data la diffusione dell'argomento richiesto.

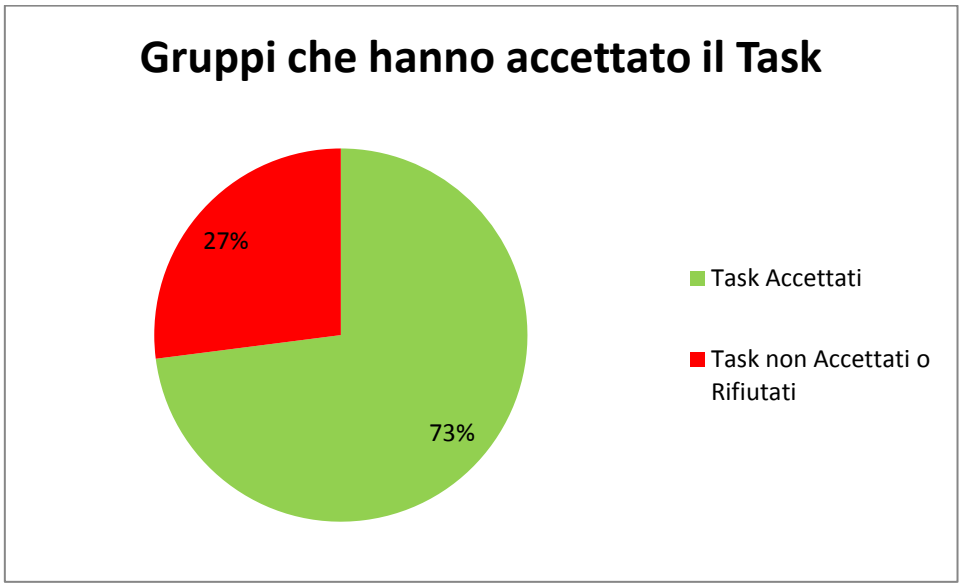

Figura 72: Accettazione del Primo Task

La Figura 72 mostra le percentuali dei gruppi che hanno accettato il primo Task. Si può notare come il 68% dei gruppi equivale a 25 gruppi su 37, quindi molto più della metà dei gruppi ha deciso di partecipare al nuovo Task.

Le risposte che sono state fornite ai vari step hanno permesso di capire quanti gruppi sono formati da persone che si interessano al mondo del calcio e hanno mostrato che un numero abbastanza elevato di

individui ha fornito le risposte corrette.
La Figura 73 vuole mettere in risalto che su 25 gruppi che hanno accettato il Task il 20% ha completato con successo i tre step previsti, il 34% si è fermato al secondo step e il 46% non è andato oltre il primo step previsto.

