Applicazione Android per dati automotive su veicoli elettrici

Laureando: Andrea Bettin 411707
Relatore: Ch.mo Prof. Michele Moro

Corso di laurea in Ingegneria Informatica

22 Novembre 2013

Anno Accademico 2012/2013

Ringraziamenti

Volevo ringraziare il Prof. Michele Moro per l’aiuto durante la stesura di questo mio lavoro.
Inoltre ringrazio l'Alkè per avermi dato la possibilità di svolgere l'attività necessaria per lo studio effettuato.
Un ringraziamento ai miei genitori e a mia sorella per il costante supporto.

Speciale è il ringraziamento alla mia fidanzata Ilenia per avermi sempre sostenuto e incoraggiato.

Infine un ringraziamento a tutti gli amici in particolare modo all'ospitalità e l'aiuto di Rossana e Domenico.

Indice

1 Introduzione

Il mercato dei veicoli elettrici sta crescendo in tutto il mondo e gran parte delle case automobilistiche hanno pianificato il lancio di nuovi modelli nel corso dei prossimi anni. Nel 2012 le automobili elettriche vendute nel mondo hanno raggiunto la significativa cifra di 120 mila unità; le stime per il 2013 parlano di un incremento nelle vendite di circa il 50%, per arrivare al probabile traguardo delle 170/190 mila vetture su scala globale: un aumento che alcuni analisti di mercato equiparano a un vero e proprio boom, tale da ipotizzare entro il 2018 la cifra record di 2,7 milioni di veicoli elettrici.

I veicoli elettrici sono oggetto di un'attenzione crescente per svariati motivi: l'esigenza e la necessità di ridurre il livello di inquinamento atmosferico nelle aree urbane (per diminuire i conseguenti rischi sanitari per la popolazione), l'incremento del costo del carburante tradizionale e i previsti incentivi governativi per stimolare gli acquisti. Ma questa attenzione è anche giustificata dal ruolo che la mobilità elettrica può svolgere nel rendere più efficiente l'intero comparto elettrico: tutte le esperienze avviate in materia di Smart Grids e Smart Cities (reti e città intelligenti) vedono proprio nell'integrazione tra la mobilità elettrica e reti elettriche un tassello fondamentale nell'intera progettazione innovativa.

All'interno dei veicoli elettrici troviamo un importare settore di mezzi, denominati commerciali, usati principalmente per attività professionali in tutti gli ambiti produttivi o di servizi. Questa categoria di veicoli mette a disposizione dell'utilizzatore, oltre ai vantaggi citati prima per i veicoli elettrici, anche caratteristiche necessarie per affrontare situazioni di lavoro quotidiane, per esempio capacità di carico fino a 1.000 Kg o di traino fino a 4.000 Kg. Lo studio effettuato in questo lavoro prende in esame un veicolo elettrico commerciale, Alkè modello XT420E/EL, anche se comunque lo studio e la tecnologia applicata possono adattarsi a qualsiasi veicolo elettrico.

Al fine di rendere un veicolo elettrico capace di comunicare con altri dispositivi intelligenti o segnalare all'utente informazioni utili per l'ottimizzazione del mezzo, in questo lavoro si utilizzerà uno smartphone di ultima generazione.

Connettendo i dispositivi elettronici del veicolo con uno smartphone, strumento alla portata di tutti e dalle molteplici capacità, è possibile individuare in poco tempo e anche da remoto dati come: l'autonomia del veicolo, lo stato della ricarica del veicolo, statistiche di chilometraggio, verifica dei percorsi dei mezzi, la segnalazione di avarie di qualche tipo oppure della distanza dalla stazione successiva di ricarica.

1.1 Obiettivo della tesi

L'obiettivo di questa tesi è di realizzare un'applicazione per telefoni con sistema operativo Android che evidenzi all'utente, per un veicolo o più veicoli di proprietà i seguenti dati:

- controllo della carica/scarica
- autonomia
- controllo della posizione geografica
- indicazioni sullo stile di guida
- diagnostica remota degli errori del body computer e gestione flotte aziendali

L'applicazione viene progettata per lavorare in due modalità: connessione diretta al veicolo con un cavo usb o connessione remota tramite rete 3G.

Nel primo caso l'operatore utilizza il veicolo per gli spostamenti e il dispositivo Android rileva e mostra all'operatore i dati richiesti. Nel secondo caso, l'operatore è lontano dal veicolo e l'app consente di visualizzare alcuni dati essenziali come la percentuale di carica. I dati sono forniti tramite un web service.

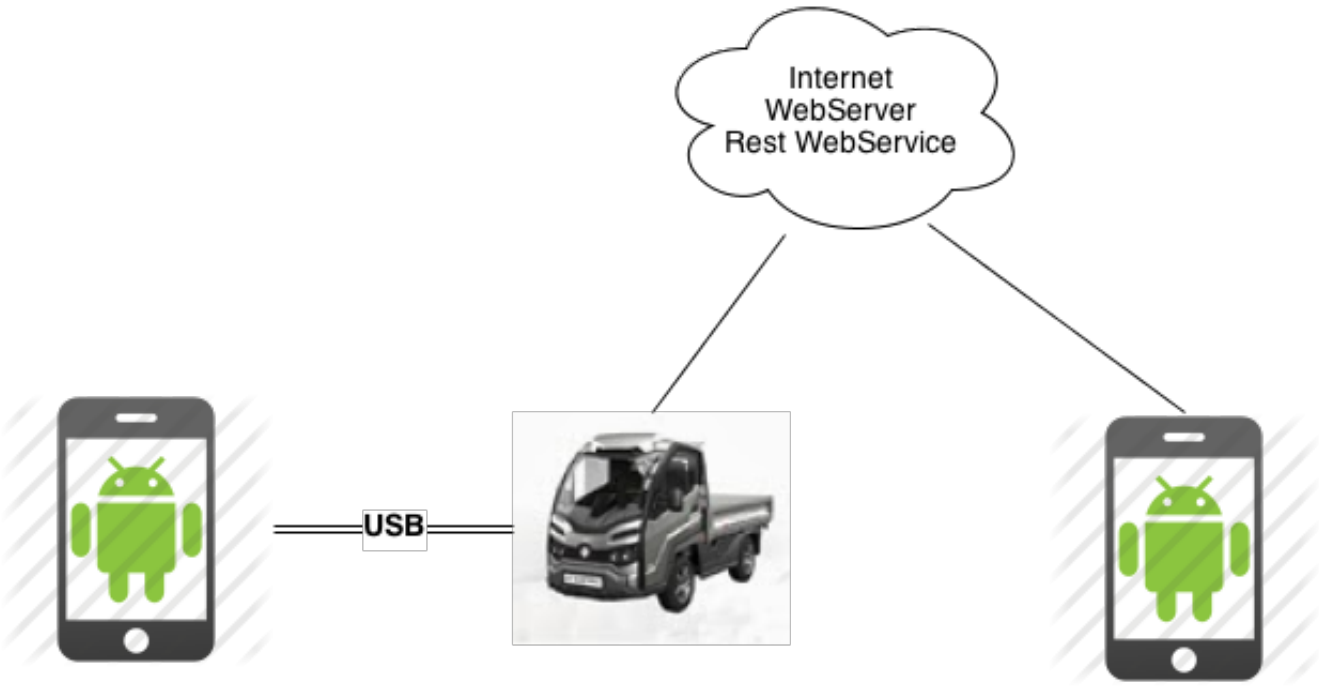


Figura 1: Schema dell'applicazione

1.2 Prerequisiti per l'utilizzo del software

Per poter testare ed utilizzare in modo completo tutto il software sviluppato è necessario disporre di:

- sistema Linux distribuzione Debian, installato nella linux-box montata sul veicolo;
- dispositivo o emulatore can-bus collegato al sistema Linux;
- ambiente di sviluppo Android con SDK android;
- smartphone con sistema operativo Android v4.0 o superiore;
- un server web Apache e un rest web service, su un server Linux di test;

2 Protocollo CAN-BUS

Lo studio affronta come primo argomento il protocollo CAN-BUS, utilizzato per la comunicazione tra dispositivi elettronici di un veicolo. Il frame del protocollo can-bus descritto in seguito sarà utilizzato come classe fondamentale nella comunicazione usb tra smartphone e linux-box.

Il Controller Area Network**,** noto anche come CAN-BUS (controller area network), è stato introdotto dalla Bosch nei primi anni '80 per applicazioni automobilistiche, per consentire la comunicazione fra dispositivi elettronici intelligenti montati su un autoveicolo, ma si è diffuso ormai in molti settori dell'industria tanto che sono sorti anche consorzi di aziende che promuovono questa diffusione.

La BMW 850 coupé, commercializzata nel 1986, è stata il primo veicolo dotato di CAN Bus. Per la prima volta, ognuno dei sistemi e sensori nei veicoli era in grado di comunicare a velocità molto elevate (25kbps - 1Mbps) su una linea di comunicazione singolo o doppio filo in contrasto con i precedenti telai multifilo.

Il protocollo CAN è un bus seriale di comunicazione digitale di tipo "broadcast" (tutti i nodi ricevono gli stessi pacchetti). Esso permette il controllo real-time con un livello di sicurezza molto elevato. Il suo successo è dovuto ai notevoli vantaggi tecnologici che offre:

elevata affidabilità: la rilevazione degli errori e la richiesta di ritrasmissione vengono gestite direttamente dall'hardware.

cablaggio semplice: CAN è un bus seriale implementato su un doppino intrecciato (schermato o meno a seconda delle esigenze). I nodi non hanno un indirizzo che li identifichi e possono quindi essere aggiunti o rimossi senza dover riorganizzare il sistema o una sua parte.

tempi di risposta rigidi: la tecnologia CAN prevede molti strumenti hardware e software e sistemi di sviluppo per protocolli ad alto livello (il bus CAN implementa solo i primi due livelli della pila ISO-OSI) che consentono di connettere un elevato numero di dispositivi mantenendo stringenti vincoli temporali.

alta immunità ai disturbi: lo standard ISO11898 raccomanda che i circuiti integrati di interfaccia possano continuare a comunicare anche in condizioni estreme, come l'interruzione di uno dei due fili o il cortocircuito di uno di essi con massa o con l'alimentazione.

gestione per ogni nodo degli errori: ciascun nodo è in grado di rilevare il proprio malfunzionamento e di autoescludersi dal bus se esso è permanente. Questo è uno dei meccanismi che consente alla tecnologia CAN di mantenere la rigidità delle temporizzazioni, impedendo che un solo nodo metta in crisi l'intero sistema.

diffusione dello standard: la larga diffusione del protocollo CAN in questi venti anni ha determinato un'ampia disponibilità di circuiti integrati ricetrasmettitori, di microcontrollori che integrano porte CAN, di tools di sviluppo, oltre ad una sensibile diminuzione del costo di questi sistemi. Questo è molto importante per far sì che uno standard si affermi nell'ambito industriale.

Questo BUS ha capacità "Multi-Master", ovvero tutti i nodi della rete posso trasmettere (ruolo master), più nodi della rete possono chiedere il canale trasmissivo contemporaneamente.

I nodi di una rete CAN non sono caratterizzati da indirizzi di rete, come su rete ethernet, ma i messaggi vengono inviati in brodcast con un identificativo univoco per la rete, il quale indica anche la priorità dell'informazione inviata.

L'identificativo deve essere deciso come specifica della rete can che si va ad assemblare e deve essere correttamente impostato nel software di tutti i dispositivi, chiamati ECU (electronic control units), presenti nella rete.

Facendo riferimento alla schematizzazione in livelli definita dall'ISO (International Standard Organization) col progetto OSI (Open System Interconnection), si può ritenere che il Bus CAN implementi il **Physical Layer** ed il **Data Link Layer**, ovvero i due livelli più bassi della pila, come è logico per un protocollo più vicino ad un Bus di Campo (forma di comunicazione digitale dedicata ai sistemi a basso livello) che ad una rete informatica del tipo Ethernet.

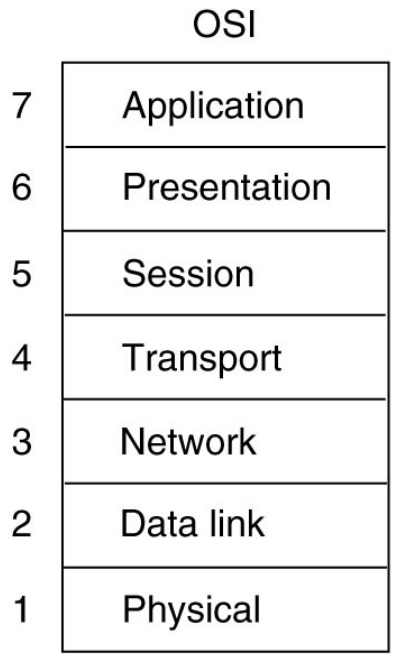


Figura 2: Livelli ISO/OSI

2.1 Livello Fisico - Physical Layer (Livello 1 OSI fisico)

Parliamo ora del primo livello della pila ISO-OSI riguardante il CAN. Esso è standardizzato in accordo con ISO 11898.

Secondo il modello ISO-OSI (ISO11898) il livello fisico deve soddisfare principalmente la specifica inerente il mezzo trasmissivo.

Questo infatti nel bus CAN deve essere un singolo canale bidirezionale, che può essere di tipo differenziale o a cavo singolo e terra (meno usato del primo). Solitamente si usa un doppino intrecciato, schermato o meno a seconda della rumorosità (elettrica e magnetica) dell'ambiente.

Un problema molto importante, nella trasmissione di segnali, sono i disturbi di tipo RFI (Radio Frequency Interferences). I disturbi RFI sono generati involontariamente (o volontariamente) dagli apparati tecnologici che oggi ci circondano. La prima linea di difesa contro le RFI è schermare i cavi con un conduttore metallico che poi viene messo a terra. Per compensare i limiti di questa metodologia, si utilizza la tecnica chiamata DIFFERENCIAL.

