Sii-Mobility

Supporto di Interoperabilità Integrato per i Servizi al Cittadino e alla Pubblica Amministrazione

Trasporti e Mobilità Terrestre, SCN_00112

Deliverable ID: DE3.18a
Titolo: Moduli di connessione con SII per mobile, web e applicazioni fisse

Data corrente	M16, Aprile 2017
Versione (solo il responsabile puo' cambiare versione)	0.3
Stato (draft, final)	Publico
Livello di accesso (solo consorzio, pubblico)	Consorzio
WP	WP3
Natura (report, report e software, report e HW, ...)	Report e Software per varie piattaforme: ios, android, etc.
Data di consegna attesa	M16, Aprile 2017
Data di consegna effettiva	M16, Aprile 2017
Referente primario, coordinatore del documento	UNIFI:DISIT
Contributor	Paolo Nesi, Paolo.nesi@unifi.it, Michela Paolucci, michela.paolucci@unifi.it, Imad Zaza, imad.zaza@unifi.it Angelo Difino, angelo.difino@unifi.it Claudio Badii, claudio.badii@unifi.it
Coordinatore responsabile del progetto	Paolo Nesi, UNIFI, paolo.nesi@unifi.it

Sommario

1 Introduzione ed obiettivi

Sii-Mobility intende creare una soluzione che possa abilitare un'ampia gamma di servizi al cittadino e commerciali in connessione e integrati con il sistema di mobilità: collezionando dati puntuali e attualizzati in tempo reale da varie fonti; analizzando i flussi di dati con varie tipologie di algoritmi; producendo azioni e informazioni tramite applicazioni web e mobili, totem informativi, ecc.; mettendo a disposizione dati elaborati e puntuali, che potranno essere usati da PA, gestori, e imprese per produrre servizi più efficaci ed efficienti, e anche nuovi servizi integrati. Inoltre Sii-Mobility permette a PA e PMI di caricare ulteriori algoritmi sul sistema con lo scopo di erogare servizi verso gli utenti finali e verso le PA. Per esempio: algoritmi di routing, di valutazione e predizione di condizioni critiche, di ottimizzazione delle risorse, di personalizzazione dei percorsi, di guida connessa, etc.

1.1.1 Obiettivi

Scopo di questo Deliverable è quello di descrivere il funzionamento dei moduli di connessione di Supporto all'Interoperabilità Integrata per applicazioni mobile, web e fisse. Si procede quindi con: i) la sintesi della architettura generale di SiiMobility: ii) l'analisi dei vari casi d'uso relativi ai sensori e alle applicazioni mobile: iii) la descrizione in dettaglio della infrastruttura (store) che permette la gestione e interconnessione dei dati.

2 Architettura generale e Moduli di connessione con i sensori

In *Figura 1* è illustrato il ruolo dei Sensori e delle applicazioni Mobile e Web, nell'architettura generale di Sii-Mobility.
Le applicazioni Web e Fisse (totem), hanno lo scopo di visualizzare le informazioni gestite dall'intero sistema in base al tipo di utente a cui sono destinate, non inviano pertanto informazioni al sistema. I sensori, al contrario svolgono il compito di registrare dati e inviarli al sistema. Infine le applicazioni mobile e gli attuatori hanno il doppio ruolo di inviare i dati registrati al sistema e di visualizzarne altri. In questo DE si analizzano nel dettaglio i casi d'uso relativi alle applicazioni mobile e ai sensori/attuatori per i quali è possibile individuare due distinti caso d'uso, ciascuno dei quali verrà descritto in dettaglio nei paragrafi successivi:

- Caso d'uso 1: Applicazione Mobile
- Caso d'uso 2: Sensori (Arduino, etc.)
- Caso d'uso 3: Applicazioni web

In ciascun caso d'uso verranno trattati i seguenti argomenti:

- Tipo di utenti coinvolti: utenti finali, operatori pubblici, developers, etc.
- Protocolli di comunicazione e API usati/e per inviare/ricevere dati
- Analisi dei dati legati alla tipologia di utente/sensore e motore per l'invio di raccomandazioni/suggerimenti agli utenti (standard e metodologie usate, scopo delle raccomandazioni in base agli obiettivi di progetto, aspetti legati alla privacy, etc.)
- Descrizione dello store

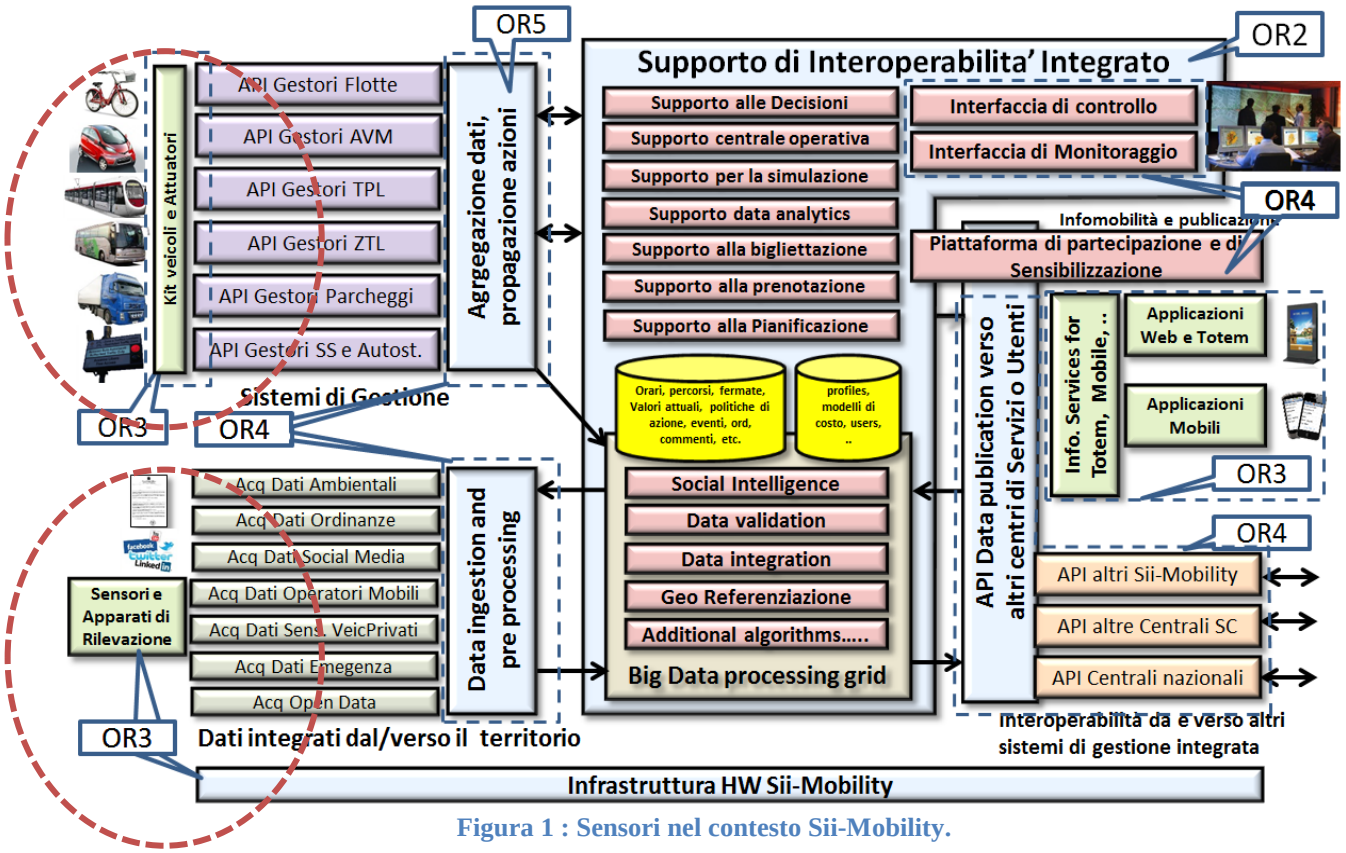

Figura 1 : Sensori nel contesto Sii-Mobility.