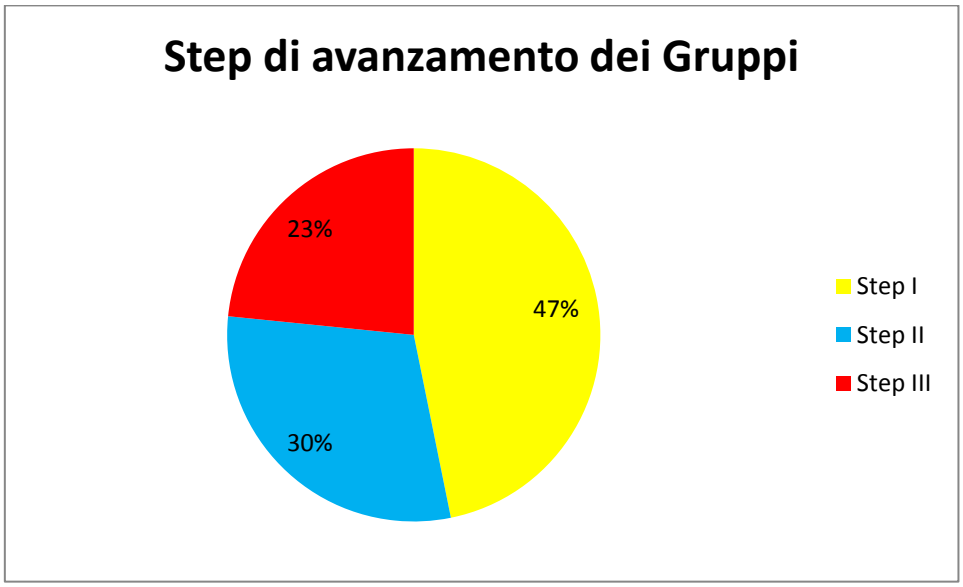

Figura 73: Avanzamento dei Gruppi nel Primo Task

7.3.2 Secondo Task

Al secondo Task, che poneva la domanda: *"Chi è l'attuale presidente della Repubblica Italiana? Specificare prima il nome e dopo il cognome"* e prevedeva le risposte *"Sergio" e "Mattarella",* hanno partecipato 37 gruppi.

Questa domanda, a differenza della prima, si incentrava su un argomento di cultura generale, dove chiunque ha potuto fornire facilmente la risposta.

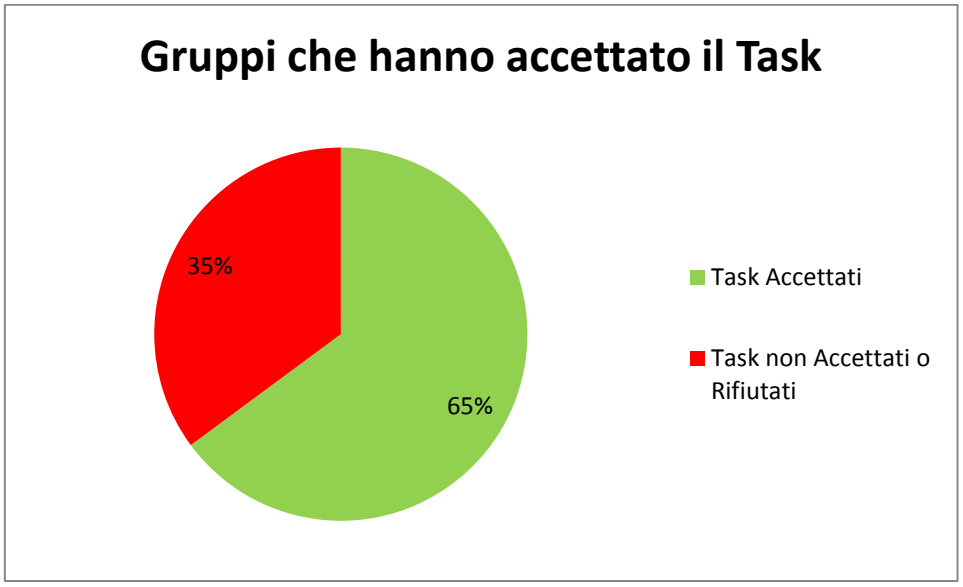

Figura 74: Accettazione del Secondo Task

La Figura 74 mette in risalto che, a differenza del primo Task, al secondo hanno partecipato il 65% dei gruppi.
La Figura 75 mostra i risultati ottenuti dai 24 gruppi che hanno accettato il secondo Task. Il 63% ha completato solamente il primo step, mentre il 37% ha completato entrambi gli step previsti.