Questa tecnica sfrutta due cavi attraverso i quali viene inviato rispettivamente il segnale trasmesso come positivo, e lo stesso segnale trasmesso in negativo. I disturbi, che penetrano il cavo, saranno presenti in entrambi i cavi e avranno valori uguali. Il ricevente non deve far altro che applicare la differenza tra i due segnali, in modo tale che la parte di rumore risulterà scomparsa. Questo metodo funziona bene solo se nei due cavi c'è lo stesso disturbo. Per ottenere questo risultato è necessario che i cavi occupino lo stesso spazio. Per arrivare a una legittima approssimazione, si avvolgono i due cavi insieme (Twisted Pair).

ESEMPIO:

Il doppino è composto da due linee, una denominata H e una L (high e low). Per rappresentare il bit 0, entrambe le linee restano a 2,5V (la loro diferenza è di 0 volts).Il valore logico 1 viene rappresentato portando la linea H a 3,5V (rispetto a massa) e la linea L a 1,5V (Con d.d.p. tra le due linee di 2 volts).

La lunghezza massima del bus comporta una velocità velocità ridotta nella trasmissione (oltre 1Mbps entro 40m).

2.2 Data Link Layer (Livello 2 Osi: collegamento)

Il Data Link Layer è a sua volta implementato attraverso altri due livelli: l'Object Layer ed il Transfer Layer.

Il primo si occupa del filtraggio dei messaggi arrivati.

In una comunicazione broadcast tutti i nodi ricevono gli stessi pacchetti: nel'Object Layer si scartano quelli non rilevanti per il nodo considerato. Inoltre l'Object Layer si occupa della gestione dei messaggi da trasmettere e dell'interfaccia con l'Application Layer.

Il Transfer Layer si occupa di definire: il formato dei messaggi, l'arbitraggio per la contesa del canale trasmissivo, la segnalazione e la correzione degli errori, l'esclusione dei nodi mal funzionanti e filtraggio messaggi..

Il formato dei messaggi nel protocollo CAN sono in cinque differenti strutture, utilizzati in varie situazioni: Data Frame, Remote Frame, Error Frame, Overload Frame, Interframe Space.

In questo studio ci limiteremo, per esigenze di semplificazione, alla descrizione del Data Frame essendo questo l'unico frame trasmesso all'applicazione Android.

Data Frame (D.F.): è il tipo di messaggio più diffuso e permette la trasmissione dei dati da un nodo trasmettitore (TX) a tutti gli altri, che si comportano quindi come ricevitori (RX); ciascun nodo decide separatamente se ritenere rilevanti i dati ricevuti o se scartarli.

Il DF è costituito dai seguenti 7 campi:

Start of Frame (SoF): è costituito da un solo bit dominante e segnala l'inizio di un messaggio di tipo D.F. o di tipo R.F. . Ha anche una funzione di sincronizzazione per tutti gli altri nodi che riconoscono l'inizio della trasmissione.

Arbitration Field: è costituito dall'identificatore del contenuto del messaggio più un bit RTR (Remote Transmission Request) che distingue fra D.F. e R.F.. L'identificatore è di 11 bit nel protocollo CAN 2.0A ("Standard CAN") e di 29 bit nel CAN 2.0B ("Extended CAN").

Control Field: è costituito da 6 bit, di cui 4 servono a specificare il numero di byte di cui è composto il messaggio vero e proprio (Data Field), mentre 2 sono riservati a future espansioni del protocollo.

Data Field: contiene i dati veri e propri, che vanno da un massimo di 8 bytes (64bit) ad un minimo di 0, come indicato nel DLC. I bytes vengono inviati dal più significativo al meno significativo.

CRC Field: è costituito da 16 bits, di cui i primi 15 contengono la sequenza di controllo (cyclic redundancy check) mentre l'ultimo è un bit recessivo di delimitazione.

ACK Field: è costituito da un bit detto ACK Slot ed uno di delimitazione (ACK Delimiter).

End of Frame (EoF): è costituito da 7 bits recessivi che indicano la fine del Frame.

Start of Frame 1 bit	Arbitration Field 11 bit	Control Field 6 bit	Data Field 0 a 8 byte	CRC Field 16 bit	ACK Field 2 bit	End of Frame 7 bit

Figura 3: Data Frame CAN 2.0A

2.3 Livello Software

La definizione dell'Application Layer è infine lasciata interamente al progettista (non è standardizzato), al quale spettano i dettagli dell'interfacciamento degli utenti verso il bus.

Per quanto riguarda il sistema operativo Android esistono più applicazioni che si connettono tramite wifi o blueetooh al lettore CAN BUS ed esaminano i dati. Una per tutte CAN-Bus Analyzer (PCAN USB).

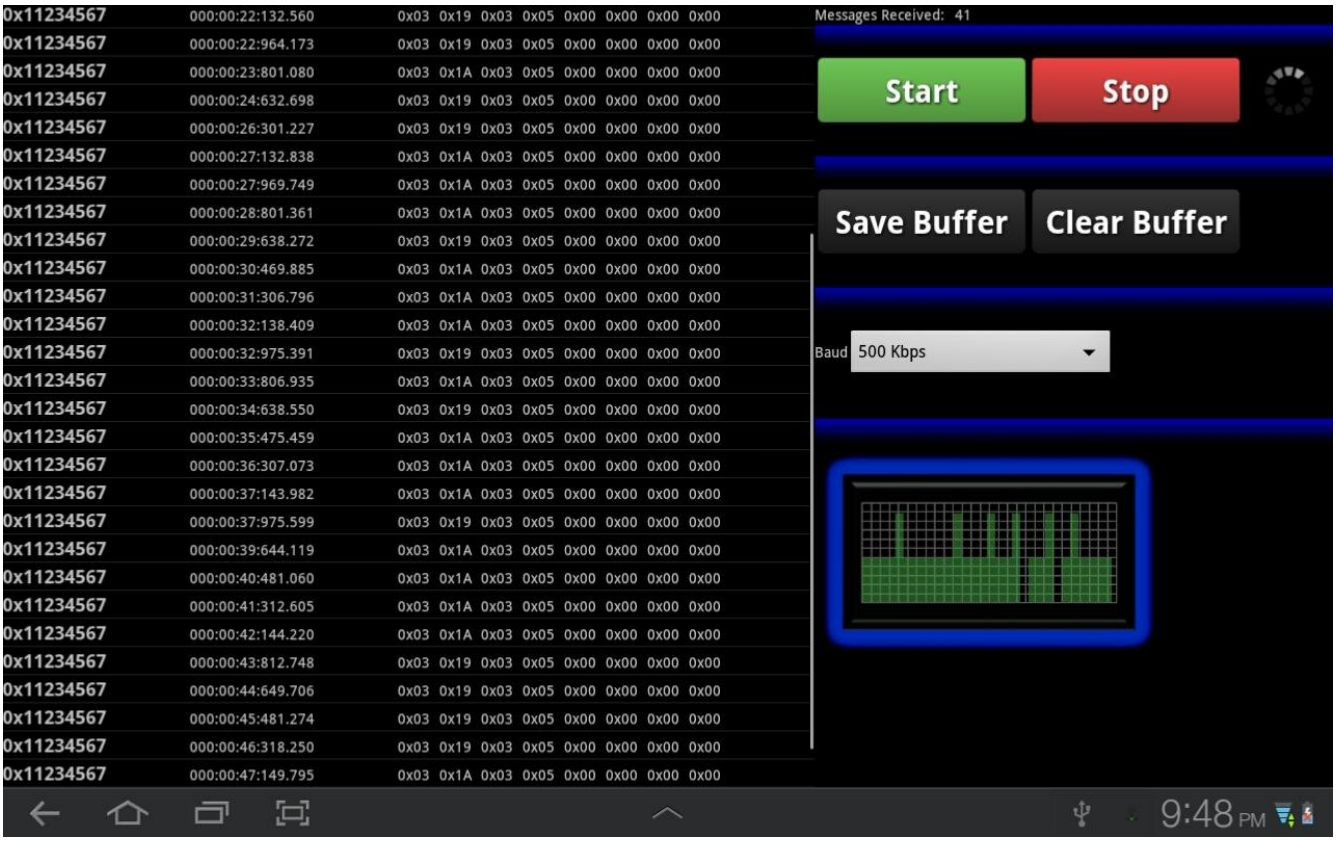


Figura 4: CAN-Bus Analyzer

Nelle prove per la comunicazione con linux-box e Android si utilizzano i comandi del pacchetto can-util installato e compilato insieme a SocketCAN (paragrafo 2.4).

Per effettuare un dump di tutte le informazioni inviate utilizziamo:

```
$ candump vcan0
…
vcan0  440  [8] 40 00 80 00 00 00 00 00   '@.......'
vcan0  442  [8] 42 00 80 00 00 00 00 00   'B.......'
vcan0  440  [8] 40 01 80 00 00 00 00 00   '@.......'
…
```

vcan0 è l'interfaccia utilizzata, il valore 440 o 442 è Arbitration Field o l'id dell'informazione trasmessa, [8] il Control Field e "40 01 80 00 00 00 00 00" gli 8 byte dell'informazione trasmessa il Data Field.

E' possibili salvare il dump del comando precedente in un file per poi riprodurlo e inviarlo nuovamente ad una interfaccia:

```
$ canplayer vcan0=vcan1 < candump-2013-05-06_190603.log
```

2.4 SocketCan

SocketCan utilizza le API di Socket Berkeley, lo stack di rete di Linux e implementa il dispositivo CAN driver come interfaccia di rete. SocketCan API è stata progettata il più simile possibile ai protocolli TCP / IP per consentire ai programmatori, di operare più agevolmente avendo familiarità con la programmazione di rete. Il concetto SocketCAN utilizza il modello dei dispositivi di rete, che consente a più applicazioni di accedere a uno o può dispositivo contemporaneamente. Inoltre, una singola applicazione è in grado di accedere a più reti CAN in parallelo.

Il concetto SocketCAN estende la Berkeley socket API in Linux con l'introduzione di una nuova famiglia di protocolli, PF_CAN (macro definita nel kernel linux in include/linux/socket.h), che coesiste con le altre famiglie di protocolli come PF_INET per il protocollo Internet.

La comunicazione con il bus CAN è quindi eseguita analogamente all'uso del protocollo Internet via socket.

SocketCan è un insieme di driver open suorce e stack di rete, sviluppato e mantenuto da Volkswagen per il Kernel Linux. Era inizialmente conosciuto come Low Level CAN Framework (LLCF).

I driver tradizionali CAN per Linux solitamente si basano sul modello dispositivi a carattere, tipo speciale di file che rappresenta una periferica (/dev/sda) o un dispositivo virtuale su cui è possibile effettuare operazioni di input/output per singoli byte. Tipicamente essi permettono l'invio e la ricezione dal controller CAN. Le implementazioni tradizionali di questa classe di driver consentono ad un unico processo di accedere al dispositivo, il che significa che tutti gli altri processi sono bloccati nel frattempo. Inoltre, tutti questi driver differiscono leggermente nell'interfaccia presentata all'applicazione, riducendo la portabilità.

Componenti fondamentali del SocketCAN sono i driver dei dispositivi di rete per diversi controller CAN e l'implementazione della famiglia di protocollo CAN. La famiglia di protocolli PF_CAN fornisce le strutture per consentire diversi protocolli sul bus: socket raw per la comunicazione CAN diretta e il protocollo di trasporto per le connessioni point-to-point. Inoltre il gestore di trasmissione che fa parte della famiglia di protocollo CAN fornisce altre funzioni, per esempio per l'invio di messaggi CAN periodici o realizzare filtri dei messaggi complessi.

La patch del CAN è stata aggiunta nel kernel 2.6.25 di Linux. Nel frattempo sono stati aggiunti alcuni driver dei controller e si continua a lavorare per aggiungere altri driver.

Comandi per linux per attivare un'interfaccia canbus:

```
# caricamento dei moduli can, can_raw, can_bcm, vcan
$ modprobe can
$ modprobe can_raw
$ modprobe can_bcm
$ modprobe vcan
# attivo con i diritti di root una scheda virtuale di rete vcan0 e
visualizzo lo stato
$ sudo ip link add dev vcan0 type vcan
$ sudo ip link set up vcan0
$ ip link show vcan0
3: vcan0: <NOARP,UP,LOWER_UP> mtu 16 qdisc noqueue state UNKNOWN
link/can

# setto alla scheda vcan0 il bitrate 125kbps, come nel nostro studio
$ sudo ip link set vcan0 type can bitrate 125000
```

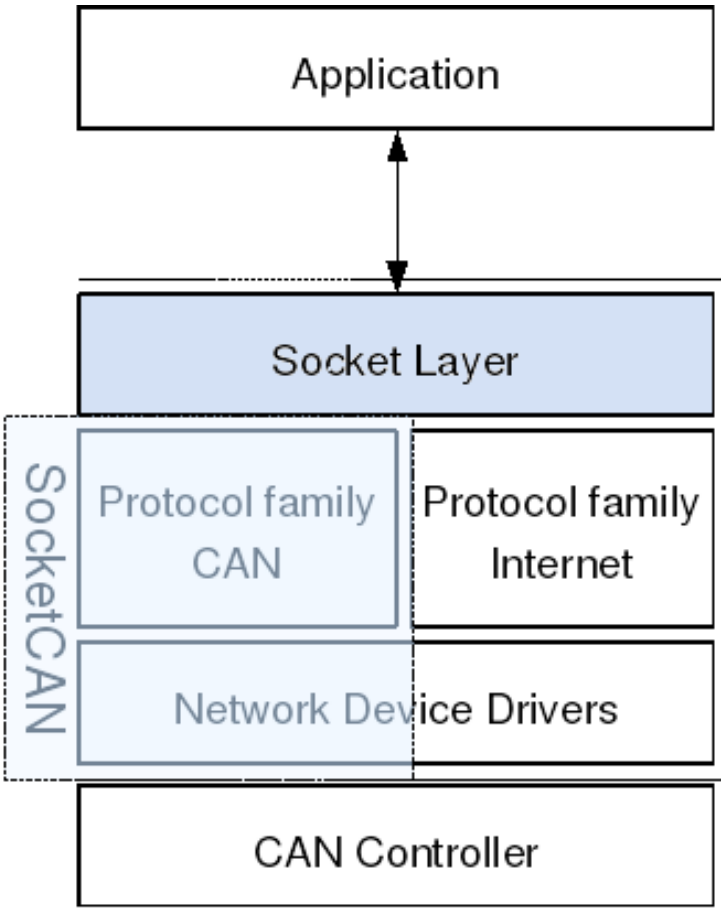


Figura 5: SocketCAN

Nel progetto preso in esame SocketCan si adatta molto bene alla reali necessità. La linux-box può essere collegata a più dispositivi can presenti nel veicolo e di conseguenza attraverso le interfacce di rete presenti è possibile inviare le informazioni al dispositivo Android.