2.1 Descrizione del Caso d'uso 1: Applicazione Mobile

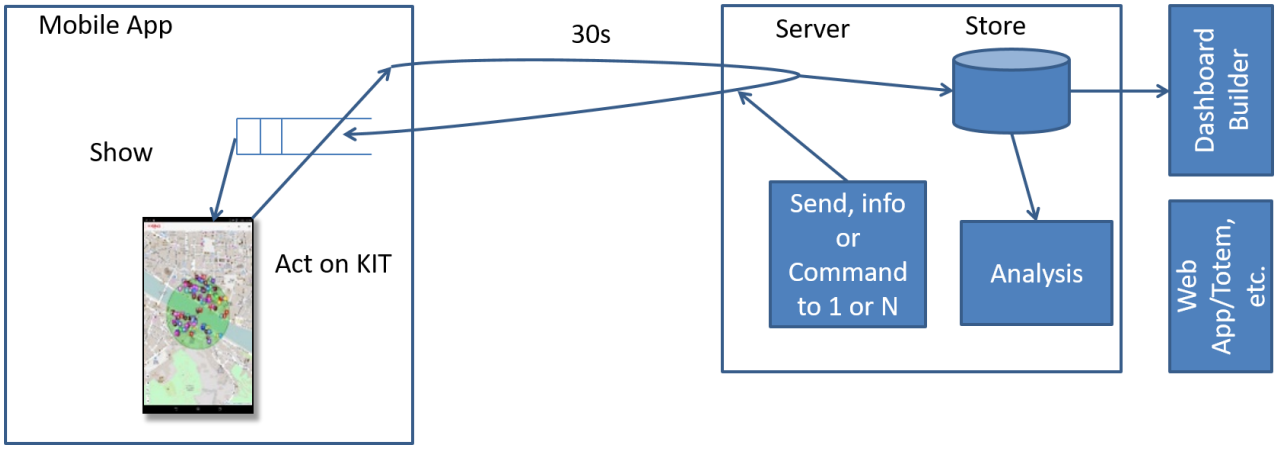

Figura 2: Mobile Phone and Sensor Server and Manager (SSM).

In questo caso d'uso è presente una interazione tra la applicazione mobile e il sistema. L'applicazione mobile svolge il doppio ruolo di:

- Registrare e inviare dati al sistema in base a precise specifiche e regole che rispettino la privacy degli utenti (movimento degli utenti, preferenze espresse, azioni effettuate sulla App, tipologia di utente, etc.)

- Visualizzare:
    - dati statici e real time (esattamente come avviene nel caso delle applicazioni web e dei totem) in base alle richieste dell'utente
    - dati legati strettamente all'utente, ovvero dati che tengono traccia di: posizione dell'utente e profilo dell'utente (informazioni geolocalizzate, raccomandazioni e suggerimenti, etc.)

2.1.1 Messaggi da Mobile a store

L'applicazione mobile è un'applicazione ibrida che può essere installata su più sistemi operativi. Attualmente può essere installata su: Android, iOS, Windows 10, Windows 10 Mobile. Inoltre, data la natura ibrida dell'applicazione, questa viene riadattata per renderla fruibile come Applicazione Web. La distribuzione dell'applicazione su più sistemi operativi ha portato a differenti strategie di recupero dati basate sulla possibilità o meno di eseguire servizi in background all'interno dei vari dispositivi. Attualmente il servizio di recupero dati funziona come un servizio in background sempre attivo per chi installa l'applicazione su un dispositivo con sistema operativo Android.
Negli altri sistemi operativi, non è stato sviluppato un servizio in background dedicato e quindi vengono raccolti dati durante l'utilizzo dell'applicazione in foreground.
Entrambi i servizi (foreground e background) seguono lo stesso flusso di operazioni, anche se sono formalmente diversi essendo scritti in due linguaggi diversi: Java per il servizio in background di Android e Javascript per gli altri sistemi operativi.

I dati che vengono raccolti sono quelli relativi alla posizioni e ai movimenti degli utenti tramite informazioni raw relative a latitudine, longitudine, velocità di spostamento, altitudine e accuratezza di tali misure fornita dal GPS del dispositivo.
Altri dati raccolti sono relativi alla disposizione, in particolare sul centro di Firenze, dei wifi pubblici con SSID FirenzeWifi e UnifiWifi, per avere una visione di insieme sulla corretta distribuzione di tali punti di accesso al wifi libero.
Infine, per capire su quali dispositivi è più diffusa l'applicazione, se viene usata più dai turisti che dai cittadini e per tenere traccia delle versioni dell'applicazione attulmente in uso, vengono raccolti dati relativi ai sistemi operativi e ai modelli dei dispositivi, che versione dell'applicazione è attualmente installata, che profilo e che lingua ha selezionato l'utente quando l'applicazione è in esecuzione.

Il protocollo di raccolta dei dati ruota principalmente intorno alla possibilità di recuperare in un determinato momento su un determinato dispositivo la posizione GPS o meno.

Il servizio cerca di recuperare le informazioni relative al Provider che fornisce la posizione preferendo il GPS a quella rilevata dal NETWORK. Se nessuno dei due è disponibile vuol dire che l'utente ha disabilitato il rilevamento della posizione e invieremo al server un json di tipo "status" con il quale verremo a conoscenza che il servizio sta ancora eseguendo ma non è possibile fare altro su quel disposivo. Vedremo in seguito che potremo inviare altri json di tipo "status" ma questo ha la caratteristica di avere latitudine e longitudine impostate a 0 dato che non possiamo recuperare nessuna informazione relativa alla posizione.

Se almeno uno dei due provider risulta attivo si potrebbero avere altri due problemi: tale provider non riesce in quel momento a rilevare la posizione del dispositivo (per esempio l'utente si trova all'interno di un palazzo dove il segnale GPS non riesce a recuperare informazioni dai satelliti) oppure la posizione rilevata non è aggiornata (l'utente è riuscito a recuperare la posizione ma poi è

entrato all'interno di un palazzo: la posizione è ancora l'ultima rilevata ma è passato troppo tempo). In entrambi i casi viene richiesto un aggiornamento della posizione da parte del servizio verso il provider attivo in quel momento: se l'aggiornamento non è possibile, viene nuovamente inviato un json di tipo "status" con latitudine e longitudine impostati a 0.

Supponendo di riuscire a recuperare una posizione aggiornata è possibile effettuare una scansione dei wifi e dei beacon che si trovano nelle vicinanze del dispositivo per inviarli in seguito al server.

Un primo controllo che viene effettuato prima di eseguire una scansione dei wifi è quello relativo alla modalità aereo che potrebbe essere inserita sul dispositivo: se la modalità aereo è attiva non viene effettuata nessun'altra operazione e viene inviato un json di tipo "status" con valorizzati solamente i campi di latitudine e longitudine che arrivati a questo punto dell'algoritmo saranno validi.
Se la modalità aereo non è attiva allora può essere effettuata la scansione: si controlla se sia possibile attivare il modulo wifi del dispositivo e se è possibile viene attivato e viene effettuata la scansione per raccogliere i dati nelle vicinanze del dispositivo. Nel caso in cui il modulo wifi fosse stato disattivato viene ripristinato lo stato iniziale del dispositivo per non interferire con le scelte dell'utente.

Ottenuta la lista dei dispositivi wifi nelle vicinanze del dispositivo viene effettuato un primo filtraggio che elimina i dati dei wifi aventi lo stesso MAC di altri già salvati se la rilevazione è temporalmente e spazialmente vicina ad un'altra.

Il secondo filtraggio viene effettuato per rimuovere il più possibile i wifi privati rilevati durante la scansione: vengono subito salvati i dati relativi a wifi pubblici e liberi con SSID FirenzeWifi e UnifiWifi. Se i dati dei wifi pubblici non riescono a coprire un ampio spettro spaziale e temporale allora vengono tenuti in memoria anche i wifi privati che riescono a sopperire ai "buchi" nel recupero di informazioni che sarebbero avvenuto con i wifi pubblici.

Il terzo filtraggio avviene confrontando i dati rimasti con quelli già inviati al server e anche in questo caso per dati aventi lo stesso MAC si eliminano quelli spazialmente e temporalmente vicini ad altri già inviati.

Una volta salvati i dati da inviare questi vengono inviati periodicamente dal servizio in background: se l'invio va a buon fine i dati inviati vengono cancellati da quelli salvati altrimenti vengono rimessi in coda per essere inviati nel successivo periodo.

Se l'invio va a buon fine in riposta viene restituito un json contenente il messaggio di successo di inserimento dei dati e una parte del json può contenere un messaggio o più messaggi generati dall'engager per l'utente. Questi messaggi vengono salvati in dei file temporanei che si trovano nelle cartelle dove può accedere l'applicazione e una volta che l'applicazione viene avviata vengono letti e vengono eseguite le operazioni necessarie a mostrare all'utente i nuovi messaggi arrivati.

Nella Figura 3 è possibile vedere le tempistiche con le quali vengono eseguite le operazioni contenute nel servizio che gira in background. Il blocco che è denominato "Save Current Location" ogni volta che è disponibile la posizione GPS salva dei dati una lista e quando vengono inviati i dati queste posizioni vengono utilizzate per capire come si sta muovendo l'utente in un determinato momento.