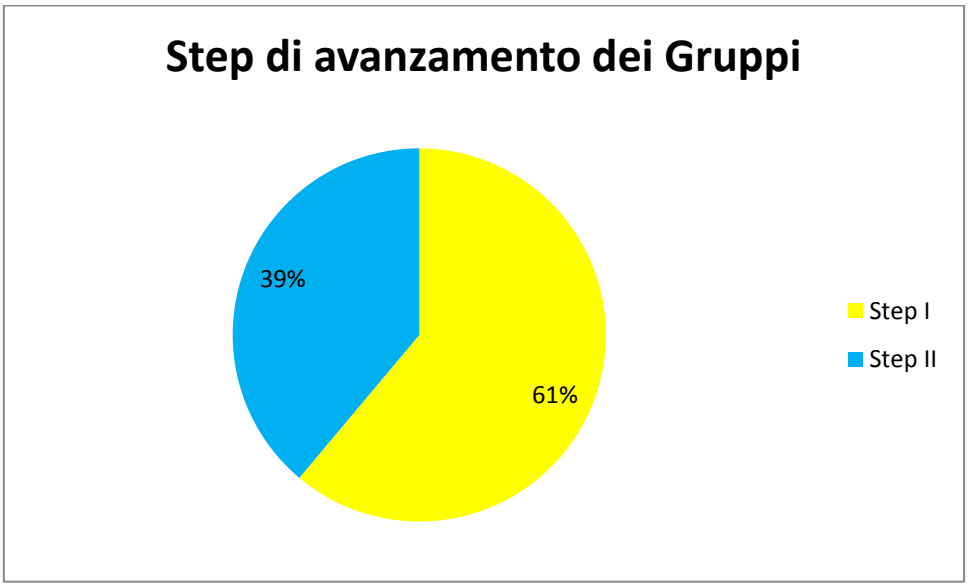

Figura 75: Avanzamento dei Gruppi nel Secondo Task

Nel secondo Task, si può notare un forte aumento sulla percentuale dei gruppi che hanno completato tutti gli step del Task. Questo, probabilmente, è dovuto alla forte diffusione dei media e alla maggior vicinanza temporale della notizia.

7.3.3 Terzo Task

Il terzo Task prevedeva la domanda *"Quali sono le prime 4 note musicali?"* e le risposte *"Do", "Re", "Mi" e "Fa"*. Hanno partecipato al Task 37 gruppi.
Il Task è stato incentrato sul genere musicale, ma si è preferito mantenere la domanda generica in modo da fornire a tutti gli utenti la possibilità di rispondere.

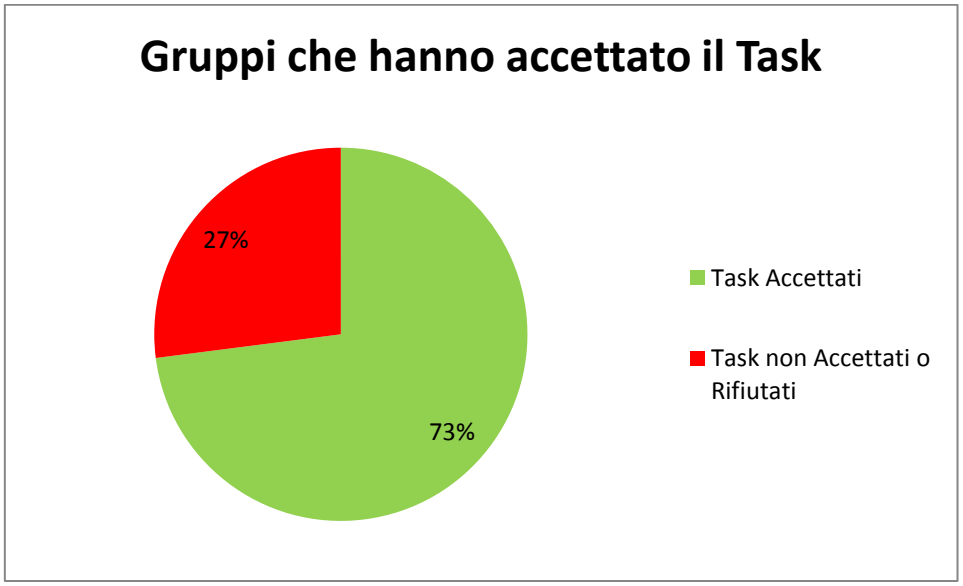

Figura 76: Accettazione del Terzo Task

La Figura 76 mostra le percentuali di accettazione del terzo Task. In particolare si può notare come ben il 73% dei gruppi ha accettato il Task.