3 Smartphone e sistema operativo

In questo capitolo si fornirà una panoramica generale degli smartphones di ultima generazione e si spiegherà cosa sono le app. Si procederà spiegando l'architettura di Android e il ruolo della Dalvik Virtual Machine.

3.1 App e sistemi operativi

I dispositivi mobili di ultima generazione, chiamati smartphone, stanno diventando i nuovi personal computer. Il computer desktop non sta comunque scomparendo, ma il mercato degli smartphone è in continua e rapida crescita.

I dispositivi vengono utilizzati abitualmente come computer da sempre più persone e per più scopi; sono convenienti per la loro portabilità e per riunire molte funzionalità (ad esempio ricevitore gps e fotocamera digitale per citare le più utilizzate).

Il mercato smartphone, nel secondo trimestre 2013 ha avuto un incremento del 46.5% (Fonte Gartner Agosto 2013) e le vendite di smartphone hanno totalizzato 225 milioni di unità. La più forte crescita degli smartphone si registra nelle aree Asia/Pacifico, America Latina ed Europa dell'Est, dove le vendite corrono, rispettivamente, al ritmo del 74.1%, 55.7% e del 31.6%.

I software che gestivano i telefoni delle passate generazioni erano molto rudimentali, non definibili come sistemi operativi e piuttosto simili gli uni agli altri. Ora nei cellulari di ultima generazione sono veri e propri Sistemi Operativi in grado di gestire i più svariati applicativi.

Oggi i sistemi operativi più usati sono cinque: Apple iOS, Android, Symbian OS, Windows 8 mobile, Blackberry OS.

Ogni smartphone è fornito di una serie di applicazioni preinstallate: un browser, una rubrica e un calendario. Quando si vuole aggiungere una funzionalità non presente, come ad esempio un software per l'ufficio, un videogioco o una immagine di sfondo è necessario installare l'applicazione deputata a tale scopo.

Attualmente le applicazioni per smartphone sono raccolte e distribuite tramite i cosiddetti "Markert" ogni produttore ha creato il proprio. Le due piattaforme che per prime hanno colto l'importanza dello sviluppo di nuove applicazioni sono state iOS e Android, per cui rispettivamente troviamo oggi disponibili all'incirca 1.000.000 applicazioni per l'App store e Google play.

3.2 Android: concetti fondamentali

La scelta del sistema operativo Android per il dispositivo mobile che useremo per la nostra tesi è stata determinata:

- dalla natura open source di Android
- da SDK dedicato bene documentato (che andremo ad approfondire più avanti)
- possibilità del thetering usb, indispensabile per la nostra applicazione
- compatibilità di Android con un elevato numero di smartphone

Android e un sistema operativo per dispositivi mobili sviluppato inizialmente da Android Inc. e acquisito da Google, il gigante di internet, nel 2005. Fondamentalmente Android non è sviluppato da zero, è infatti un sistema operativo che si basa su diverse versioni del kernel Linux, cio contraddistingue questo sistema operativo dagli altri per la sua natura open source e per la sua versatilità, infatti può funzionare su qualsiasi dispositivo mobile. Il primo smartphone dotato di piattaforma Android e stato l'HTC Dream, presentato il 22 ottobre del 2008. L'ultima versione di Android è la 4.4 KitKat rilasciata il 14 ottobre 2013 ha integrato numerose nuove funzionalità e tecnologie.

3.3 Architettura di Android

Il sistema operativo Android è basato su kernel Linux e consiste in una struttura formata da vari livelli o layer, ognuno di essi fornisce al livello superiore un'astrazione del sistema sottostante. La figura sottostante raffigura il cosiddetto Software Stack di Android.

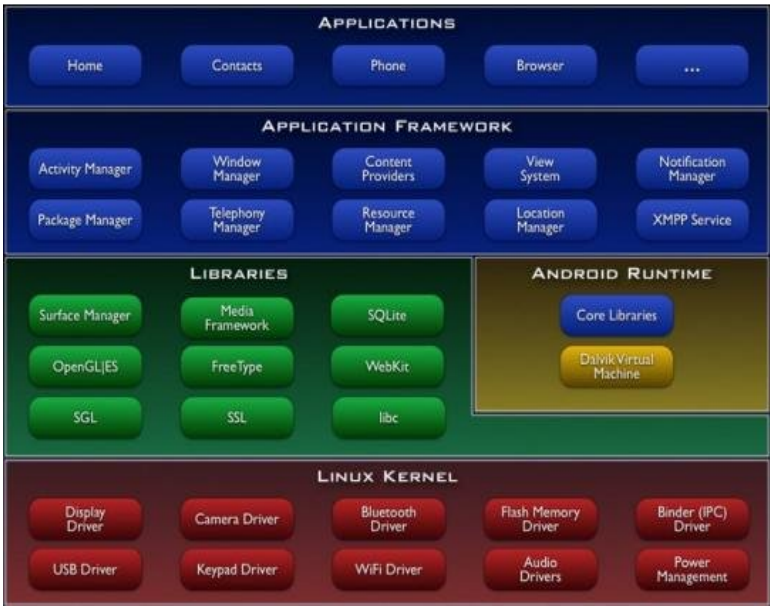


Figura 6: Lo stack di Android

L'**Applications Layer** è Il livello più alto dello stack è costituito dalle applicazioni non solo quelle native ma tutte quelle che vengono installate.

L'**Application Framework** Layer è basato su classi Java riutilizzabili su nuove applicazioni eseguite in una macchina virtuale ad hoc denominata Dalvik Virtual Machine (DVM).

Il livello **Libraries Layer** include una serie di librerie C/C++ che vengono usate da vari componenti del sistema. Attraverso l'Application Framework gli sviluppatori hanno accesso ai servizi forniti da queste librerie.

Android Runtime Layer è formato dalle cosiddette Core Libraries e dalla Dalvik Virtual Machine (DVM). Le Core Libraries includono buona parte delle funzionalità fornite dalle librerie standard di Java a cui sono state aggiunte librerie specifiche di Android.

La Dalvik Virtual Machine, che è una particolare macchina virtuale progettata appositamente per dispositivi mobili, su cui girano le applicazioni installate. Essa garantisce una certa indipendenza del software dalle varie architetture hardware possibili.

Un'essenziale caratteristica della Dalvik Virtual Machine è quella di riuscire a gestire in maniera molto parsimoniosa ed efficiente le poche risorse messe a disposizione dai dispositivi mobili.

Questo risultato è stato raggiunto grazie a numerosi sforzi e accorgimenti, come per esempio la rimozione di numerose librerie Java non necessarie ai dispositivi mobili o l'utilizzo di un particolare tipo di Bytecode (linguaggio intermedio più astratto tra il linguaggio macchina e il linguaggio di programmazione), diverso dal Bytecode Java.

L'adozione di un'architettura Register-Based 10 per la DVM, in contrapposizione alla natura delle JVM che sono stack machines, costituisce un altro esempio di ottimizzazione.

La DVM è stata concepita per poter essere eseguita in più istanze contemporaneamente sullo stesso dispositivo, ogni applicazione viene associata ad un processo che gira all'interno di una DVM ad esso dedicata. Dato che le varie DVM sono isolate, un eventuale malfunzionamento di un'applicazione non influenza le altre né mette in pericolo la stabilità del sistema operativo stesso.

Tra gli ultimi aggiornamenti che sono stati fatti alla DVM troviamo l'introduzione della compilazione Just In Time (JIT) che offre consistenti incrementi prestazionali.

La DVM si appoggia al kernel Linux per funzionalità quali la gestione di thread e la gestione della memoria a basso livello.

Alla base dello stack Android troviamo un kernel Linux nella versione 2.6. La scelta di una simile configurazione è nata dalla necessità di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante attraverso l'utilizzo di diversi driver.

3.4 SDK di Android

L'SDK (Software Development Kit) di Android è stato rilasciato dalla OHA (Open Handset Alliance) per la prima volta a novembre 2007 fornendo negli anni diversi aggiornamenti. Essa include gli strumenti di sviluppo, le librerie, un emulatore del dispositivo, la documentazione (in inglese), alcuni progetti di esempio, tutorial e altro. È installabile su qualsiasi computer che usi come sistema operativo Windows Xp/Vista/7/8, Vista, Mac OS X (dalla versione 10.4.8) o Linux. L'IDE (Integrated Development Environment) ufficialmente supportato per lo sviluppo di applicazioni per Android è Eclipse, per cui è fornito un plug-in progettato per fornire un potente ed un integrato ambiente in cui costruire le applicazioni. L'SDK è basato sul linguaggio di programmazione Java che fornisce una serie di API specifiche per mezzo del quale è possibile interagire con il sistema operativo Android, controllare l'hardware del dispositivo e lo sviluppo dell'interfaccia grafica.

Le applicazioni Android sono caratterizzate da una certa dualità: parti dinamiche scritte in Java e parti statiche scritte in XML. Le parti statiche possono essere quelle caratteristiche che non cambiano durante l'esecuzione dell'applicazione, come per esempio il design delle finestre; tipico delle parti dinamiche sono invece gli aspetti programmatici come per esempio la gestione degli eventi.
Per lo sviluppo delle applicazioni è disponibile una completa documentazione in http://developer.android.com.

Ai fini della programmazione, il team di Android ha specificato nella documentazione ufficiale vari termini per definire vari tipi di applicazioni:

Activities: sono quelle applicazioni destinate a una interazione diretta con l'utente utilizzando lo schermo e i dispositivi di input messi a disposizione dallo smartphone. Ad esempio un'applicazione per leggere le mail la prima attività è la lista delle mail, una seconda attività è la composizione di una mail nuova.

Services: applicazioni che per loro natura svolgono delle operazioni autonome e che vengono richiamati dalle attività al bisogno; gira in sottofondo e non interagisce direttamente con l'utente. Un esempio di servizio è com.android.inputmethod.latin, ossia il componente che fa comparire la tastiera quando si seleziona (con i tasti o con un "tocco" sul touch-screen) un campo di input testuale.

Content providers: gestisce un insieme condiviso di dati delle applicazioni. È possibile memorizzare i dati nel file system, un database SQLite, sul web, o qualsiasi altro luogo di memorizzazione persistente a cui l'applicazione può accedere. Attraverso il content providers, altre applicazioni possono interrogare o anche modificare i dati (se è permesso).

Per esempio, il sistema Android fornisce un content providers che gestisce le informazioni di contatto dell'utente. Come tale, qualsiasi applicazione con le autorizzazioni appropriate possono interrogare parte del fornitore di contenuti (come ad esempio ContactsContract.Data) per leggere e scrivere informazioni su una persona particolare.

Broadcast receivers: viene utilizzato quando si intende intercettare un particolare evento, attraverso tutto il sistema. Ad esempio lo si può utilizzare se si desidera compiere un'azione quando si scatta una foto.

3.5 Socket TCP/IP per android

Come descritto nella sezione 2.4, il sistema SocketCan permette di utilizzare il CAN-BUS montato sul veicolo come un dispositivo di rete. La linux-box del veicolo ha quindi presente nelle interfacce di rete il dispositivo chiamato vcan0 che invia i messaggi ricevuti dal dispositivo CAN in brodcast.

Verrà sfruttato in questo caso un'applicazione "ponte" che gira sulla linux-box, che redireziona i messaggi can ricevuti in un socket TCP con il dispositivo Android collegato tramite tethering usb.

A questo punto il dispositivo mobile utilizza i dati ricevuti e visualizza tutto tramite l'interfaccia utente.

Figura 7: linux-box veicolo e Socketcan

3.5.1 Definizione socket

Con socket si identifica un'astrazione software progettata per poter utilizzare delle API (Application Programming Interface) standard e condivise per la trasmissione e la ricezione di dati attraverso una rete oppure come meccanismo di comunicazione tra processi (IPC).

Il paradigma socket chiamato Berkeley nasce con il sistema operativo Unix BSD 4.2 (uscito nel 1983) come API. Solo nel 1989 l'Università di California Berkeley rilascia il suo sistema operativo e la libreria di rete con licenza libera dai vincoli di licenza Unix di proprietà di AT&T Corporation. Tutti i sistemi operativi moderni presentano ora qualche implementazione dell'interfaccia socket Berkeley, poiché quest'ultima è considerata l'interfaccia standard per la connessione a Internet.