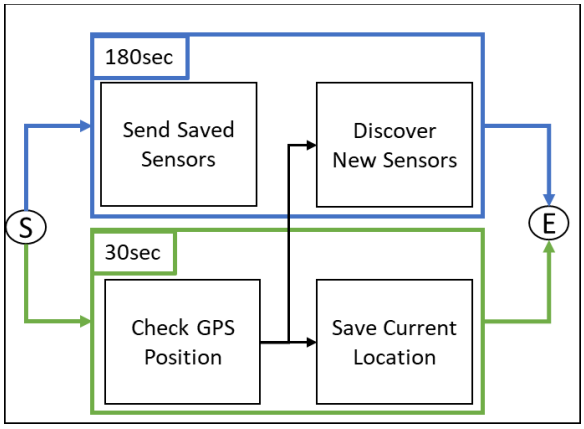

Figura 3 – Tempistiche del servizio di recupero dati

Sia il servizio in background che il servizio in foreground eseguono le stesse operazioni. Analizzando i dati raccolti fino ad ora è possibile notare come il servizio in foreground sia più preciso nell'invio dei dati, avvenendo quando l'applicazione è aperta è possibile effettuare tutte le chiamate di salvataggio e di invio dei dati previste. Nel caso del servizio in background, politiche diverse adottate dalle varie versioni di Android, sia per quanto riguarda le versioni ufficiali che quelle sviluppate dai costruttori dei dispositivi, stanno cercando di limitare l'utilizzo delle risorse (cpu, ram e network) da parte dei servizi che girano in background e aumentano il periodo di esecuzione di tale servizi che può passare dai 30 secondi teorici (di Figura 3) ai 5 minuti che vengono impostati di base da dispositivi creati da Samsung.

2.1.2 Descrizione lato server

Per quanto riguarda la gestione dei dati, sono state messe a disposizione delle API per l'inserimento dei dati nello store e la visualizzazione. Tali operazioni possono essere effettuate in base ai diritti di accesso del richiedente.

Le API disponibili sono le seguenti:

1. Insert Sensors
2. Get Sensors

API 1	Insert Sensors
Request	**HTTP POST: http://www.disit.org/sensor/api.php**
Accept	application-json
Parameters	**NO**
Json	Esempio: [{ "uid": "96d8ecaedc0f2e33262b7c2abd8492d0bbd438a25bcebc392def069553a66a5b", "uid2": "1c8867fe0fc4f9ec8309db95ff2fd079", "appID": "fdck-a", "version": "4.0.0", "lang": "it", "profile": "all", "device_model": "kminilte", "date_pre_scan": "2017-07-04 11:15:03", "lat_pre_scan": "43.79864", "long_pre_scan": "11.25356", "date": "2017-07-04 11:16:03", "latitude": "43.79864",

<table>
<tr><td></td><td>
<pre>
    "longitude": "11.25356",
    "accuracy": 20,
    "altitude": 0,
    "heading": 0,
    "provider": "fused",
    "type": "wifi",
    "rssi": "-94 dB",
    "network_name": "UnifiWiFi",
    "MAC_address": "00:c8:8b:5b:dc:dc",
    "frequency": "2462 Mhz",
    "capabilities": "[ESS][BLE]",
    "status": "",
    "prev_status": "",
    "avg_speed": "0.0001",
    "lin_acc_x": "0.05507",
    "lin_acc_y": "0.04474",
    "lin_acc_z": "0.08365",
    "avg_lin_acc_magn": "0.16139",
    "speed": "0.0001"
  },
  {
    "uid": "96d8ecaedc0f2e33262b7c2abd8492d0bbd438a25bcebc392def069553a66a5b",
    "uid2": "1c8867fe0fc4f9ec8309db95ff2fd079",
    "appID": "fdck-a",
    "version": "4.0.0",
    "lang": "it",
    "profile": "all",
    "device_model": "kminilte",
    "date_pre_scan": "2017-07-04 11:15:03",
    "lat_pre_scan": "43.79864",
    "long_pre_scan": "11.25356",
    "date": "2017-07-04 11:16:03",
    "latitude": "43.79864",
    "longitude": "11.25356",
    "accuracy": 20,
    "altitude": 0,
    "heading": 0,
    "provider": "fused",
    "type": "wifi",
    "rssi": "-96 dB",
    "network_name": "FirenzeWiFi",
    "MAC_address": "00:c8:8b:5b:dc:db",
    "frequency": "2462 Mhz",
    "capabilities": "[ESS][BLE]",
    "status": "",
    "prev_status": "",
    "avg_speed": "0.0001",
    "lin_acc_x": "0.05507",
    "lin_acc_y": "0.04474",
    "lin_acc_z": "0.08365",
    "avg_lin_acc_magn": "0.16139",
    "speed": "0.0001"
  }]
</pre>
</td></tr>
<tr><td>Response</td><td>
caso1: risposta in caso di inserimento dati (in formato JSON)
esempio: se vengono inserite N misurazioni nello store, si ha una risposta del tipo:
<pre>
{
"insert_result":"N Sensors stored!",
  "engager_result": {
        "assessor":true,
        "engagement":[],
        "assistance":[]
  }
}
</pre>
Caso2: segnalazione di eventuali errori di inserimento dei dati
JSON malformato: ‘Error in the Json schema format.’

Caso3: segnalazione di eventuali errori di inserimento dei dati
ERRORE di connessione con il DB: ‘HTTP 500 Internal server error’
</td></tr>
</table>

API 2	Get Sensors
Request	http://www.disit.org/sensor/api_select.php
Parameters	action* = 'get_sensors' user* = SI VEDA file di configurazione pwd* = SI VEDA file di configurazione type = {html, csv, json} limit = INT Offset = INT
Response	• CASO 1 (type = html): rende una pagina web con inseriti gli elementi in una tabella • CASO 2 (type = csv): rende un file csv con gli elementi richiesti • CASO 3: (type = json): rende un file con gli elementi richiesti in formato json.

A livello di tipologia di Database, è possibile memorizzare i dati sia su un database MySQL che su un DB NoSql.

La struttura delle tabelle è la seguente sia su MySQL che su Phoenix/HBase

TABELLA 'sensors'

Nome del campo	Per Tipo di misurazione	descrizione	obbligatorio	DB type	Unità di misura/formato/
idmeasure	B	Identificativo misurazione	No	int(11)	Nessuna condizione ulteriore
UUID	B	User id (proximityUUID)	No	varchar(35)	Nessuna condizione ulteriore
sender_IP	All	IP dell'apparecchio che invia dati	Si	varchar(25)	Formato IP
date	All	data di rilievo	Si	datetime	dateFormat(new Date(), "yyyy-mm-dd HH:MM:ss"
date_pre_scan	All	data di pre-rilievo (per controllare che non ci siano ritardi dal tempo di rilievo al tempo di salvataggio)	Si	datetime	dateFormat(new Date(), "yyyy-mm-dd HH:MM:ss"
type	All	Identifica il tipo di misurazione : wifi/beacon se il record si riferisce ad un wifi/beacon trovato nei dintorni del dispositivo, status se non	No	varchar(45)	{beacon, wifi, status}

		vengono trovati sensori nei dintorni del dispositivo o se non è possibile determinare la posizione GPS			
latitude	All	latitudine apparecchio	Si per B e W	double	WGS84 (decimal degrees)
longitude	All	longitudine apparecchio	Si per B e W	double	WGS84 (decimal degrees)
lat_pre_scan	All	latitudine apparecchio (per controllare che non ci siano spostamenti dal rilievo al salvataggio dei dati)	No	double	WGS84 (decimal degrees)
long_pre_scan	All	longitudine apparecchio apparecchio (per controllare che non ci siano spostamenti dal rilievo al salvataggio dei dati)	No	double	WGS84 (decimal degrees)
network_name	W	Nome della rete wifi rilevata	No	varchar(45)	Nessuna condizione ulteriore
sensor_name	B	Nome del beacon	No	varchar(45)	Nessuna condizione ulteriore
MAC_address	All	Indirizzo MAC del dipositivo wifi o beacon	Si per B e W	varchar(45)	Formato MAC
power_n	B, W	Potenza del segnale wireless rilevato	No	int(11)	dB
rssi_n	B, W	Potenza del segnale wireless rilevato	No	int(11)	dB
frequency_n	B, W	Frequenza alla quale sta lavorando il WiFi rilevato	No	int(11)	MHz
minor	B	Identificativo che serve per rilevare la posizione del beacon	No	int(11)	
major	B	Identificativo che	No	int(11	