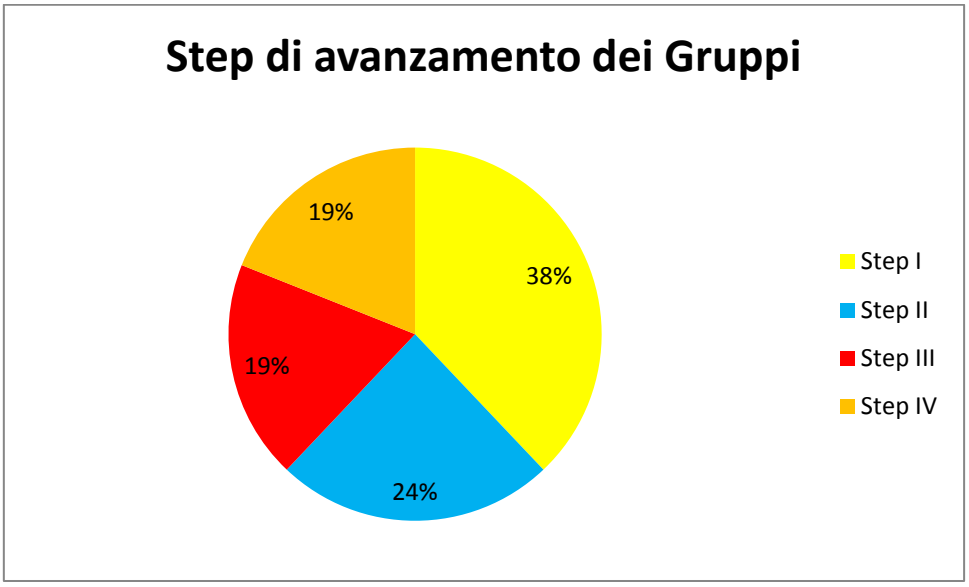

Figura 77: Avanzamento dei Gruppi nel Terzo Task

Nella Figura 77 notiamo come al terzo Task, che prevedeva la risoluzione di ben 4 step, il 38% dei gruppi è rimasto fermo al primo step, il 24% al secondo step, il 19% al terzo step e il 19% ha completato tutti e quattro gli step. Il numero di gruppi che ha completato tutti i 4 step previsti è, in percentuale, inferiore rispetto ai primi due Task. Il motivo di ciò è da ricercare, quasi certamente, nel maggior numero di step presenti.

7.3.4 Quarto Task

Il quarto Task prevedeva la domanda "*Qual è il titolo della canzone nota anche come Volare, scritta nel 1958 da Franco Migliacci e Domenico Modugno e da quest'ultimo originariamente interpretato?*" e le risposte "*Nel*", "*blu*", "*dipinto*", *"di" e "blu"*.
Hanno partecipato al Task 37 gruppi.
Il Task è stato incentrato sul genere musicale, ma si è preferito mantenere la domanda generica in modo da fornire a tutti gli utenti la possibilità di rispondere.