Il socket, quindi, è il punto in cui il codice applicativo di un processo accede al canale di comunicazione per mezzo di una porta, ottenendo una comunicazione tra processi che lavorano su due

macchine fisicamente separate. Dal punto di vista di un programmatore, un socket è un particolare oggetto sul quale si leggono e scrivono i dati da trasmettere o ricevere.

In questo contesto tratteremo i socket Berkeley su protocollo IP, utilizzato da SocketCAN e da Android.

Elenchiamo la lista delle primitive fornite da Socket Berkeley:

socket(): crea un nuovo socket, allocando le risorse necessarie, ritorna un intero che identifica il socket

listen(): viene utilizzato sul lato server, e mette in ascolto un socket TCP

connect(): viene utilizzato sul lato client, e assegna un numero di porta libera al socket.

accept(): viene utilizzato sul lato server. Accetta un tentativo del client per creare una nuova connessione TCP, e crea un nuovo socket associato all'indirizzo.

send() e **recv()** o **write()** e **read()** , o **sendto()** e **recvfrom()**: vengono utilizzati per inviare e ricevere dati da/a un socket remoto.

close(): fa sì che il sistema liberi le risorse assegnate a un socket. In caso di TCP, la connessione viene terminata.

gethostbyname() e **gethostbyaddr()**: vengono utilizzate per risolvere i nomi host e gli indirizzi. Solo IPv4.

select(): viene utilizzato per fornire un elenco di socket pronti a leggere, o a scrivere, o che presentano errori.

poll(): viene utilizzata per verificare lo stato di un socket in un set di sockets. Il set può essere testato per vedere se ogni socket può essere scritto, letto o se si è verificato un errore.

getsockopt(): viene utilizzata per recuperare il valore corrente di una particolare opzione per il socket specificato.

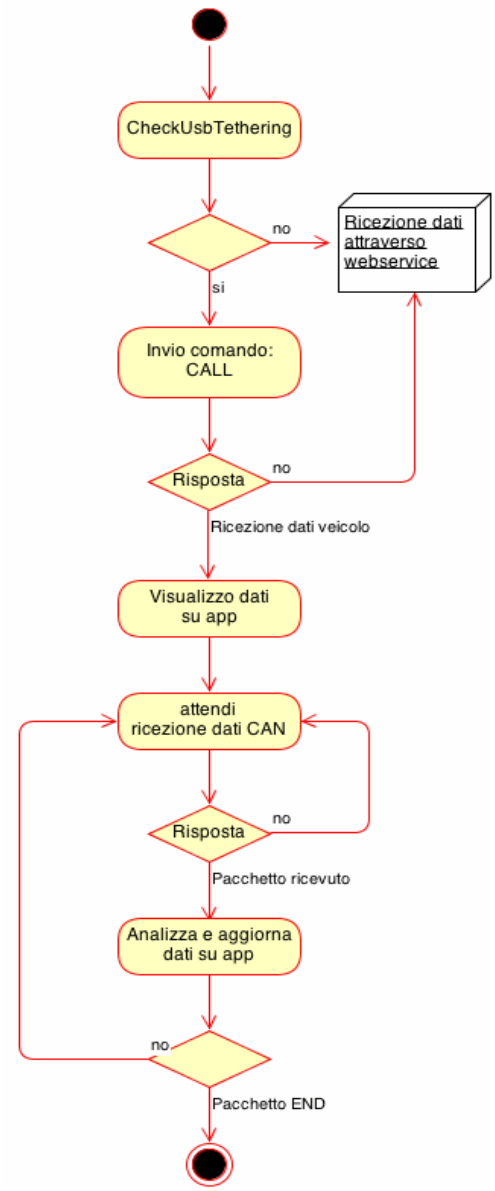
setsockopt(): viene utilizzato per impostare una particolare opzione del socket specificato.

Riportiamo l'esempio di codice per instaurare in Android un socket con un server predefinito:

```
…
import java.net.Socket;
…

public static final String SERVERIP = "192.168.42.10"; //IP address linux box
public static final int SERVERPORT = 4444;
…

Socket s = new Socket(SERVERIP, SERVERPORT);

BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(s.getOutputStream()));

out.write(outMsg);
out.flush();

String inMsg = in.readLine()

…
```

3.5.2 Protocollo comunicazione dell'applicazione con linux-box su thetering

Nell'applicazione di cui al presente studio verrà sviluppato in particolare un minimo protocollo di comunicazione tra la linux-box e lo smartphone. Nel seguente schema viene analizzata la comunicazione tra i dispositivi:



L’applicazione, come prima attività, controlla se il telefono è collegato alla porta usb tramite tethering alla linux-box. Se questo è verificato si connette ad una determinata porta ed invia il comando CALL.

Nella linux-box l’applicazione che si occupa di inoltrare i pacchetti CAN-BUS al telefono Android tramite il socket risponde al comando CALL, con l’invio delle informazioni della macchina come ad esempio il numero di matricola.

Da quel momento l’app Android rimarrà in attesa dell’arrivo del pacchetto CAN-BUS fino ad un pacchetto di fine comunicazione che convenzionalmente sarà il comando END, che determinerà la chiusura della comunicazione. Il pacchetto END sarà inviato dalla linux-box quando il veicolo elettrico verrà spento.

3.6 Tethering usb

L’applicazione sviluppata nella modalità connessione diretta a veicolo tramite usb, deve comunicare con la linux-box attraverso protocolli di rete, quindi è necessario instaurare tale infrastruttura tra gli apparati.

Android permette di attivare il tethering per offrire accesso alla rete ad altri dispositivi che ne sono sprovvisti. La connessione tra i due dispositivi può avvenire via wireless (Bluetooth o Wi-Fi) o in maniera cablata USB (è il nostro caso).

La tecnologia sfruttata in questo caso è quella che viene chiamata IP Masquerading in cui le connessioni generate da un insieme di computer vengono "presentate" verso l'esterno con un solo indirizzo IP.

Nella pratica Android, dopo l’abilitazione della funzione come nella figura 8 sottostante, attiva sull’interfaccia usbN (es: usb0) un server DHCP, un server DNS e un gateway fornendo in questo modo una connessione ad Internet ad un altro dispositivo. Il range di indirizzi ip forniti alla linux-box, dal server DHCP, è 192.168.42.1-255.

L’indirizzo della linux-box viene reperito risolvendo il nome di rete conosciuto a priori. Ad esempio:

```
import java.net.InetAddress;
InetAddress addr = null;
…
addr = InetAddress.getByName("linux-box");
…
```

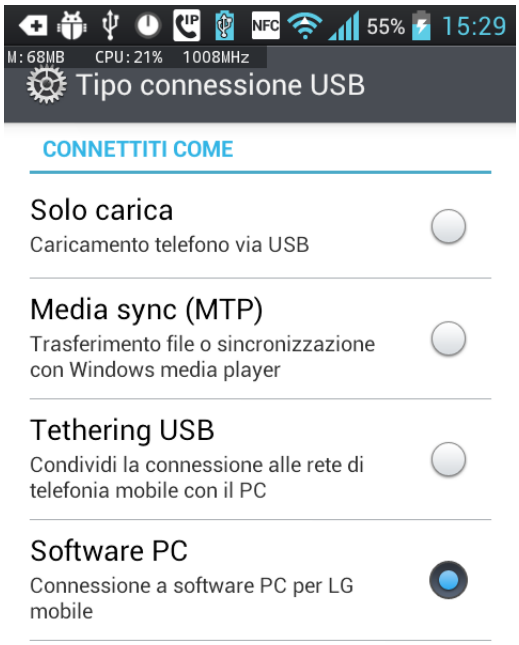


Figura 8: Selezione connessione Android

3.7 Webservice e REST su Android

Secondo la definizione data dal World Wide Web Consortium (W3C) un Web Service (o servizio web) è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete; caratteristica fondamentale di un Web Service è quella di offrire un'interfaccia software (descritta in un formato automaticamente elaborabile quale, ad esempio, il WSDL, Web Services Description Language) utilizzando la quale altri sistemi possono interagire con il Web Service stesso, attivando le operazioni descritte nell'interfaccia tramite appositi messaggi utilizzando dei particolari protocolli (il più famoso è il SOAP).
Un Web Service permette di essere trasparente al linguaggio di programmazione. Lo scambio dei dati avviene tramite il metodo GET o POST messi a disposizione dal protocollo HTTP oppure utilizzando direttamente il formato XML, tutti metodi indipendenti dal linguaggio di programmazione utilizzato.

Per il progetto in esame e per una più semplice implementazione dell'applicazione lato client e per un migliore utilizzo della banda disponibile (spesso si utilizzano formati più compatti del SOAP), si è scelto di utilizzare un REST (Representational State Transfer) Web Service.

REST è una particolare tipologia di architettura software per la comunicazione nei Web Service. Tale architettura utilizza principalmente, a livello di applicazione, il protocollo HTTP e non un protocollo proprietario (es. SOAP).

Inizialmente REST venne descritto da Fielding nel contesto del protocollo HTTP; un sistema RESTful, però, si può tranquillamente appoggiare ad un qualunque altro protocollo.

Perché un Web Service sia conforme alle Specifiche REST, deve avere alcune specifiche caratteristiche:

- architettura basata su client/server;
- stateless, cioè ogni ciclo di request/response deve rappresentare un'interazione completa del client con il server;
- uniformemente accessibile, cioè ogni risorsa deve avere un indirizzo univoco ed ogni risorsa di ogni sistema presenta la stessa interfaccia, precisamente quella individuata dal protocollo HTTP.

Il Web Service fornito risponde alle richieste REST in formato JSON.

JSON, acronimo di JavaScript Object Notation, è un formato adatto per lo scambio dei dati in applicazioni client-server. È basato sul linguaggio JavaScript Standard ECMA-262 3ª edizione dicembre 1999, ma ne è indipendente. La semplicità di JSON ne ha decretato un rapido utilizzo.

Questo fatto lo ha reso velocemente molto popolare grazie alla diffusione della programmazione in JavaScript nel mondo del Web.

I tipi di dati supportati da questo formato sono:

- booleani (true e false);

- interi, reali, virgola mobile;
- stringhe racchiuse da doppi apici (");
- array (sequenze ordinate di valori, separati da virgole e racchiusi in parentesi quadre [ ]);
- array associativi (sequenze coppie chiave-valore separate da virgole racchiuse in parentesi graffe);
- null.

La maggior parte dei linguaggi di programmazione possiede un typesystem molto simile a quello definito da JSON per cui sono nati molti progetti che permettono l'utilizzo di JSON con altri linguaggi quali, per esempio: ActionScript, C, C#, ColdFusion, Common Lisp, Delphi, E, Erlang, Java, JavaScript, Lua, ML, Objective CAML, Perl, PHP, Python, Rebol e Ruby.

Il seguente esempio è un semplice menu in JSON:

```
{
   "type": "menu",
   "value": "File",
   "items": [
      {"value": "New", "action": "CreateNewDoc"},
      {"value": "Open", "action": "OpenDoc"},
      {"value": "Close", "action": "CloseDoc"}
   ]
}
```

In Android, già dalle versioni di API level 1, sono presenti Classi per manipolare oggetti json. La classe è presente nel package org.json con i seguenti oggetti:

- JSONArray: array di valori indicizzati. I valori possono essere qualsiasi combinazione di JSONObjects, JSONArrays, stringhe, booleani, interi, long, doubles o NULL
- JONObject: classe per modificare o assegnare nomi o valori dell'oggetto jsJSONn questione. I valori possono essere qualsiasi combinazione di JSONObjects, JSONArrays, stringhe, booleani, interi, long, doubles o NULL
- JSONStringer: implementa toString(), ritorna una stringa dell'oggetto JSON
- JSONTokener: effettua il parser di una stringa e la codifica in JSON (RFC 4627).

3.7.1 Risorse del web service REST

Il web service REST è stato pensato e progettato per essere utilizzato da più applicazioni client eseguite su differenti sistemi operativi. Le richieste sono state implementate durante lo sviluppo dell'applicazione mobile seguendo le specifiche decise. Riportiamo di seguito le risorse individuate e sviluppate con le conseguenti risposte JSON.

Pur precisiamo che il servizio della linux-box preposto all'invio dei dati al web service non è incluso nel nostro studio, indichiamo comunque che i dati inviati fanno sempre parte dell'elaborazione dei pacchetti CAN-BUS indicati nel precedente capitolo.

Con il valore {user} è definito un identificativo univoco per individuare l'utente proprietario del veicolo.

Di fatto il client richiede al server di elaborare dei dati che gli vengono spediti via rete (nella fattispecie viene utilizzata la rete mobile del cellulare) e restituiti dopo l'elaborazione effettuata dal server.