		serve per rilevare la posizione del beacon		)	
capabilities	W	Possibilità di connessione relative al wifi rilevato	No	varchar(125)	Stringa = "[stringa1], [stringa2], …" **(*)**
speed	All	Velocità alla quale si sta muovendo il dispositivo. (Se il provider è GPS allora viene fornita da questo, altrimenti viene calcolata in base alle rilevazioni GPS precedenti.)	No	double	m/s (metri al secondo)
altitude	All	Altitudine del dispositivo rispetto al livello del mare	No	double	metri
provider	All	Indica come sono stati ricavati i dati relativi alla posizione	No	varchar(45)	{gps, network, fused}
accuracy	All	Precisione relativa alle coordinate GPS rilevate	No	double	metri
heading	All	Posizionamento del dispositivo rispetto al True North	No	double	Gradi (0-360) relativi al "vero nord"
device_model	All	String che indica il modello del dispositivo con il suo acronimo tecnico	No	varchar(255)	Nessuna condizione ulteriore
device_id	All	Codice SHA256 calcolato sul device.uid restituito dal plugin device di Cordova	No	varchar(64)	Nessuna condizione ulteriore
status	All	Indica se l'utente si trova in movimento (e con quale mezzo di trasporto) o se è fermo	No	varchar(45)	I possibili valori sono: stay, walk, car/bus/moto, train,airplane
prev_status	All	Indica se l'utente si trovava in movimento (e con quale mezzo di trasporto) o se era	No	varchar(45)	Nessuna condizione ulteriore

		fermo			
appID	All	Identificativo della applicazione che sta inviando i dati al server.	No	varchar(45)	Le applicazioni attuali utilizzano la seguente sintassi:xdck-y dove x: {f,t} per indicare Firenze o Toscana. y:{a,i,w,b} per indicare Android, iOS, Windows, Browser
version	All	Versione della applicazione che sta inviando i dati al server	No	varchar(45)	Per le nostra applicazioni una stringa del tipo x.y.z dove x,y,z sono numeri
lang	All	Lingua impostata sulla applicazione dall'utente	No	varchar(10)	Nessuna condizione ulteriore
uid2	All	Codice MD5 calcolato sul MAC Address del dipositivo (lower case e senza due punti) che sta inviando i dati. Stringa alfanumerica di 16 caratteri (64bit).	No	varchar(64)	Nessuna condizione ulteriore
profile	All	Categoria di profilo utente	No	varchar(45)	{studente, cittadino, pendolare, neutro e turista }
payload	VA MESSO	nel prox use case	NO	longtext	JSON ben formato

Le misurazioni sono relative alla presenza di: wifi (W), beacon (B), status (S)
Se nei dintorni del dispositivo dell'utente non vengono trovati wifi o beacon o non sono disponibili le coordinate GPS allora viene inviata una misurazione con tipo status (S).
Con questo tipo di misurazione è possibile avere dati relativi alla posizione dell'utente anche nel caso in cui non si abbiano W o B nei dintorni e, nel caso in cui non sia disponibile la posizione, si può capire se il servizio di invio dei dati viene ancora eseguito sul dispositivo anche se non riesce a trovare la posizione.

(*) esempio: [WPA-PSK-TKIP][WPA2-PSK-CCMP+TKIP][ESS]

TABELLA 'user_eval'. Rispetto alla tabella 'sensors' contiene:

- Un ristretto numero di sensori:
  - se si effettua una chiamata multi-sensore, inserisce solo il primo sensore
- Una selezione di field:
  - Profile
- I seguenti campi aggiuntivi (double, NON obbligatori con default a NULL):
  - lin_acc_x | accelerazione asse x
  - lin_acc_y | accelerazione asse y
  - lin_acc_z | accelerazione asse z
  - avg_lin_acc_magn | media della magnitudo dell'accelerazione
  - avg_speed | media della velocita'

Alcuni campi nel dettaglio:

Nome parametro	Descrizione	Esempio
appID	Identifica l'app dalla quale stanno arrivando i dati. Si potrebbe inviare una stringa composta dalle lettere iniziale delle parole che compongono il nome della app, con l'aggiunta finale di una lettera a,i,w che sta ad indicare il sistema operativo (Android,iOS, Windows)	fdck-a per Firenze dove, cosa... km4city installata su Android
version	Indica la versione attualmente installata nel device che sta inviando i dati. E' una stringa composta da 3 numeri separati da punti.	2.4.1
lang	Indica la lingua che ha attualmente selezionato l'utente. Stringa di due lettere che indica la lingua.	it per italiano en per inglese
uid2	Codice MD5 calcolato sul MAC Address del dipositivo (lower case e senza due punti) che sta inviando i dati. Stringa alfanumerica di 16 caratteri (64bit).	1f3870be274f6c49

2.1.3 Analisi dei dati e invio di raccomandazioni e suggerimenti

Sii-mobility utilizza il modulo User Engagement Engine per l'invio di raccomandazioni e suggerimenti all'utente. Il principale scopo dello User Engagement Engine (UEEngine) è di promuovere un cambiamento di comportamento della base utenza.
Tramite lo UEEngine, un operatore della città definisce strategie (campagne di stimolo) con lo scopo di:

- suggerire un certo comportamento virtuoso alla base utenza;
- ricompensare l'utenza che segue tale comportamento.

Il comportamento virtuoso viene suggerito tramite la generazione di un insieme di messaggi (engagement) a fronte del verificarsi di un insieme di condizioni sul contesto nel quale si trova l'utente. La combinazione "if <insieme-di-condizioni> then <azione>" è specificata nelle regole di engagement.

La ricompensa viene distribuita agli utenti tramite bonus (o simili) a fronte della collezione di un numero minimo di punti virtuali. I punti virtuali sono incassati nel caso in cui l'utente segue il comportamento virtuoso suggerito nei messaggi di engagement (in questo caso l'engagement viene etichettato come "eseguito"). Il numero di punti virtuali necessari a ritirare un bonus e il numero di punti incassati in caso di comportamento virtuoso è specificato nelle regole di ricompensa.
UEEngine si occupa di:

- monitorare la base utenza attiva e del contesto nel quale si trova;
- selezionare le regole di ingaggio con condizioni di contesto verificate;
    - e inviare agli utenti gli engagement specificati in tali regole;
- tener traccia degli engagement che sono stati visualizzati e che sono stati etichettati come "eseguiti";
- accreditare l'utenza che esegue gli engagement con un numero di punti come specificato nelle regole di ricompensa;
- distribuire i bonus richiesti dall'utente a fronte del raggiungimento di un numero di punti minimo come specificato nelle regole di ricompensa.

Il sistema mette a disposizione dell'operatore:

- un editor di regole (RULE ENGAGEMENT EDITOR) che permette di:
    - comporre le condizioni sul contesto dell'utente
    - di specificare il relativo engagement, da scegliere fra un set predefinito;
- Un'interfaccia JAVA per:
    - definire la modalità di distribuzione delle regole (engagement bannato)
    - definire quando un engagement può essere considerato eseguito
- Una portale web (WALLET GUI) per definire le regole di ricompensa per le diverse campagne:
    - definire i bonus
    - definire quali regole di engagement partecipano alla campagna e in quale misura

In dettaglio, il sistema:

- legge da una sorgente di informazione (in stream continuo) per recuperare gli utenti attivi a sistema (potenziali target di engagement, identificati mediante identificativo univoco) e le relative informazioni di contesto lato terminale (p.es. posizione gps);
- per ogni contesto dell'utente:
    - arricchisce con altre informazioni (contattando sorgenti secondarie) p.es. profilo utente, informazioni di mobilità dell'utente, punti di interesse nella sua prossimità, …;
    - verifica quali regole di ingaggio sono verificate e memorizza le azioni (engagement) specificate, pronte per essere distribuite all'utente
- a richiesta ritorna al Sensor Manager gli engagement disponibili
- registra gli engagement segnalati come cancellati dagli utenti
- tiene traccia degli engagement effettivamente visualizzati sul terminale dell'utente
- tiene traccia degli engagement che sono stati eseguiti

2.1.3.1 Architettura

L'architettura dello UEEngine è presentata nella **Errore. L'origine riferimento non è stata rovata.**. Da sinistra: l'operatore definisce le strategie utilizzando l'Engagement Rule Editor e tramite la Wallet GUI definisce eventuali bonus o ticket da distribuire in caso di comportamento virtuoso (per esempio: una risposta ad un sondaggio, l'utilizzo di un mezzo pubblico suggerito, etc.). Le strategie sono automaticamente tradotte in Condizioni osservabili e Azioni che possono essere inviate all'utente. Le condizioni si basano sull'evoluzione del profilo utente, sull'analisi del suo comportamento e sul contesto nel quale si trova (informazioni direttamente recuperabili dalle KM4City Apis [Bellini et al., 2014b]). Le regole di engagement sono definite usando Drools Workbench, l'interfaccia grafica di Drools [Drools].