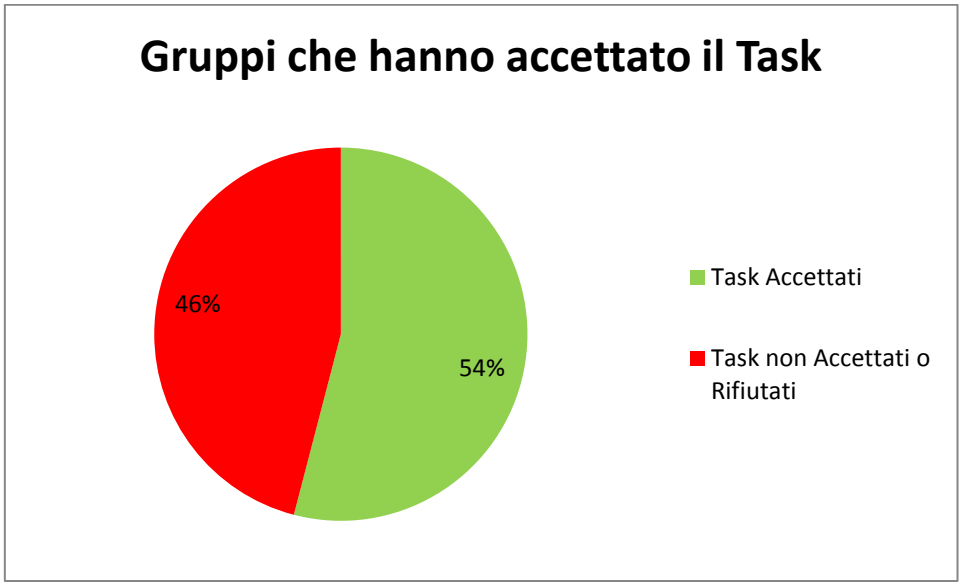

Figura 78: Accettazione del Quarto Task

La Figura 78 mostra le percentuali di accettazione del quarto Task. Si può notare come solamente il 54% dei gruppi ha preso parte al Task. Nella Figura 79, invece, viene messo in evidenza lo step di avanzamento dei gruppi nel quarto Task.

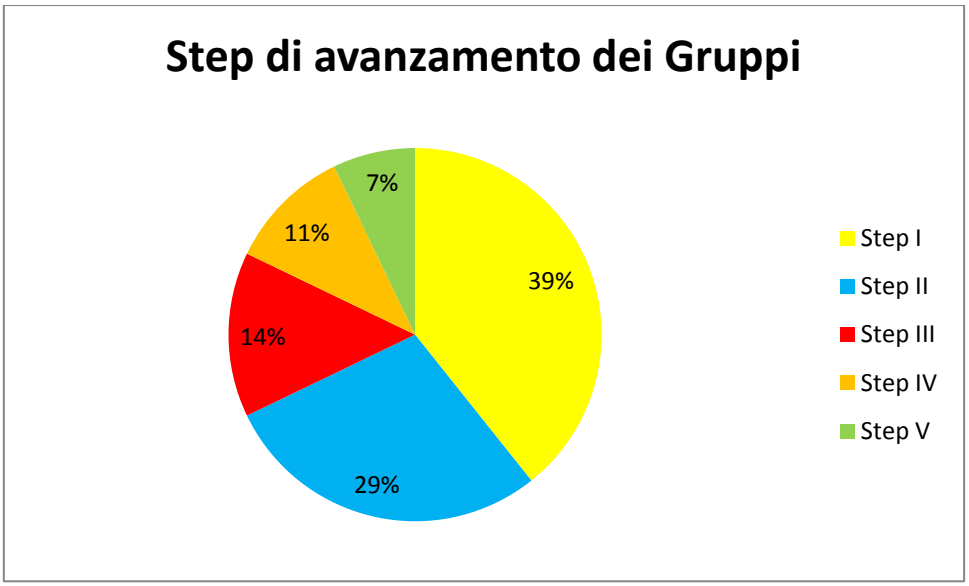

Figura 79: Avanzamento dei Gruppi nel Quarto Task

7.3.5 Considerazioni sul nuovo Task

Il nuovo Task ha immesso all'interno del progetto ParticipAct il nuovo concetto di Gruppo presentandolo agli utenti sotto forma di piccole azioni da completare per raggiungere un obbiettivo comune.

Gli utenti hanno accolto questo modo di vivere il Crowdsensing con molto interesse. Mediamente più della metà dei gruppi ha preso parte al nuovo Task (vedi Figura 80).