Le richieste sono:

- http://server/version : informazione di versione del web service per controlli su retrocompatibilità delle applicazioni
- http://server/{user}/info : informazioni dell'utente, utilizzabili per effettuare controllo su contratti di manutenzione o assistenza del veicolo
- http://server/{user}/vehicles/list : lista dei veicoli di proprietà
- http://server/{user}/vehicles/list/error : lista dei veicoli di proprietà che contengono errori
- http://server/{user}/vehicles/list/charge : lista dei veicoli di proprietà in carica
- http://server/{user}/vehicle/{numero matricola}/info : dati del veicolo corrispondenti alla classe vehicle descritta nel paragrafo 4.4
- http://server/{user}/vehicle/{numero matricola}/position : coordinate del veicolo nel momento della richiesta
- http://server/{user}/vehicle/{numero matricola}/track : tracciato del veicolo dell'ultimo percorso registrato

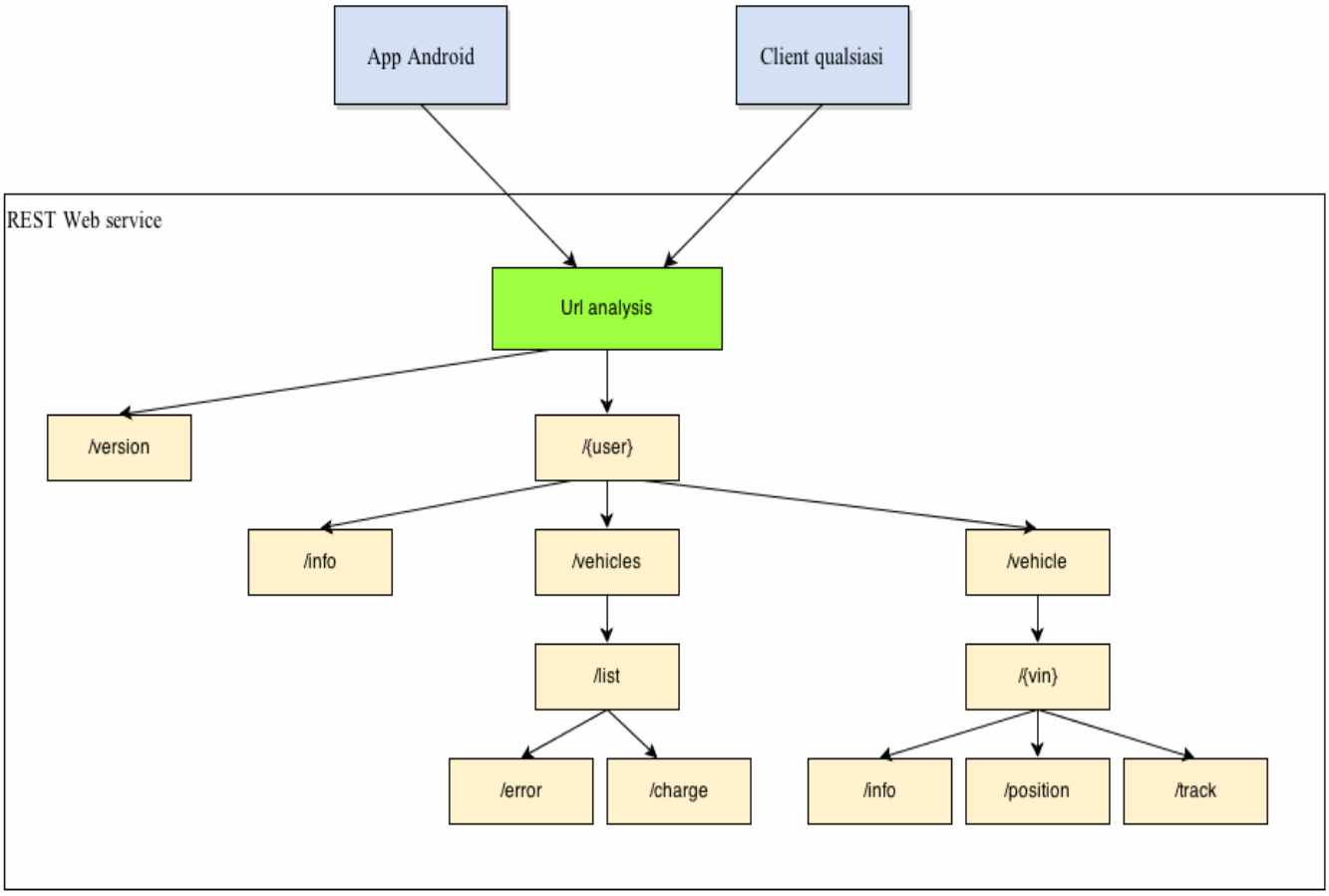


Figura 9: Schema web service rest

E' stato necessario implementare richieste specifiche per la sola posizione o per le macchine con errori per effettuare un veloce accesso nell'uso dell'app.

Infatti nell'aggiornamento della posizione della macchina nella mappa è utile un tempo di refresh di circa un secondo, poiché il server REST rispondendo solo con i dati indispensabili latitudine e longitudine del veicolo, riduce il traffico su rete, evitando congestionamenti.

Per la lista delle macchine con errori, la necessità di ridurre i dati trasferiti deriva dal fatto che l'applicazione effettuerà richieste frequenti, con un task a parte, come si vedrà nei capitoli successivi, per avvisare il più velocemente possibile l'utente di problemi o avarie nelle macchina.

Tutte le richieste REST prima specificate utilizzano il metodo http GET per completare l'operazione; attualmente non è stato necessario implementare altre risorse con altri metodi http differenti.

Si riporta di seguito in dettaglio la chiamata e la risposta del server:

```
* About to connect() to server.com port 80 (#0)
*   Trying xx.xx.xx.xx... connected
```

```
> GET /0001/vehicles/list HTTP/1.1
>  User-Agent:  curl/7.22.0  (x86_64-pc-linux-gnu)  libcurl/7.22.0  OpenSSL/1.0.1
zlib/1.2.3.4 libidn/1.23 librtmp/2.3
> Host:  server.com
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 19 Oct 2013 12:40:38 GMT
< Server: Apache/2.2.14 (Unix)
< Content-Type: application/json
< Content-Language: it
< Transfer-Encoding: chunked
<
{"vehicles": [{"soc": "90", "state": "charge", "vin": "ALKE3TMLY20K97148"}]}
* Connection #0 to host server.com left intact
* Closing connection #0
```

3.8 Google Maps Android API

Per il progetto esaminato, un ruolo importate viene svolto dalla possibilità dell'applicazione di individuare nella mappa la posizione del veicolo, o meglio dei veicoli, nel caso di una gestione flotte.

Google mette a disposizione un set di API per Android per utilizzare i dati di Google Maps direttamente nell'applicazione. Le API gestiscono automaticamente l'accesso ai server di Google Maps, il download di dati, la visualizzazione della mappa, e le gesture sulla mappa.

È inoltre possibile utilizzare le chiamate API per aggiungere marcatori o altri oggetti grafici per completare la visualizzazione voluta. Nell'applicazione in studio questo risulta particolarmente utile per gestire segnalazioni sul percorso effettuato, ad esempio: anomalie in un determinato punto, velocità massime o consumi elevati del veicolo (si rimanda al capitolo successivo per approfondire).

Le API di Google Maps alla versione 2 (v2) sono distribuite come parte della SDK "Google Play services"; è quindi necessario installarle queste ultime nell'ambiente di sviluppo Eclipse da "Android Sdk Manager".

Per poter fare qualsiasi richiesta alle API di Google Maps, dobbiamo ottenere una API Key. La chiave è gratuita. La API key, deve essere inserirla nell'applicazione e devono essere impostati i permessi corretti perché possa essere utilizzata. Va inserita infine nel file AndroidManifest.xml.

Utilizziamo nel codice l'oggetto GoogleMap con la possibilità di definire il tipo di mappa visualizzata: standard, standard con nome delle strade o mappa base.

Per effettuare il posizionamento della mappa con le coordinate geografiche del nostro veicolo, viene utilizzata la chiamata al metodo pubblico moveCamera.

Sarà utilizzata in questo lavoro anche una particolare classe, messa a disposizione da Android, con il nome Location in android.location.Location, che ci permetterà di calcolare la distanza del minimo percorso tra due punti. Il metodo è distanceBetween che, tramite le coordinate passate come parametro, ritorna la distanza approssimativa tra i due punti.

4 Applicazione automotive per veicolo elettrico

L'applicazione studiata ha la funzione di consentire di visualizzare le informazioni, reperite da più canali come can-bus e web service, in maniera chiara e semplice.

Infatti l'utilizzatore del veicolo dovrà essere in grado, attraverso il dispositivo mobile, di reperire le informazioni utili per le attività svolte in breve tempo; in alternativa deve essere l'applicazione stessa, tramite pop-up, ad avvisare dell'esistenza di criticità.

Nell'uso quotidiano di un veicolo elettrico i parametri essenziali sono principalmente: autonomia in km, percentuale di scarica o di carica delle batterie, il percorso effettuato e possibilità di effettuare una diagnostica per individuare problemi nel veicolo.

Effettuiamo in questo capitolo una panoramica dello studio fatto, partendo dalle caratteristiche principali del veicolo elettrico utilizzato ed analizzando come le informazioni sono prelevate ed elaborate dall'applicazione.

4.1 Informazioni generali su un veicolo elettrico

Un veicolo elettrico è dotato di un motore elettrico che utilizza l'energia chimica accumulata in batterie ricaricabili. Un motore elettrico ha un rendimento energetico del 90%, sicché solo il 10% dell'energia elettrica utilizzata viene dissipata, mentre la restante parte è trasformata in energia meccanica.

Il paragone con un motore endotermico non sussiste: rispetto a quello elettrico, il propulsore a combustione interna, benzina o diesel, riesce a malapena ad arrivare ad un rendimento del 30%.

Altro grande vantaggio delle auto elettriche è di non emettere nell'ambiente CO_2 ed altri gas serra-inquinanti, fino ad arrivare all'impatto zero se le batterie sono ricaricate con energia prodotta da fonti rinnovabili. Le prestazioni degli attuali veicoli elettrici si stanno progressivamente avvicinando a quelle dalle auto a combustione interna.

Il lavoro svolto è stato effettuato su un veicolo elettrico esistente sul mercato l'Alke XT. Il veicolo Alke XT è formato da:

- batterie principalmente di due tipi al piombo acido o al piombo puro
- motore elettrico AC
- elettronica di controllo Curtis
- display multifunzione per monitorare le funzionalità del veicolo

Dati del modello XT420E/EL:

Potenza 12 kW / 72 V
Velocità massima: 61 km/h o 32 km/h (marcia ridotta)

Capacità batterie: 13kWh - 26kWh
Autonomia fino a 75 km o 140 km con pacco batterie 26kWh
Numero batterie trazione 13kWh: 6x 12V 26kWh: 12x 12V

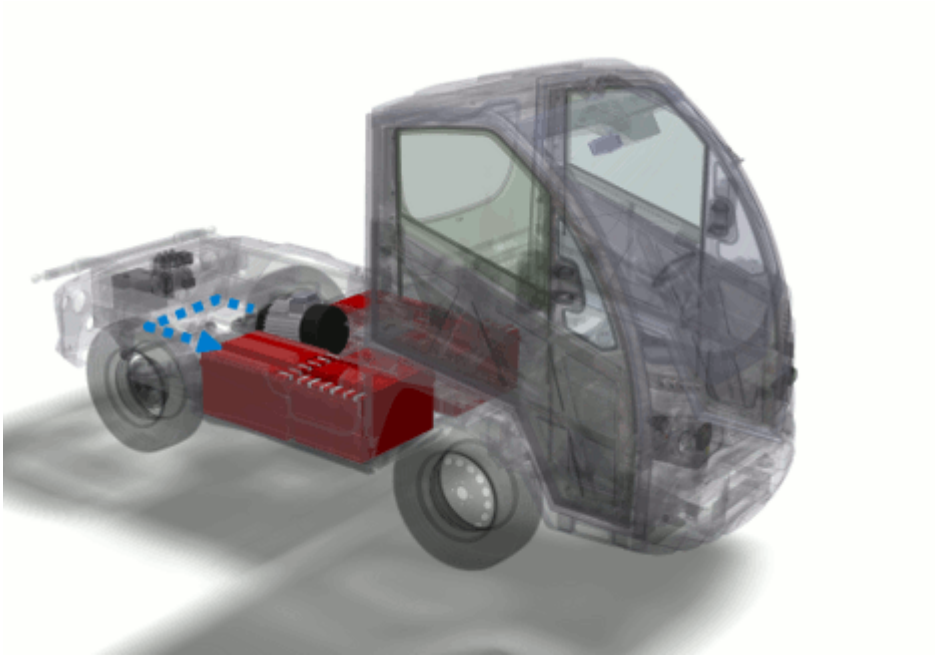


Figura 10: Veicolo Alkè XT

I test sono stati effettuati collegando lo smartphone alla linux-box montata su veicolo ed effettuando percorsi reali su strada urbana.

4.1.1 Autonomia

L'effettiva autonomia di un veicolo elettrico dipende da molteplici fattori ed è il dato più complesso da calcolare. Sono presenti fattori dipendenti dalla tecnologia costruttiva del veicolo e fattori che dipendendo dall'utilizzo effettivo dello stesso .

La tipologia, il peso e l'efficienza meccanica sono caratteristiche tipicamente costruttive e hanno un forte impatto sull'autonomia del veicolo, come avviene anche nei tradizionali veicoli a carburante combustibile.

Un secondo parametro riguarda il numero e il tipo di batterie impiegate, che contengono l'effettiva energia utilizzabile dal veicolo. Possono essere usate batterie al piombo-acido, batterie al NiMH (hanno una più alta densità di energia e possono consentire autonomie dell'ordine dei 200 km) oppure nuove batterie al litio, che consentono autonomie dell'ordine di 400–500 km.

I fattori che non dipendendo dalle caratteriste costruttive sono:

- percorso effettuato: una strada sterrata o di montagna comporta un aumento di consumo dell'energia e di conseguenza una diminuzione dell'autonomia
- stile di guida: le alte velocità determinano un consumo alto di energia quindi un'autonomia ridotta
- temperature: se la temperatura è di 25-35 °C il rendimento delle batterie è elevato e quindi si hanno consumi bassi
- carichi nel cassone: per il veicolo utilizzato nello studio è necessario tenere conto anche se stiamo sfruttando o meno la capacità di carico, poiché se stiamo effettuando un trasporto di merci, i consumi aumentano.