Le informazioni di contesto degli utenti sono continuamente analizzate e recuperate dal Sensor Server e le azioni di engagement sono distribuite periodicamente al Sensor Manager a fronte di richiesta ad hoc. In alcuni casi UEEngine può elaborare più azioni contemporaneamente e può richiedere la generazione di engagement di breve o lunga durata. La lista degli engagement inviata agli utenti è memorizzata localmente per analisi successive e per evitare repliche. Se l'utente visualizza sul terminale un engagement questo viene etichettato come "visto".
Il sistema mantiene una versione aggiornata del contesto dell'utente e inoltre controlla quali condizioni (specificate nel set di regole di engagement) sono valide. Ogni qualvolta tutte
le condizioni specificate in una regola di engagement sono verificate, la piattaforma prepara e consegna l'azione specificata alla SENSOR APP (invia un engagement all'utente). Allo stesso tempo, la piattaforma monitora se e quando l'engagement inviato è realmente eseguito (p.es. se una richiesta di sondaggio viene inviata all'utente, la piattaforma controlla se il sondaggio è stato visualizzato, compilato e sottomesso) ed eventualmente aggiorna il contesto dell'utente secondo le informazioni specificato nell'insieme di regole di feedback. Per una descrizione più approfondita si rimanda a [**Errore. L'origine riferimento non è stata trovata.**].

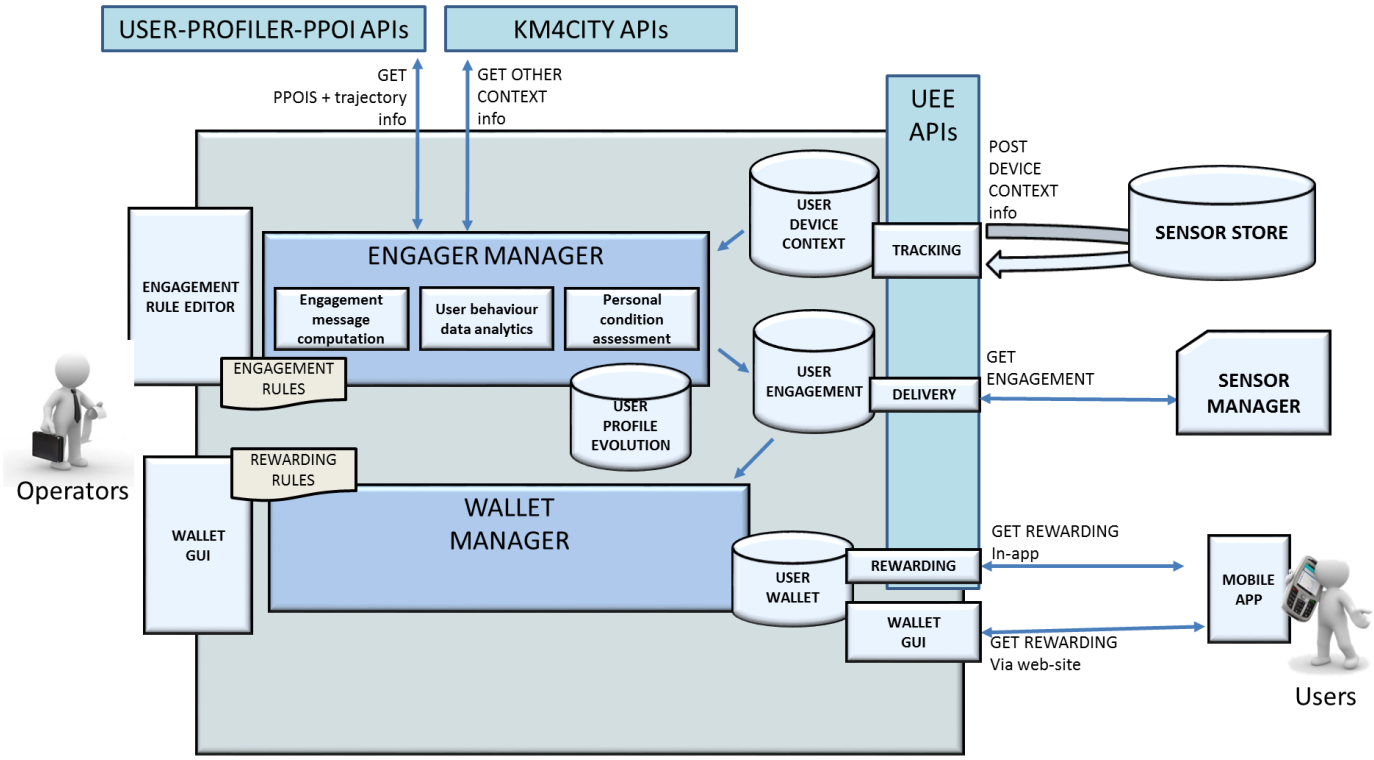

Figura 4: architettura dello User Engagement Engine (UEEngine).

2.1.3.2 Contesto e condizioni

Le variabili di contesto a disposizione dell'operatore per scrivere le condizioni delle regole di ingaggio sono classificate in quattro macro-classi, si veda *Figura 5*.

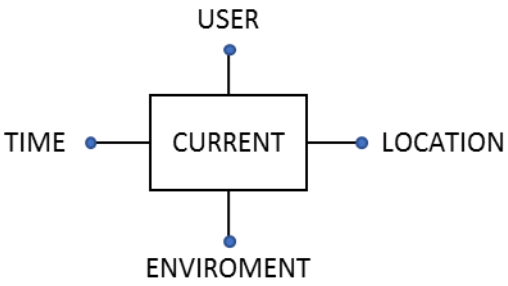

Figura 5: Macro-classi delle variabili di contesto.

USER:
- languages (list of string: en, it, …)
- profile (string: all, student, citizen, etc.)
- ppois (list of PPOI)
- mobilityMode (string: still, walk, …)
- howLongMobilityMode(double)
- switchMobilityMode (string: null, parking, in_mobility)
- last PPOI (PPOI)
- current PPOI (PPOI)
- next PPOI (PPOI)
- currentPPOI (PPOI)
- nextPPOI (PPOI)

PPOI:
- name (string: HOME, WORK, SCHOOL, PPOIx, SPENTTIME, …)
- accuracy (float)
- location (LOCATION)
- confirmation (boolean: null, true, false)

switchMobilityMode.PARKING:
- livepoi.current!=null (not in mobility)
- livePPOI.speed > 7.0 (in car)

switchMobilityMode.IN_MOBILITY:
- in the last 11 minutes, there are at least 3 gps date
- every gps data is 100 mt
- average speed is > 5m/s

nextPPOI:
- at least 2 prediction in same day-slot/weekday
- at least 75% of accuracy

LOCATION
- gpsLatitude (double)
- gpsLongitude (double)
- address (string)
- number (string)
- municipality (string)
- cpz (string)

- gpsLatitudeCluster (double) – cluster of 100 mt
- gpsLongitudeCluster (double) – cluster of 100 mt

TIME
- milliseconds epoch time (long)
- daySlot (string: afternoon, lunch, …)
- year/month/dayOfMonth/dayOfWeek/weekOfMonth (integer)
- Hours/minutes (integer)
- timeClustered (string) – cluster of 5 minutes

ENVIROMENT
- events (list of EVENT)
- closePois (list of POI) – cluster of 100 mt
- closestPoi(POI) – cluster of 40 mt

POI
- location (LOCATION)
- name (string)
- serviceType (string)
- serviceUri (string)
- tipo (string)
- typeLabel (string)

EVENT
- poi (POI)
- endData (string)
- place (string)
- price (float)

2.1.3.3 Engagement

I messaggi di engagement che possono essere presentati agli utenti sono di tre tipi SURVEY, REQUEST_PHOTO, SHOW e suddivisi in due classi, ENGAGEMENT e ASSISTANCE.
- classe: ENGAGEMENT
  - type: SURVEY
  - type: REQUEST_PHOTO
- classe: ASSISTANCE
  - type: SHOW

Ogni engagement ha le seguenti caratteristiche:
- time_elapse: (minuti), default di 60 minuti, ma può essere cambiato da ogni regola
  - dopo quanto tempo il messaggio scade (non viene più visualizzato)
- action_bannedfor: (minutes) default di 0 minuti
  - per quanto tempo il singolo messaggio, dopo essere "visto", non viene più inserito in coda per essere inviato all'utente
- action_sendrate: (minutes) default di 0 minuti
  - per quanto tempo un messaggio della stessa classe non viene più inviato al device, dopo che lo stesso è stato "inviato" o "visto"

        - se minore di threshold_live_minutes (10 minuti), lo scenario viene considerato LIVE (vengono sempre inseriti in coda tutti gli engagement collezionati della stesa regola)
- action_howmany: default=1
    - quanti engagement contemporaneamente vengono inviati al device
- action_rulename: nome della regola

2.1.3.4 Protocollo di invio dati

Dall'esterno il Sensor Server and Manager (SSM) richiede gli engagement associati ad un certo utente. Lo UEEngine ritorna le ultime entry, in ordine di tempo, di ogni categoria (take photo, message, show, …) e che:

- non sono ancora scadute
- che non sono ancora state inviate
- se last sent e' minore del rate (parametro su regola)
- entry create dopo l'ultimo la data di creazione dell'ultimo SENT (non vengono mandate entry vecchie)
- ne vengono tornate HOW_MANY per ogni regola

Appena inviate, gli engagement vengono settate in status "inviate".