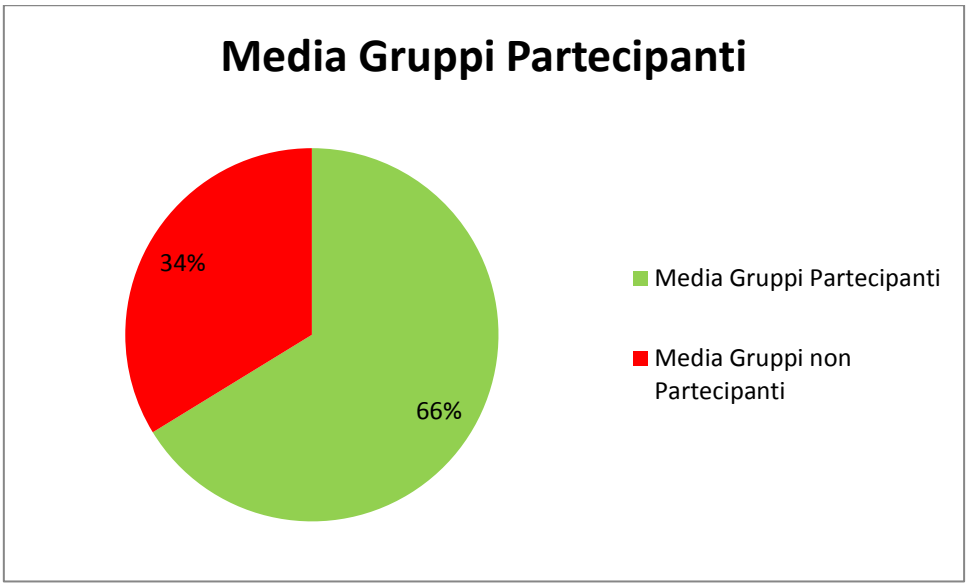

Figura 80: Media Gruppi Partecipanti

Altra importante considerazione deve essere fatta sul numero di step ideale per rendere possibile alla maggior parte dei gruppi il completamento del Task. La Figura 81, invece, mette in evidenza la percentuale di completamento per ogni singolo step. Si può subito notare che, in media, più il Task è formato da un numero minore di step e più è probabile che il gruppo lo porti a termine. È evidente, quindi, che per avere un Task di gruppo con una durata e una percentuale di completamento accettabili bisogna prevedere un numero di step pari a 3 in modo da permettere ad una buona parte dei gruppi di raggiungere l'obiettivo.

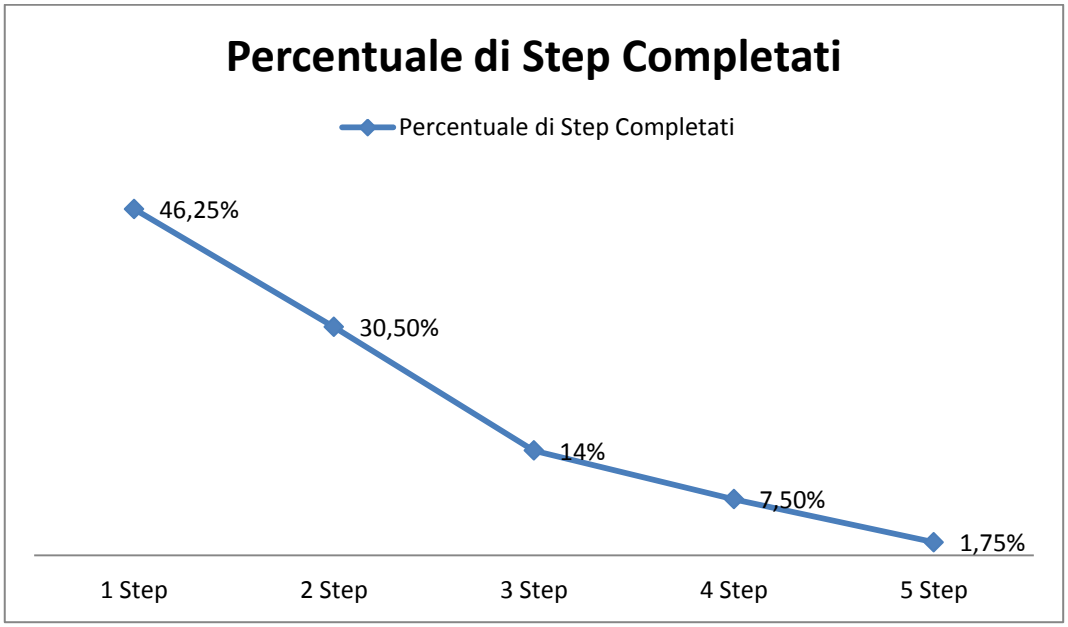

Figura 81: Percentuale di Completamento degli Step

Conclusioni e Sviluppi futuri

Nel mondo moderno la diffusione di nuovi dispositivi con sensori integrati offre importanti vantaggi per la raccolta dei dati all'interno di una Smart City. Basti pensare che la raccolta stessa viene effettuata direttamente dai cittadini e ciò porta a risparmi significativi in quanto non va prevista l'installazione di una infrastruttura ad hoc.