Nel nostro studio l'autonomia residua viene trasmessa dal veicolo, perché calcolata dal body computer presente nello stesso; l'applicazione comunque attua dei controlli sul dato inviato, incrociandolo con lo stile di guida o la distanza dal punto di arrivo ed effettua le opportune azioni:

“Suggerimento diverso stile di guida”: Se l'autonomia è inferiore alla distanza minima calcolata dall'applicazione, verrà suggerito all'utente di cambiare lo stile di guida (ad esempio se l'utente utilizza il veicolo con la marcia fast1 verrà suggerito di passare alla marcia fast2).

4.1.2 Carica/Scarica batterie

In un veicolo elettrico le batterie sostanzialmente subiscono continui cicli di scarica/carica. Per il veicolo Alkè XT i due tipi di batterie usate, al piombo puro oppure al piombo acido, hanno rispettivamente una durata di circa 2 anni (400 cicli) oppure una durata di 4/5 anni (1000-1500 cicli).

Più sopra abbiamo descritto come può avvenire la scarica incidendo sull'autonomia, ma un altro elemento importante è la carica delle batterie. Il veicolo Alkè ha a bordo un caricabatterie elettronico standard che permette alla macchina elettrica di caricarsi direttamente dalla rete ENEL su una normale presa di corrente 220V (10A). Questo rende possibile la carica dell'auto elettrica in qualsiasi posto.

Per un set di batterie da trazione sono necessarie 8 ore; è possibile anche caricare parzialmente le batterie in quanto non soffrono del cosiddetto "effetto memoria". E' possibile dunque fare una carica di circa 2 ore durante la pausa pranzo aumentando l'autonomia fino al 30%.

L'applicazione sviluppata ha a disposizione, per effettuare i controlli sulle batterie, il valore chiamato soc (State of Charge) che rileva la percentuale di carica delle stesse. L'uso principale di questo valore nel nostro studio è la valutazione della carica delle batterie, soprattutto quando l'utente non è nelle vicinanze del veicolo. L'azione intrapresa dall'app sarà:

“Avviso batterie cariche alla soglia impostata” il fruitore dell'applicazione imposta una soglia di carica delle batterie, al raggiungimento della quale verrà avvisto.

4.1.3 Percorso effettuato

Il veicolo elettrico è dotato di un ricevitore gps collegato alla rete CAN-BUS, quindi in grado di inviare le coordinate ai dispositivi della rete can oppure inviarle al web service tramite la linux-box.

Le coordinate sono trasmesse in due valori: la latitudine che va da -90.0° a 90.0° e la longitudine che va da -180.0° a 180.0°.

L'applicazione elabora i dati ricevuti tramite le api di Google Map viste nel paragrafo 3.8, intervenendo con determinate azioni in casi specifici:

“Avviso se l'autonomia residua non è sufficiente” il cliente può impostare nell'applicazione il punto di arrivo. Tale parametro verrà confrontato con l'autonomia residua e, se necessario l'applicazione mostrerà l'avviso nel caso in cui la carica delle batterie non sia sufficiente per raggiungere il punto di destinazione.

4.1.4 Diagnostica

Gli apparati elettronici presenti nella rete can del veicolo inviano determinati errori in caso di avarie o di situazioni critiche quali: sovratemperature del motore o dell'elettronica, sovratensioni o sovracorrenti oppure tensioni troppo basse per la batteria.

Nel bus can viene inviato l'errore nel momento in cui è rilevato, quindi l'applicazione Android dovrà visualizzare l'errore nello stesso momento, con le dovute segnalazioni nel caso di gravi anomalie.

Per il web service vengono memorizzati e messi a disposizione gli ultimi 10 errori: questo permette una consultazione dei precedenti. Questa gestione si presta molto bene al caso d'uso tipico del web service per una valutazione della diagnostica in un punto non prossimo alla macchina.

4.2 Rete dati su veicolo elettrico (CAN BUS)

Abbiamo descritto nel capitolo 2 i livelli del protocollo CANBUS, utilizzato soprattutto nel campo automotive. Nel veicolo elettrico preso in considerazione viene utilizzato un canale can da 1 Mbps e vengono utilizzati tutti gli accorgimenti hardware descritti.

I dati principali utilizzati per la comunicazione CAN-BUS tra veicolo e linux-box si compone di un ciclo in 5 frame con ID: 0x0A1, 0x1A1, 0x2A1, 0x3A1 e 0x4A1.

Le tempistiche di invio sono di circa 50 ms per completare l'intero ciclo dal primo id al quinto id.

4.3 Dettagli implementativi per protocollo can-bus

Nello specifico la linux-box invia allo smartphone Android, tramite il socket stabilito, i dati CAN-BUS pressoché uguali a i frame ricevuti dal dispositivo CAN. Le tempistiche di ricezione sono le stesse di invio dal dispositivo CAN.

Per effettuare una comunicazione efficiente senza "sprecare" banda con caratteri non utili vengono inviate stringe a lunghezza fissa nella quale i campi sono definiti. Ad esempio:

000157320A180102500000000000

00015732 = contatore incrementale in millisecondi utilizzato per sincronizzare i dati ricevuti
0A1 = ID del frame CAN
8 = il Control Field
01 02 50 00 00 00 00 00 = i dati veri e propri

Effettuiamo un calcolo della banda occupata:
ogni pacchetto è formato da 28 byte = 224 bit in 0,050 sec devono transitare 5 frame quindi:

224 x 5 = 1120 bit

in un secondo sono:

20 x 1120 = 22400 → 22,4 kbps.

Inviare il contatore incrementale dei millisecondi è indispensabile per la sincronizzazione dei dati sull'interfaccia utente; infatti valutando la differenza tra i frame ricevuti è possibile ottenere l'esatta sequenza di visualizzazione.

La comunicazione con can-bus viene sfruttata per inviare dati con velocità di aggiornamento elevata; ad esempio la velocità del veicolo.

4.4 Dati per la gestione di un veicolo elettrico

Il procedimento di progettazione per l'applicazione ha individuato la classe principale per l'astrazione del problema ovvero la classe vehicle, utilizzabile indipendentemente dalla connessione del veicolo e con all'interno tutti i dati e i metodi per la visualizzazione di essi.

Lo schema di questa classe viene seguito anche dalle risposte JSON per le risorse REST; questo permette un'agevole conversione delle risposte in nuove istanze di classi vehicle.

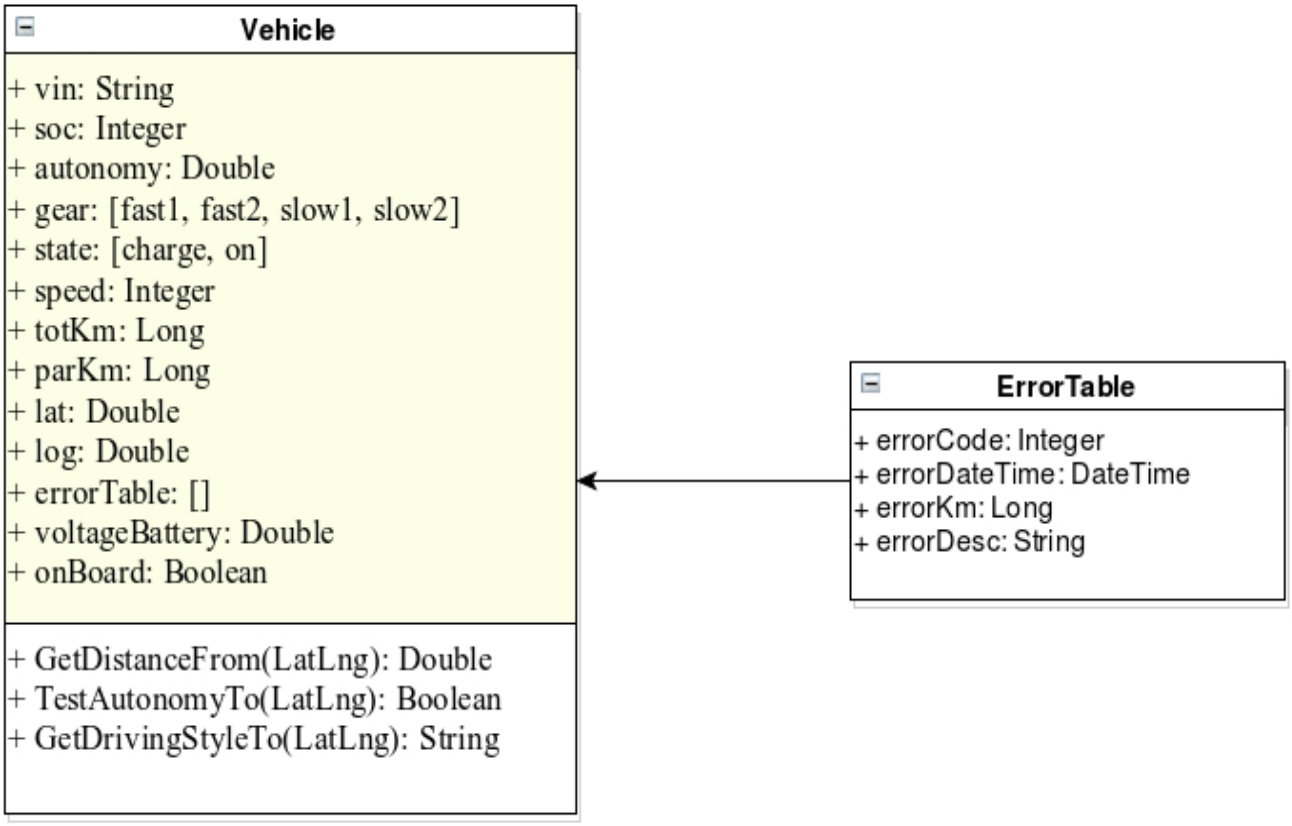


Figura 11: classe Vehicle

Descrizione delle variabili:

vin: String : numero univoco di matricola

soc: Integer : State of Charge, indicatore di scarica delle batteria da 0 a 100

autonomy: Double : indica in km l'autonomia residua

gear: [fast1, fast2, slow1, slow2] : 4 modalità di marcia

- fast1: modalità veloce ma limitata sulle accelerazioni e sul raggiungimento della velocità massima
- fast2: modalità veloce senza nessun limite
- slow1: modalità lenta ma limitata sulle accelerazioni e sul raggiungimento della velocità massima
- slow2: modalità lenta senza nessun limite

state: [charge, on]: in carica o accesso e in movimento

speed: Integer: velocità

totKm: Long: km totali del veicolo

parKm: Long: km parziali del veicolo azzera bili dal body computer

lat: Double: latitudine del veicolo ricevuta dal gps montato a bordo

log: Double: longitudine del veicolo ricevuta dal gps montato a bordo

errorTable: []: tabella degli errori diagnistici

voltageBattery: Double: tensione della batteria

onBoard: Boolean: variabile booleana che tiene conto se lo smartphone è collegato con usb a bordo

Per la classe vehicle sono state implementati i seguenti metodi:

GetDistanceFrom(LatLng): Double

Fornisce la distanza minima del veicolo da un punto geografico passato come parametro

TestAutonomyTo(LatLng): Boolean

Restituisce il valore vero se l'autonomia è sufficiente per raggiungere il punto geografico passato come parametro

GetDrivingStyleTo(LatLng): String

Restituisce una stringa riportando lo stile di guida da adottare per raggiungere il punto geogarfico del parametro

Altra classe rilevante per lo studio dell'app è il can_frame. Classe che ricalca a tutti gli effetti le classi implementate nel kernel linux per i frame can del socketcan, con l'aggiunta di un valore per tenere conto del tempo trascorso.

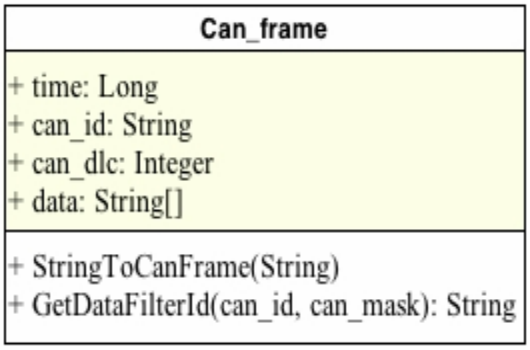


Figura 12: classe Can_frame

time: contatore incrementale in millisecondi

can_id: l'identifacatore del messaggio can

can_dlc: lunghezza del campo data

data: array di sringhe che contiene i dati ricevuti

I metodi indispensabili per effettuare l'elaborazione dei pacchetti can sono:

StringToCanFrame(String)

Carica la stringa ricevuta dal socket nell'istanza della classe

GetDataFilterId(can_id, can_mask): string

Ritorna il valore esadecimale corrispondente al mascheramento passato con can_mask del frame can con id sempre passato come parametro

4.4.1 Schema a blocchi dell'applicazione

Abbiamo specificato nei capitoli precedenti quali classi intervengono nell'applicazione per gestire le due comunicazioni, can-bus e rest web service. Proseguiamo lo studio descrivendo nei dettagli come i dati vengono processati da alcuni thread opportuni.

Il processo inizia con la verifica dell'esistenza di una comunicazione tramite can-bus o rest; se viene verificata la connessione tramite usb, verrà creato un thread (thCanTcpClient) per ricevere i dati ed elaborarli, il quale passerà i valori ad un'istanza della classe vehicle e al thread di aggiornamento analizzato successivamente.

Il thread thCanTcpClient effettuerà la procedura di connessione utilizzando il protocollo del paragrafo 3.5.2, tramite il comando CALL, effettuando la chiamata e recuperando le informazioni del mezzo; successivamente attenderà stringhe della forma di cui al paragrafo 4.3 ed effettuerà la relativa codifica.

Se il processo di riconoscimento identificherà la connessione al web service rest, verrà richiesta e visualizzata la lista dei veicoli e successivamente il processo principale si occuperà di aggiornare la lista ogni 30 secondi, tramite la chiamata al web service “http://server/{user}/vehicles/list”.