Request	HTTP GET: http://192.168.0.17:8080/engager-api/engager?user_id=af97e11488be5af2408ab27ddd90d52ad763b5d4ff1e5f89e6fc378fdf785f75 PARAMETRI: user_id: identificatore univoco rappresentante l'utente
Response	JSON: {"actions":[{"type":"TAKE_A_PHOTO","msg":"http://www.disit.org/km4city/resource/82ec01f786ba28189f4a2e0a40b8830fit","classe":"ENGAGEMENT","id":485,"time_elapse":1468234958000}]} PARAMETRI: Classe: ASSISTANCE / ENGAGEMENT Type: SHOW / REQUEST_PHOTO / SURVEY Title: titolo dell'engagement Msg: messaggio dell'engagement Uri: eventuale uri rappresentante l'engagement Gps: eventuale posizione relativa all'engagement ServiceName: eventuale nome del servizio relativo all'engagement ServiceType: eventuale tipologia del servizio relativo all'engagement Label: eventuale etichetta del servizio relativo all'engagement Descrizione: descrizione aggiuntiva dell'engagement

2.1.3.5 Esempi di regole

ALERT su PARCHEGGIO: si comunica all'utente un messaggio di allerta nel caso in cui parcheggi in una controlled-parching-zone diversa a quella (sua) di residenza:

Descrizione alto livello:
IF user.switch==PARCHING
AND Place.cpz!= user.cpz_home
THEN send an ALERT

- Elapsed in DEFAULT (un'ora)
- Bannedfor=0 (default)
- Sendrate=0 (default)

- Howmany=1 (default)

Regola descritta usando Drools Workbench, *Figura 6*:

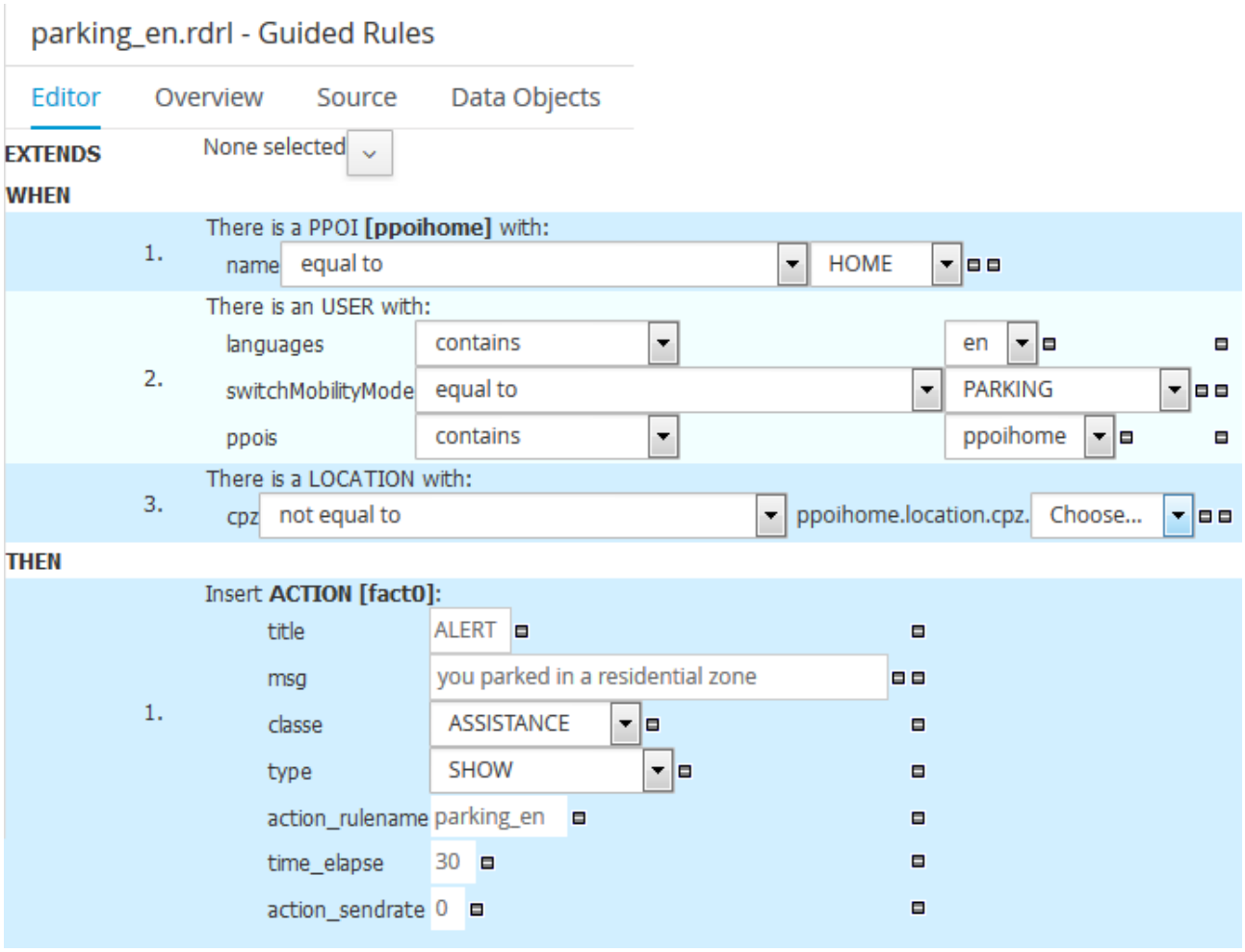

Figura 6: Regola descritta usando Drools Workbench.

Regola tradotta in MVEL:

```
rule "parking_en"
dialect "mvel"
when
ppoi:PPOI(name == "HOME")
USER(languages contains "en", switchMobilityMode ==
"PARKING", ppois contains ppoi)
LOCATION(cpz != ppoi.location.cpz)
then
ACTION fact0 = new ACTION();
fact0.setType("SHOW");
fact0.setTitle("ALERT");
fact0.setMsg("you parked in a residential zone");
fact0.setTime_elapse(30);
fact0.setAction_sendrate(0);
insert(fact0);
end
```