Gli utenti che partecipano alla raccolta dei dati non sono, però, sempre disposti a fornire informazioni sul proprio stile di vita o sui propri spostamenti. Si rende, quindi, necessaria un'azione di incentivazione verso la massa di cittadini interessata. In questo lavoro di tesi si è cercato di invogliare i partecipanti del progetto ParticipAct attraverso la costituzione di gruppi di persone che potessero collaborare tra di loro per raggiungere un obiettivo comune.

Per poter formare i gruppi ci si è basati sui dati della localizzazione raccolti in un periodo di circa un mese in modo da avere dei gruppi attendibili dal punto di vista dei reali contatti tra persone. L'analisi ha subito dimostrato che gli studenti appartenenti alla stessa facoltà universitaria molto spesso si ritrovano per studiare insieme e, soprattutto, ha evidenziato che gli individui appartenenti a diverse facoltà si incontrano al di fuori dell'ambito universitario per lo più nel fine settimana.

Per la valutazione del lavoro di gruppo si è deciso di far risolvere semplici quesiti in un periodo di circa una settimana. I dati raccolti hanno mostrato, fin da subito, la presenza di alcuni gruppi che hanno completato i vari step del task con successo, ma il risultato più importante è che il nuovo Task mostra che i partecipanti attivi sono distribuiti all'interno del gruppo e non è sempre la stessa persona a compiere il lavoro, creando così, all'interno dei gruppi, una condivisione dell'obiettivo finale tra tutti i componenti.

Nella valutazione dei risultati si è notato, però, che con l'aumentare del numero degli step, sempre meno gruppi portavano a termine il Task. All'aumentare di ogni step, nella maggior parte dei casi, meno della metà dei gruppi continuava a partecipare al lavoro. Per questo motivo si è preferito stimare a 3 il numero di step consigliato per permettere il completamento del Task ad una quantità considerevole di gruppi.

In futuro, i gruppi creati potrebbero essere sfruttati per poter favorire la comunicazione tra i vari utenti permettendo la condivisione delle abitudini giornaliere con i propri amici oppure si potrebbe permettere la creazione di azioni condivise dagli utenti dei gruppi. Altro possibile scenario potrebbe riguardare la possibilità di richiedere aiuto ai propri amici per il completamento di un task, ad esempio si potrebbe chiedere ad un amico di scattare una foto ad un monumento se si è fisicamente lontani dallo stesso.

Un ulteriore punto di estensione potrebbe essere visto nella volontà di aumentare la mobilità degli utenti facendoli partecipare a raccolte di premi virtuali precedentemente posizionati in punti ben precisi della Smart City. In questo modo si potrebbe aumentare la quantità di dati raccolti in aree dove gli individui tendono a passare maggiormente il proprio tempo.

In un altro scenario si potrebbe delegare ai gruppi la scelta di un amministratore per la creazione di azioni coordinate tra i membri dello stesso gruppo in modo da rafforzare il legame già esistente tra loro. Ad esempio di potrebbe eleggere un rappresentante per ogni gruppo e permettergli di posizionare premi sulla mappa della Smart City in modo che gli altri membri possano partecipare alla raccolta degli stessi rafforzando il proprio legame attraverso una sana competizione.