Questo processo si occuperà anche della gestione dell'avviso quando le batterie raggiungono la soglia impostata dall'utente. La lista dei veicoli sarà selezionabile e nel momento in cui si selezionerà un veicolo verrà passato tutto alla classe vehicle per la visualizzazione del dettaglio.

A questo punto, indipendentemente dalla connessione, la visualizzazione del dettaglio sarà assegnata alla classe vehicle e al thread thUpdateDetailVehicle di aggiornamento. Il thread di aggiornamento si occuperà anche della gestione di due notifiche:

- avviso se l'autonomia residua non è sufficiente, quando viene impostato il punto di arrivo
- suggerimento diverso stile di guida, se impostato il punto di arrivo

Il posizionamento del veicolo sulla mappa, all'avvio della schermata del dettaglio, sarà gestito da un thread che utilizzerà la chiamata “http://server/{user}/vehicle/{numero matricola}/position“ nel caso di web service oppure effettuerà la lettura delle coordinate nella classe vehicle per connessione usb.

Un processo chiamato thErrorVehicles, sempre in esecuzione con la connessione al web service, sarà dedicato alla gestione degli errori nella lista delle macchine e sarà in grado di avvisare l'utente con una notifica opportuna. La risorsa richiesta al web service, in questo caso, è “http://server/{user}/vehicles/list/error ”.

I thread utilizzati nello sviluppo dell'applicazione sono della classe Runnable di java.lang.Runnable, classe derivata dalla classe Thread ma pensata per aggiornare componenti grafici senza compromettere la stabilità dell'applicazione.

Riportiamo di seguito lo schema a blocchi dell'applicazione:

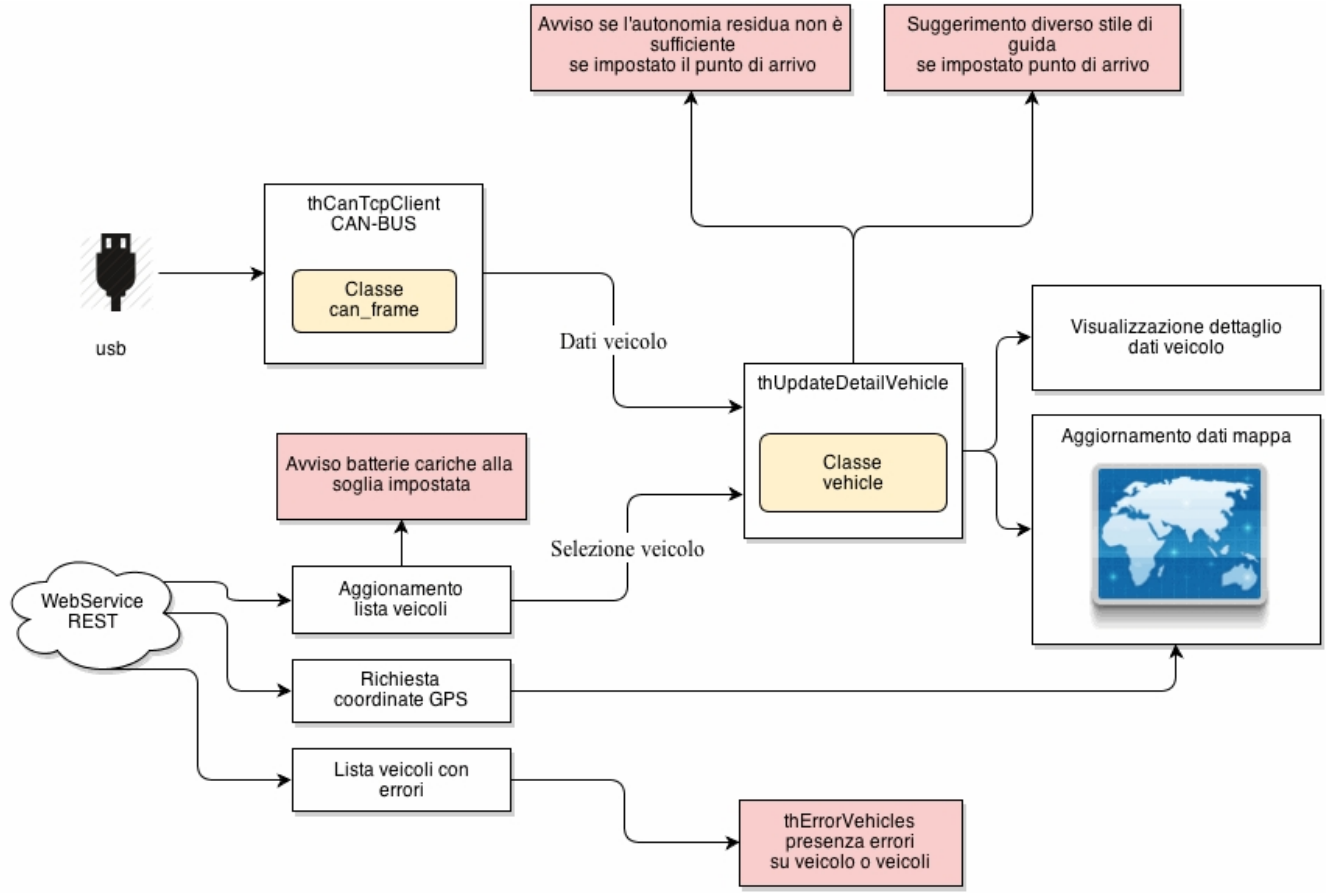


Figura 13: Schema a blocchi dell'app

4.5 Gestione Flotta

Con gestione flotta viene inteso l'insieme delle attività gestionali relative a tutti gli autoveicoli di una azienda. Per le imprese, i costi relativi ai mezzi di trasporto possono essere una componente rilevante dei costi aziendali, ai quali va riservata quindi la dovuta attenzione.

Per il nostro studio, le attività nelle quali l'applicazione sviluppata può essere d'aiuto nel controllo di una flotta di veicoli Alkè XT sono:

- attività logistica: collocazione del mezzo nel territorio, assegnazione delle attività in base all'autonomia del veicolo

- attività di mantenimento dei mezzi: manutenzione ordinaria e straordinaria per errori rilevati dalla diagnostica
- attività di gestione corrente: controllo consumi e percorsi effettuati
- attività di pianificazione: pianificazione consegne

Lo scopo della gestione della flotta è quello di migliorare l'efficienza e la produttività delle attività legate al settore trasporti delle aziende, riducendone i costi complessivi.

L'applicazione può essere installata su più dispositivi mobili. Nel caso di una flotta si può pensare che ogni operatore sia dotato di uno di essi e altri dispositivi siano a disposizione di chi è deputato all'organizzazione ed al controllo dell'attività. In questa maniera tutte le persone coinvolte nell'attività aziendale sono in grado di reperire le informazioni necessarie per ottimizzare le operazioni, ad esempio: chi organizza i servizi può individuare in maniera veloce i veicoli fermi per la carica o che richiedono manutenzione, oppure decidere se ridirigere un veicolo, destinandolo ad un determinato servizio in base all'autonomia residua.

5 Manuale

Dopo aver descritto le componenti che costituiscono l'applicazione e aver fornito i dettagli della comunicazione, in questa sezione verrà descritto come il software si presenta all'utente nelle operazioni di gestione.

La prima operazione che l'applicazione esegue è il controllo di connessione tramite usb alla linux-box. Dopo aver connesso il cellulare alla linux-box tramite il cavo usb, compare la dialog d'attesa di connessione: il riconoscimento sarà automatico e non richiede impostazioni particolari.

Figura 14: Ricerca rete can bus

Se la connessione tramite usb ha successo, l'app passa al dettaglio del veicolo descritto in seguito, altrimenti si procederà alla connessione tramite web service rest richiedendo la lista dei veicoli di proprietà. In questo momento è necessario la connessione ad internet del dispositivo mobile; in caso contrario comparirà l'avviso di mancanza di connessione.

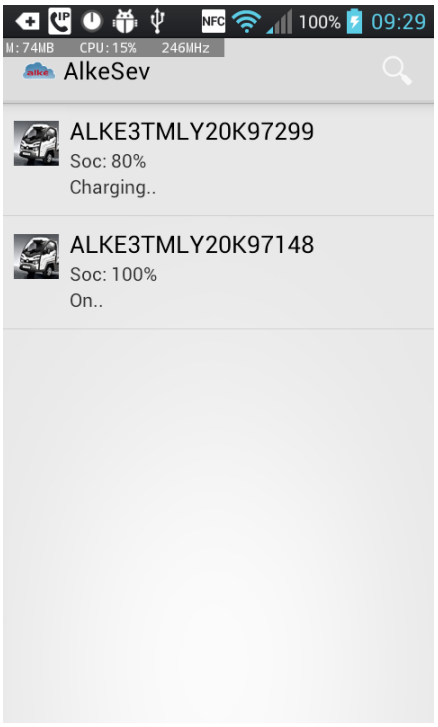


Figura 15: Lista veicoli

Nello screenshot d'esempio, è possibile constatare la presenza di due veicoli, dei quali viene riportata la matricola e lo stato. Come si può rilevare il primo mezzo è in carica all'ottanta percento e l'altro è acceso completamente carico.

A questo punto è possibile selezionare direttamente un veicolo oppure ricercarne uno tramite matricola attraverso la funzione di ricerca, premendo l'icona “lente” nella barra in altro a destra.

Quando l'utente seleziona il veicolo d'interesse, può visualizzare i dati in dettaglio. Questa schermata è la stessa visualizzata immediatamente dopo la connessione usb precedentemente descritta.

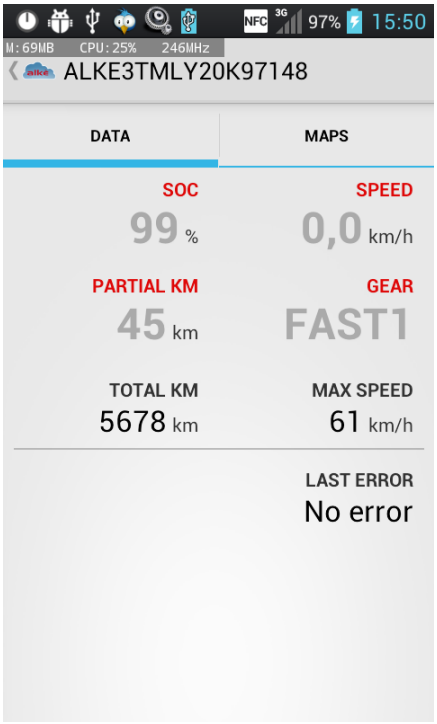


Figura 17: Dettaglio veicolo

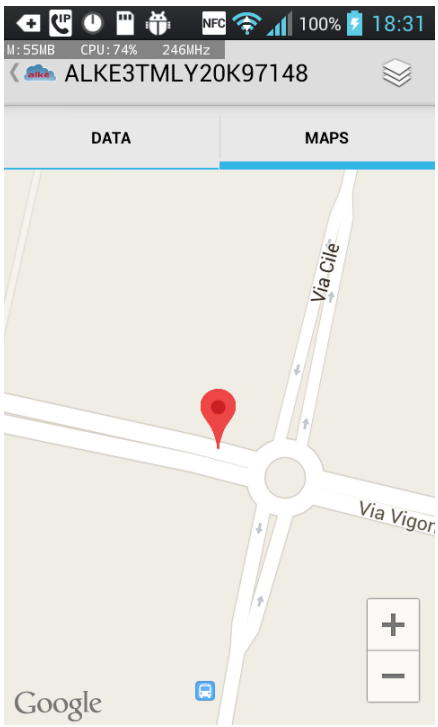


Figura 16: Dettaglio veicolo mappa

Il dettaglio si compone di due tab. Il primo tab con i dati del veicolo:

- soc
- velocità
- km parziali
- marcia impostata
- km totali
- velocità massima
- errori presenti, in questo caso non ci sono errori da segnalare

Nel secondo tab è possibile visualizzare la posizione geografica del veicolo aggiornata in tempo reale.

L'applicazione rimane nella videata del dettaglio durante la connessione usb o dopo la selezione del veicolo dalla lista.

Nel corso della connessione tramite web service, quindi nel caso in cui l'operatore non è in prossimità del veicolo, l'applicazione avvisa quando un mezzo controllato segnala errore, e dunque una richiesta di manutenzione.

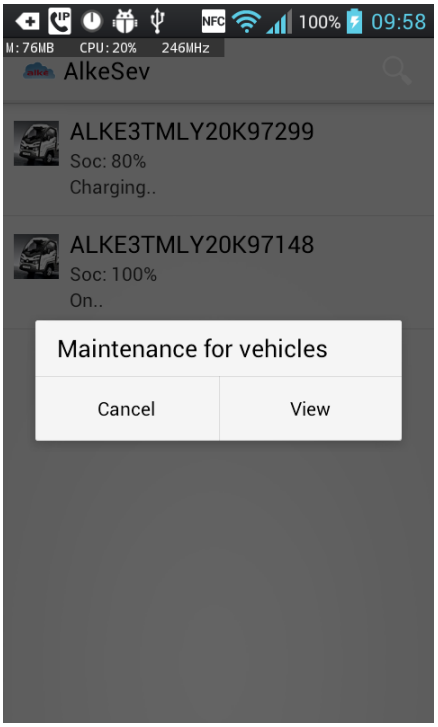


Figura 18: Lista veicoli

Abbiamo riportato la schermata del messaggio di allerta che consente la visualizzazione dei veicoli che richiedono manutenzione;

Qui sotto a sinistra la lista dei veicoli segnalati e a destra il dettaglio del veicolo con l'errore segnalato.