Esempi di engagement visualizzati sulla MOBILE APP, *Figura 7.*

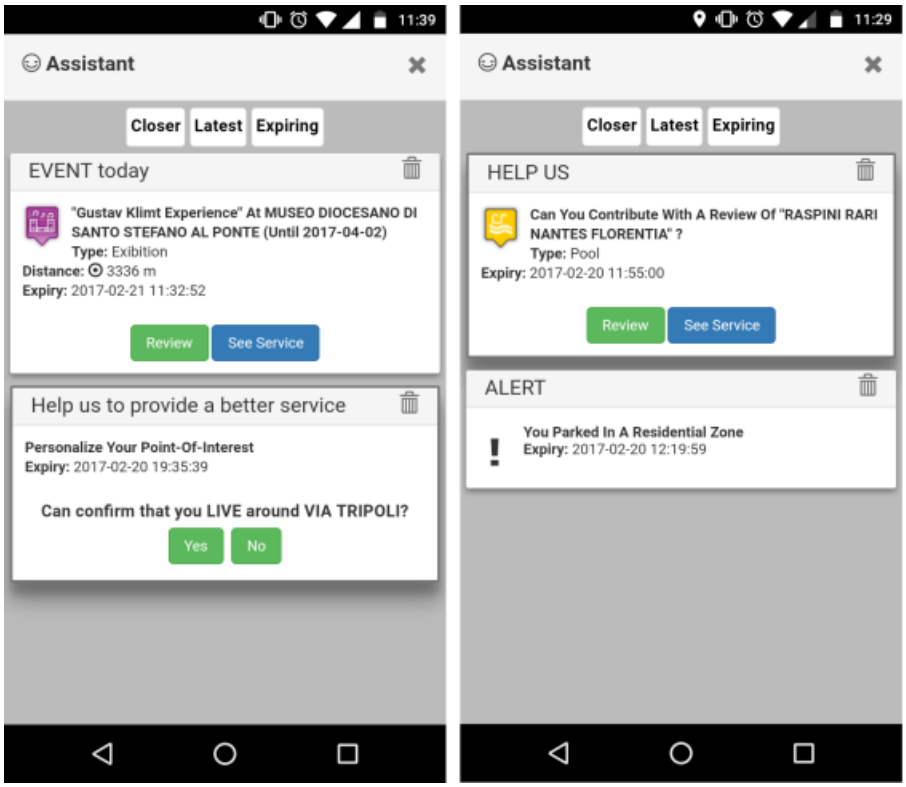

Figura 7: Esempi di engagement visualizzati sulla MOBILE APP

2.2 Descrizione del Caso d'uso 2: Sensori (Arduino)

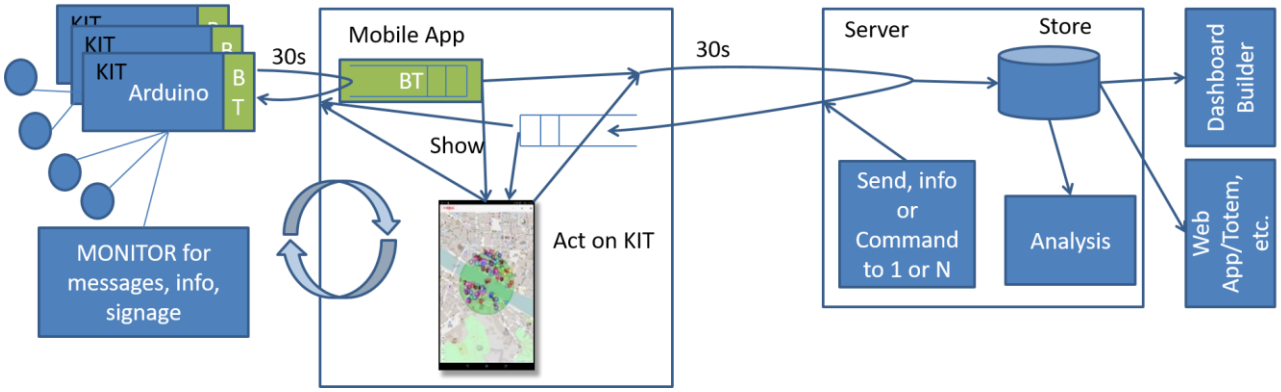

Figura 8: Caso d'uso2 – Sensori, Mobile Phone e Sensor Server and Manager (SSM).

Si veda questo caso d'uso (*Figura 8*) come una estensione del Caso d'uso 1. Si inseriscono qui le descrizioni che mancano nel precedente.
In questo caso d'uso si aggiungono i sensori. I dati provengono da fonti di tipo diverso quali: Bike, macchine private e pubbliche, Taxi (tvcam, video, etc.), Autobus (Bus tv cam, video, tem 1-5, ertc.)

2.2.1 Messaggi da Sensori a Mobile

I sensori sono leggibili da parte del dispositivo solamente in determinate situazioni: nel caso del bike sharing è possibile recuperare i dati quando l'utente sta effettivamente utilizzando una bicicletta e nel caso di guida in auto è possibile recuperare i dati da dispositivi collegati alla centralina solamente se questi sono effettivamente installati. Se ci troviamo in una delle situazioni dove è possibile recuperare i dati, questi possono essere recuperati attraverso dei dispositivi bluetooth ed è possibile aggiungerli ai json visti nella sezione dove si descrivevano i dati recuperati dai dispositivi wifi.
Il servizio di recupero dei dati da dispositivi bluetooth viene eseguito in parallelo a quello di recupero dei dati wifi. Il servizio di invio dei json memorizzati è lo stesso per entrambre le procedure di recupero delle informazioni.

2.2.2 Messaggi da Mobile a store

Il comportamento è lo stesso descritto nel paragrafo 2.1.1.

2.2.3 Descrizione lato server: Sensor Server and Manager

Il comportamento è lo stesso descritto nel paragrafo 2.1.2.

2.2.4 Analisi dei dati e invio di raccomandazioni e suggerimenti

L'analisi dei dati per il Caso d'uso "Sensori (Arduino)" segue lo stesso workflow descritto nel paragrafo 2.1.4 "Analisi dei dati e invio di raccomandazioni e suggerimenti", sebbene gli engagement proposti al terminale Arduino si focalizzano sui dati raccolti descritti nei paragrafi 2.2.2 e 2.2.3.
Esempi di messaggi presentati all'utente possono essere

- Vista retro bus
- Semaforo rosso connesso
- Segui una velocita' di 40 Km/h per onda verde
- Mantieni una velocità piu' lineare per consumare meno benzina
- Limite di 50Km/h su questa tratta
- Cambio percorso

2.3 Descrizione del Caso d'uso 3: Applicazione Web

Le applicazioni web e fisse sfruttano semplicemente le API di Sii-Mobility (sia quelle generali del sistema, sia quelle legate solo ai sensori/attuatori) per la visualizzazione dei dati di interesse senza inviare dati nello store. Il caso d'uso, come spiegato in precedenza, è una restrizione di quello descritto per le applicazioni mobile e per gli attuatori (API2 'Get Sensors', descritta nel paragrafo 2.1.1, caso d'uso 1).

3 Descrizione Store

Lo store è realizzato mediante database architetturalmente eterogenei: Oracle Mysql è il motore del database relazionale e Apache Hbase è il motore del database non relazionale.
Apache Hbase è un database distribuito basato su BigTable di Google caratterizzato da un'elevata efficienza nella gestione di grandi volumi di dati, tolleranza ai guasti e scalabilità che derivano dal layer sottostante Apache Hadoop.
Apache Hbase (si veda *Figura 10*) si basa sul paradigma master – slave dove:

- il master gestisce il namespace delle tabelle
- gli slave(denominati Region Server) gestiscono le regioni - le parti raggruppati per righe - delle tabelle

Apache Hadoop è un framework su cluster che espone i seguenti servizi:

- un filesystem distribuito ovvero HDFS
- un paradigma di computazione parallela ovvero MapReduce.

Il filesystem distribuito HDFS si basa sul paradigma master-slave in configurazione High Availability – HA (si veda *Figura 9*) mediante l'uso di:

- Due master (chiamati namenode) uno attivo (Active Namenode) e uno in standby (Standby Namenode)
- Demoni Journalnode in configurazione distribuita per replicare il namespace del filesystem
- Demoni Zookeeper in configurazione distribuita a supporto dell'automatic take over in caso di guasto di uno dei due master.

La ridondanza dei dati è a carico dei nodi slaves (Datanode).

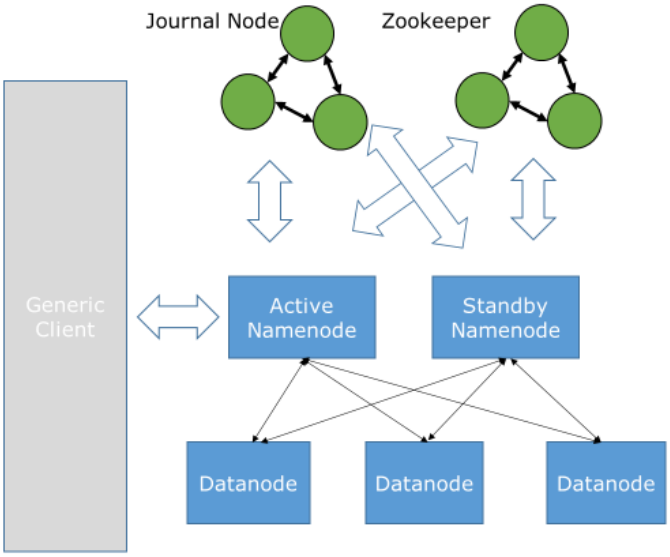

Figura 9: HDFS HA

Apache Phoenix è un coprocessore SQL per Apache Hbase che permette le operazioni comuni di creazione, modifica e ricerca sulle tabelle che verranno poi tradotte nelle operazioni primitive di hbase (creazione, modifica e "scansione").

Apache Phoenix è usato per mantenere una omogeneità nelle operazioni di interrogazione e di aggiornamento dello store .

Come per Oracle Mysql, esiste sia un'interfaccia JDBC che un'interfaccia ODBC. L'interfaccia ODBC è relizzata mediante un servizio di traduzione da richieste http verso JDBC ovvero PQS - Phoenix Query Server – (si veda figura sottostante).