Bibliografia

1: Nicholas D. Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, Andrew T. Campbell, A Survey of Mobile Sensing, 2010
2: Samsung Inc., Samsung Site, , www.samsung.com
3: Gildo Seisdedos, ¿Qué es una Smart City?, 2012
4: A. Thiagarajan, VTrack: Accurate, Energy-AwareTraffic Delay Estimation Using Mobile Phones, 2009
5: S. Consolvo, Activity Sensing in the Wild: A FieldTrial of Ubifit Garden, 2008
6: Min Mun, Sasank Reddy, Katie Shilton, Nathan Yau, Jeff Burke, Deborah Estrin, Mark Hansen, Eric Howard, Ruth West, Péter Boda, PEIR, the personal environmental impact report, as a platform for participatory sensing systems research, 2009
7: Tingxin Yan, Matt Marzilli, Ryan Holmes, Deepak Ganesan, Mark Corner, Demo Abstract: mCrowd - A Platform for Mobile Crowdsourcing, 2009
8: Moo-Ryong Ra, Bin Liu, Tom F. La Porta, Ramesh Govindan, Medusa: a programming framework for crowd-sensing applications, 2012
9: Ramanathan N, Alquaddoomi F, Falaki H, George D, Hsieh C, Jenkins J, Ketcham C, Longstaf B, Ooms J, Selsky J, Tangmunarunkit H, Estrin D, Ohmage: An open mobile system for activity and experience ampling, 2012
10: Min Mun, Sasank Reddy, Katie Shilton, Nathan Yau, Jeff Burke, Deborah Estrin, Mark Hansen, Eric Howard, Ruth West, Péter Boda, PEIR, the personal environmental impact report, as a platform for participatory sensing systems research, 2009
11: Agarwal V, Banerjee N, Chakraborty D, Mittal S, USense - A Smartphone Middleware for Community Sensing, 2013

12:Xiping Hu, Chu T.H.S., Chan H.C.B., Leung V.C.M., Vita: A Crowdsensing-Oriented Mobile Cyber-Physical System, 2013
13:Fang-Jing Wu, Tie Luo, WiFiScout: A Crowdsensing WiFi Advisory System with Gamificationbased Incentive, 2014
14:Ueyama Y., Tamai M., Arakawa Y., Yasumoto K., Back to ResultsGamification-based incentive mechanism for participatory sensing, 2014
15:Talasila M., Curtmola R., Borcea C., Alien vs. Mobile user game: Fast and efficient area coverage in crowdsensing,
16:Rajan Vaish, Keith Wyngarden, Jingshu Chen, Brandon Cheung, Michael S. Bernstein, Twitch crowdsourcing: crowd contributions in short bursts of time, 2014
17:Scuola di Scienze e Tecnologie dell'Informazione dell'Università di Urbino e Associazione Culturale NeuNet, SmartRoadSense, http://smartroadsense.it/
18:Regione Piemonte in collaborazione con CSI-Piemonte, Smartdatanet, http://www.smartdatanet.it/
19:Unibo, ParticipAct, , http://participact.ing.unibo.it/
20:M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, A density-basedalgorithm for discovering clusters in large spatial databases with noise, 1996
21:Giuseppe Cardone, Andrea Cirri, Antonio Corradi, Luca Foschini, Dario Maio, MSF: An Efficient Mobile Phone Sensing Framework, 2012
22:Spring, Spring Dveloper Site, , https://spring.io/
23:Google, Android Developer Site, , http://developer.android.com/
24:Google, Protocol Buffers - Google's data interchange format, , https://github.com/google/protobuf/
25:Pivotal Software, Spring, , https://spring.io/
26:Oracle, Java Persistence API, , http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
27:JBoss (Red Hat), Hibernate, , http://hibernate.org/
28:PostgreSQL Global Development Group, PostgreSQL, , http://www.postgresql.org/

29:Spring, Spring Expression Language (SpEL), , http://docs.spring.io/spring/docs/current/springframeworkreference/html/expressions.html
30:J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, HTTP Authentication: Basic and Digest Access Authentication, 1999, http://tools.ietf.org/html/rfc2617
31:Stephane Nicolas, RoboSpice, , https://github.com/stephanenicolas/robospice
32:Pan Hui, Eiko Yoneki, Shu-Yan, Jon Crowcroft, Distributed Community Detection in Delay Tolerant Networks