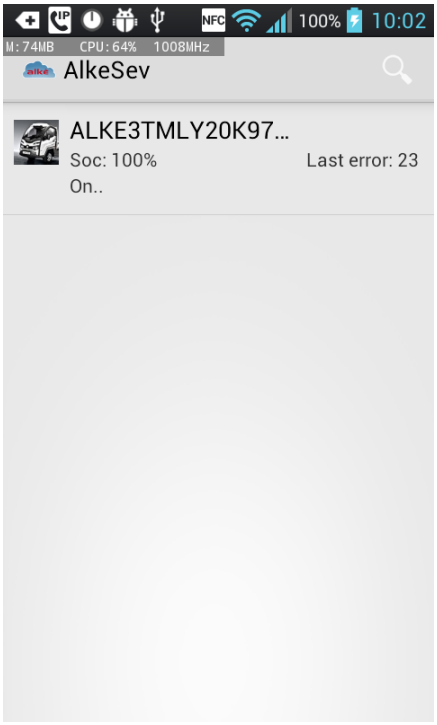


Figura 20: Dettaglio manutenzione

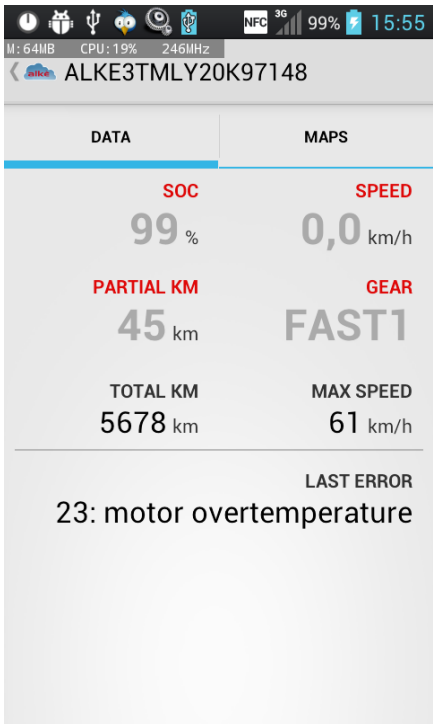


Figura 19: Dettaglio errori

L'utente, tramite le impostazioni dell'applicazione, nel dettaglio del veicolo, ha la possibilità di impostare la soglia del soc, per essere avvisato del raggiungimento della stessa, durante la carica. Questa funzionalità è attiva nel caso di connessione tramite web service, quindi può essere utile per il controllo remoto della carica del veicolo.

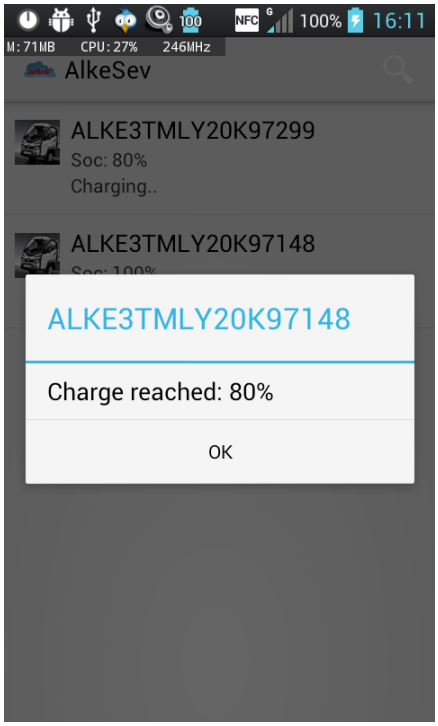


Figura 21: Soglia ricarica

L'applicazione è grado di calcolare il percorso minimo in km tra la posizione attuale del veicolo e il punto di destinazione. Il percorso minimo viene confrontato con l'autonomia e nel caso in cui questa non sia sufficiente, l'utente viene avvisato, con un warning sulla mappa. Il messaggio invita a cambiare lo stile di guida aumentando in questa maniera l'autonomia residua. Nel corso dell'uso l'avviso può comparire più volte.

A destra l'impostazione della destinazione, a sinistra l'avviso di cambio stile di guida.

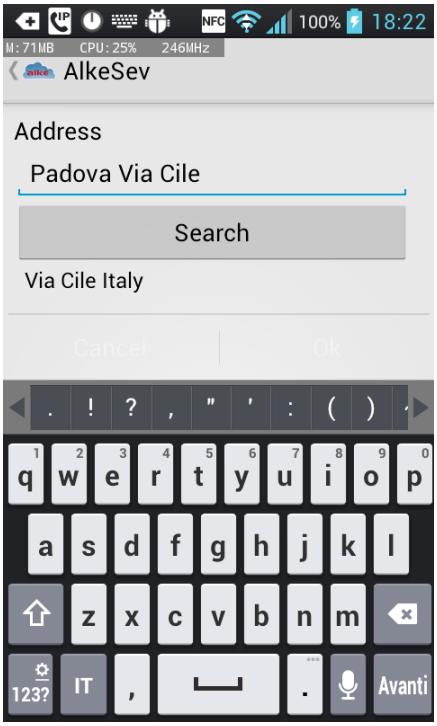


Figura 22: Selezione destinazione

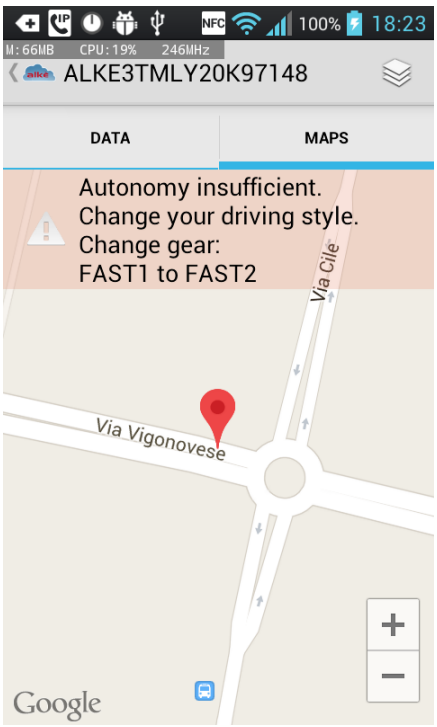


Figura 23: Avviso autonomia

Nell'impostazione della destinazione, l'applicazione confronta l'autonomia del veicolo selezionato con la destinazione impostata, avvisando immediatamente l'utente della possibilità di compiere o meno il viaggio.

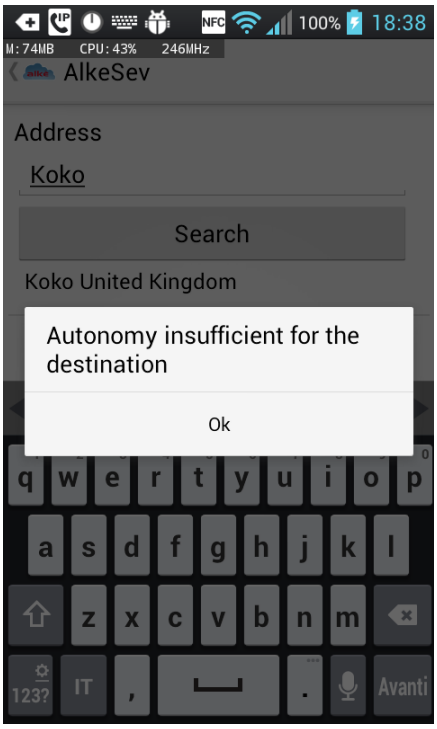


Figura 24: Avviso autonomia

6 Conclusioni

L’obiettivo principale della tesi era quello di sviluppare un’applicazione per smartphone che permettesse la visualizzazione di dati automotive su veicoli elettrici, evidenziando dati essenziali per l'uso di questa tipologia di mezzi: controllo della carica/scarica, indicazioni su autonomia, controllo della posizione geografica, indicazioni sullo stile di guida e diagnostica.

Lo studio ha descritto le caratteriste delle due modalità di connessione: la prima tramite usb, ricalcando la comunicazione can-bus, la seconda tramite il web service rest diffuso e efficiente.

L'interfaccia utente è stata pensata con uno stile essenziale per potere visualizzare i dati in maniera chiara. I messaggi di avviso (carica raggiunta, autonomia non sufficiente, lista veicoli che richiedono manutenzione) hanno ricoperto un ruolo importante nell'esporre le informazioni in maniera immediata.

L'utilizzo di Android ha permesso di sfruttare, in modo semplice ed efficace, sistemi di geo localizzazione e posizionamento su mappa, funzionalità indispensabili per l'obiettivo iniziale.

Attualmente l'applicazione è in versione beta e i test sono stati fatti a banco e su più veicoli elettrici Alkè XT percorrendo tragitti urbani. Si è potuto constatare che l'applicazione può essere di grosso aiuto per gestire al meglio l'autonomia o il tempo di ricarica del veicolo.

Uno sviluppo futuro di questo progetto potrebbe essere implementare un sistema di navigazione intelligente per veicoli elettrici, che comunichi con il mezzo in più modi. Il sistema potrebbe essere anche in grado di rilevare i punti di ricarica più vicini al veicolo e di reperire, comunicando anche con le stazioni di ricarica, dati quali la disponibilità o il tipo di stazione.

Appendice A

Risposte JSON del REST web service:

http://server/version : informazione di versione del web service per controlli su retrocompatibilità delle applicazioni

risposta:

```
{
	"versionWebService": "1.0.0",
	"versionJson": "2.0.0",
	"date": "2013-03-04 12:50:44Z"  // ISO international standard date format"
}
```

http://server/{user}/info : informazioni dell'utente, utilizzabile per effettuare controllo su contrattati di manutenzione o assistenza del veicolo

risposta:

```
{
	"user": [
	{
		"name": "Andrea",
		"assistance": "True",
		"endSupport": "2015-03-04"
	}
	]
}
```

http://server/{user}/vehicles/list : lista dei veicoli di proprietà

risposta:

```
{
	"vehicles": [
	{
		"soc": "90",
		"state": "charge",
```

```
                "vin": "ALKE3TMLY20K97148"
        }
        ]
}
```


http://server/{user}/vehicles/list/error : lista dei veicoli di proprietà che contengono errori

risposta:


```
{
        "vehicles": [
                {
                "vin": "ALKE3TMLY20K97148",
                "lastErrorCode": "23"
                }
        ]
}
```


http://server/{user}/vehicles/list/charge : lista dei veicoli di proprietà in carica

risposta:


```
{
        "vehicles": [
        {
                "vin": "ALKE3TMLY20K97148",
                "vin": "ZA9X3SLPC20K87149",
                "vin": "ZA9X3SLPC20K87150"
        }
        ]
}
```


http://server/{user}/vehicle/{numero matricola}/info : dati generali del veicolo

risposta:


```
{
        "ALKE3TMLY20K97148": [
                {
                "soc": "90",
                "state": "charge",
                "totKm": "8998",
```

```
                "parKm": "156",
                "lat":"45.670359",
                "lng":"12.235108",
                "voltagebattery": "72",
                "autonomy": "110",
                "gear": "fast1",
                "speed": "30",
                "errorTable": [
                                {
                                        "errorCode":"23",
                                        "errorDateTime":"2013-10-21T08:09:45Z",
                                        "errorKm":"8700",
                                        "errorDesc":""
                                },
                                {
                                        "errorCode":"47",
                                        "errorDateTime":"2013-10-10T08:45:45Z",
                                        "errorKm":"7701",
                                        "errorDesc":""
                                },
                                {
                                        "errorCode":"47",
                                        "errorDateTime":"2013-10-10T08:45:45Z",
                                        "errorKm":"7701",
                                        "errorDesc":""
                                }
                        ]
                }
        ]
}
```


http://server/{user}/vehicle/{numero matricola}/position : coordinate del veicolo


```
{
        "ALKE3TMLY20K97148": [
                {
                        "lat":"45.670359", "lng":"12.235108",
                }
        ]
```

}

http://server/{user}/vehicle/{numero matricola}/track : tracciato del veicolo dell'ultimo percorso registrato

risposta:

```
{
        "ALKE3TMLY20K97148": [
                {
                        "lat":"45.403073" "lon":"11.929510",
                        "time": "2013-03-06T08:37:28Z",
                        "name": "Partenza",
                },
                {
                        "lat":"45.403098", "lon":"11.929442",
                        "time":"2013-03-06T08:37:29Z",
                        "name": ""
                },
                ….
                {
                        "lat":"45.403125", "lon":"11.929395"
                        "time":"2013-03-06T08:37:33Z"
                        "name": "Fine percorso"
                }
        ]
}
```

Codifica dei pacchetti can bus su veicolo elettrico:

La velocità del veicolo viene recuperata dai frame con ID = 0A1 sul byte 2, convertendo il numero esadecimale in decimale:

0A1 8 xx 3C xx xx xx xx xx xx → 0x3C = 60 Km/h

i km totali byte 7 e 8 del frame 0A1

0A1 8 xx xx xx xx xx xx 13 88 → 0x1388 = 5000 Km totali

Per lo stato del veicolo byte 1 del frame 1A1 e byte 7 e 8 per l'autonomia

1A1 8 01 xx xx xx xx xx 00 5A → 0x01 = in carica e 0x005A = 90 km

Il frame con ID 2A1 è delegato al trasposto degli errori byte 1 il codice d'errore

2A1 8 17 xx xx xx xx xx xx xx → 0x17 = errore 23

Il frame con ID 3A1 è destinato alle coordinate latitudine gps

3A1 8 02 B4 CB C1 xx xx xx xx → 0 = segno +, 1 = segno -, 0x2B4CBC1 = 45403073 / 1000000 = 45.403073

Il frame con ID 4A1 è destinato alle coordinate latitudine gps

4A1 8 0B 60 7A 60 xx xx xx xx → 0 = segno +, 1 = segno -, 0xB607A60 = 11929510 / 1000000 = 11.929510