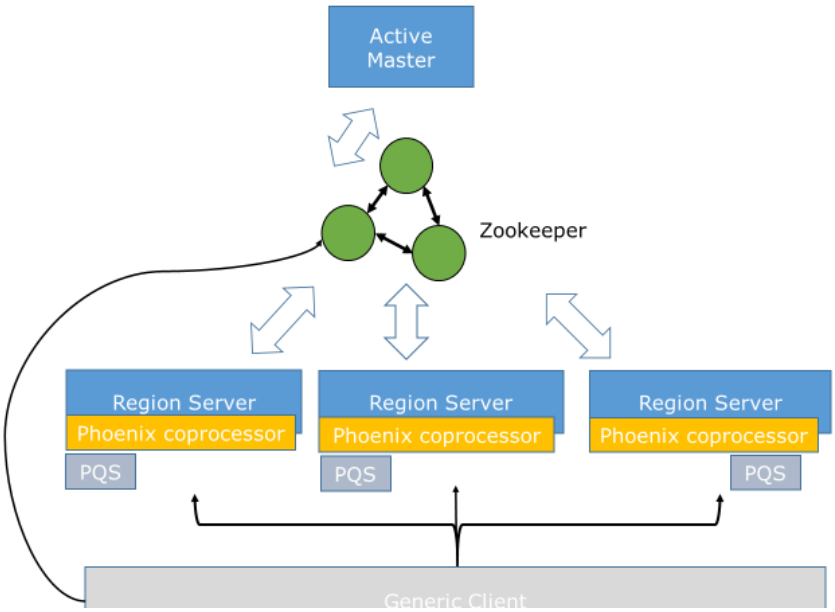

Figura 10 Apache Hbase Phoenix

In *Figura 11*, è riassunta la composizione dei servizi citati.

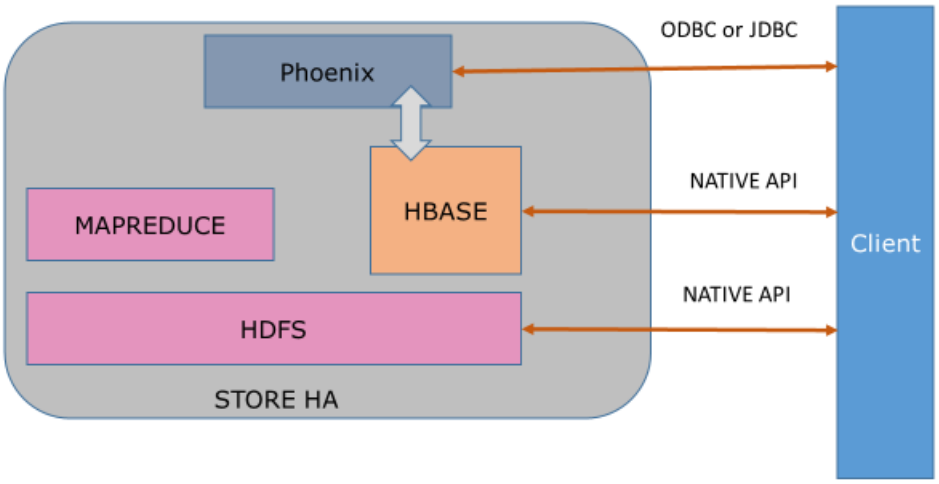

Figura 11 Architettura Database distribuita. (TODO: sistemare la figura)

4 Bibliografia

- [Bellini et al., 2013] Bellini P., Di Claudio M., Nesi P., Rauch N., "Tassonomy and Review of Big Data Solutions Navigation", Big Data Computing To Be Published 26th July 2013 by Chapman and Hall/CRC
- [Bellini et al., 2014b] P. Bellini, D. Cenni and P. Nesi, "Optimization of Information Retrieval for Cross Media contents in a Best Practice Network", International Journal Multimedia Information Retrieval, 2014, pp. 147-159.
- [Drools] Drools workbench documentation - https://docs.jboss.org/drools/

5 Acronimi

- MVEL: MVFLEX Expression Language
- PPOI: point of interest
- PPOI: personal point of interest

6 Appendice

6.1 Tabelle MySQL e Phoenix/HBase

Tabella 'sensors':

```
CREATE TABLE sensors (
  idmeasure bigint NOT NULL,
  UUID varchar(45) DEFAULT '',
  id varchar(35) DEFAULT '',
  sender_IP varchar(25),
  date timestamp DEFAULT NULL,
  type varchar(45),
  latitude double DEFAULT 0,
  longitude double DEFAULT 0,
  network_name varchar(45)  DEFAULT '',
  sensor_name varchar(45)  DEFAULT '',
  MAC_address varchar(45)  DEFAULT NULL,
  power varchar(45)  DEFAULT NULL,
  rssi varchar(45)  DEFAULT NULL,
  minor integer DEFAULT NULL,
  major integer DEFAULT NULL,
  frequency varchar(45)  DEFAULT NULL,
  capabilities varchar(125)  DEFAULT NULL,
  speed double DEFAULT NULL,
  altitude double DEFAULT NULL,
  provider varchar(45) DEFAULT '',
  accuracy double DEFAULT NULL,
  heading double DEFAULT NULL,
  lat_pre_scan double DEFAULT NULL,
  long_pre_scan double DEFAULT NULL,
  date_pre_scan timestamp DEFAULT NULL,
  device_id varchar(64)  DEFAULT NULL,
```

```
  frequency_n integer DEFAULT NULL,
  power_n integer DEFAULT NULL,
  rssi_n integer DEFAULT NULL,
  device_model varchar(255)  DEFAULT NULL,
  status varchar(45)  DEFAULT NULL,
  prev_status varchar(45)  DEFAULT NULL,
  appID varchar(45)  DEFAULT NULL,
  version varchar(45)  DEFAULT NULL,
  lang varchar(10)  DEFAULT NULL,
  uid2 varchar(64) DEFAULT NULL,
  profile varchar(45)  DEFAULT NULL,
  constraint pk PRIMARY KEY (idmeasure)
)
```

Tabella ‘user_eval:

```
CREATE TABLE `user_eval` (
  `user_eval_id` int(11) NOT NULL AUTO_INCREMENT,
  `date` datetime DEFAULT NULL,
  `device_id` varchar(64) DEFAULT NULL,
  `latitude` double DEFAULT NULL,
  `longitude` double DEFAULT NULL,
  `cc_x` int(11) DEFAULT NULL,
  `cc_y` int(11) DEFAULT NULL,
  `speed` double DEFAULT NULL,
  `altitude` double DEFAULT NULL,
  `provider` varchar(45) DEFAULT NULL,
  `accuracy` double DEFAULT NULL,
  `heading` varchar(45) DEFAULT NULL,
  `lin_acc_x` double DEFAULT NULL,
  `lin_acc_y` double DEFAULT NULL,
  `lin_acc_z` double DEFAULT NULL,
  `avg_lin_acc_magn` double DEFAULT NULL,
  `avg_speed` double DEFAULT NULL,
  `lat_pre_scan` double DEFAULT NULL,
  `long_pre_scan` double DEFAULT NULL,
  `date_pre_scan` datetime DEFAULT NULL,
  `prev_status` varchar(45) DEFAULT NULL,
  `curr_status` varchar(45) DEFAULT NULL,
  `curr_status_new` varchar(45) DEFAULT NULL,
  `curr_status_time_new` double DEFAULT NULL,
  `lat_centroid` double DEFAULT NULL,
  `lon_centroid` double DEFAULT NULL,
  `last_status_row` int(11) DEFAULT NULL,
  `appID` varchar(45) DEFAULT NULL,
  `version` varchar(45) DEFAULT NULL,
  `lang` varchar(10) DEFAULT NULL,
```

```
  `uid2` varchar(64) DEFAULT NULL,
  `profile` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`user_eval_id`),
  KEY `idx_user_eval_cc_x_cc_y` (`cc_x`,`cc_y`),
  KEY `idx_user_eval_device_id_date` (`device_id`,`date`),
  KEY `idx_user_eval_date` (`date`),
  KEY `idx_user_eval_curr_status_new` (`curr_status_new`),
  KEY `idx_user_eval_latitude_longitude` (`latitude`,`longitude`),
  KEY `Index_7` (`user_eval_id`,`date`),
  KEY `idx_user_eval_curr_status_time_new` (`curr_status_time_new`),
  KEY `idx_user_eval_last_status_row` (`last_status_row`)
) ENGINE=InnoDB AUTO_INCREMENT=29548731 DEFAULT CHARSET=utf8;
```