Problema affrontato

Si vuole progettare un software in grado di monitorare costantemente i parametri di un automobile, conoscerne in tempo reale la posizione e lo stato (Accesa, spenta, bloccata, ecc.).
L'utente finale deve quindi poter accedere all'auto in remoto, visualizzare i parametri e interagire con essa tramite il proprio computer.

Interviste

1. *Qual è la finalità per la quale si chiede la creazione del prodotto?*

   L'obiettivo si concretizza nella semplificazione della manutenzione del veicolo e nell'intervento tempestivo in caso di furto.

2. *Su quali aspetti della sicurezza interessa focalizzare l'attenzione?*

   L'interesse prioritario è quello di conoscere in tempo reale la posizione del veicolo e in caso di furto spegnerlo o bloccarlo.

3. *Quali sono le principali funzionalità del software?*

   L'utente finale, attraverso il sistema, può ricevere informazioni sullo stato sia della manutenzione, che della sicurezza del veicolo. Può quindi in caso di necessità interagire con esso.

4. *Quali sono i requisiti generali che questo software deve soddisfare?*

   Il software deve permettere, previa l'autenticazione dell'utente al sistema, una completa gestione del veicolo in remoto. Nonché la possibilità di accedere a una base di dati contenente tutte le informazione relative alla cronologia degli stati del veicolo.

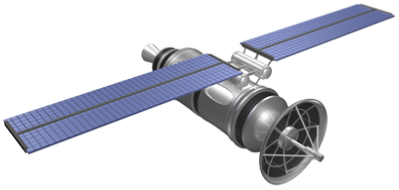
Requisiti funzionali

- Il sistema viene creato al fine di ottimizzare la manutenzione e la sicurezza del veicolo.
- Il veicolo può segnalare in tempo reale i propri parametri.
- L'utente finale può accedere tramite log-in al direttamente veicolo oppure alla cronologia delle posizioni e degli stati di esso.
- L'utente ha la possibilità di scegliere la frequenza con cui ricevere gli aggiornamenti dalla vettura.
- L'utente può visualizzare la posizione direttamente sulla mappa.
- Nel caso in cui non sia possibile connettersi al veicolo l'utente può accedere ugualmente alla cronologia.
- In caso di furto l'utente può spegnere o bloccare il veicolo da remoto.

Requisiti non funzionali

- L'utente può inviare messaggi personalizzati al veicolo, il quale li visualizza tempestivamente sul display del cruscotto.

Schema funzionale



Server SQL

Modulo GPS

AutoServer

Modulo GSM

INTERNET

Centralina

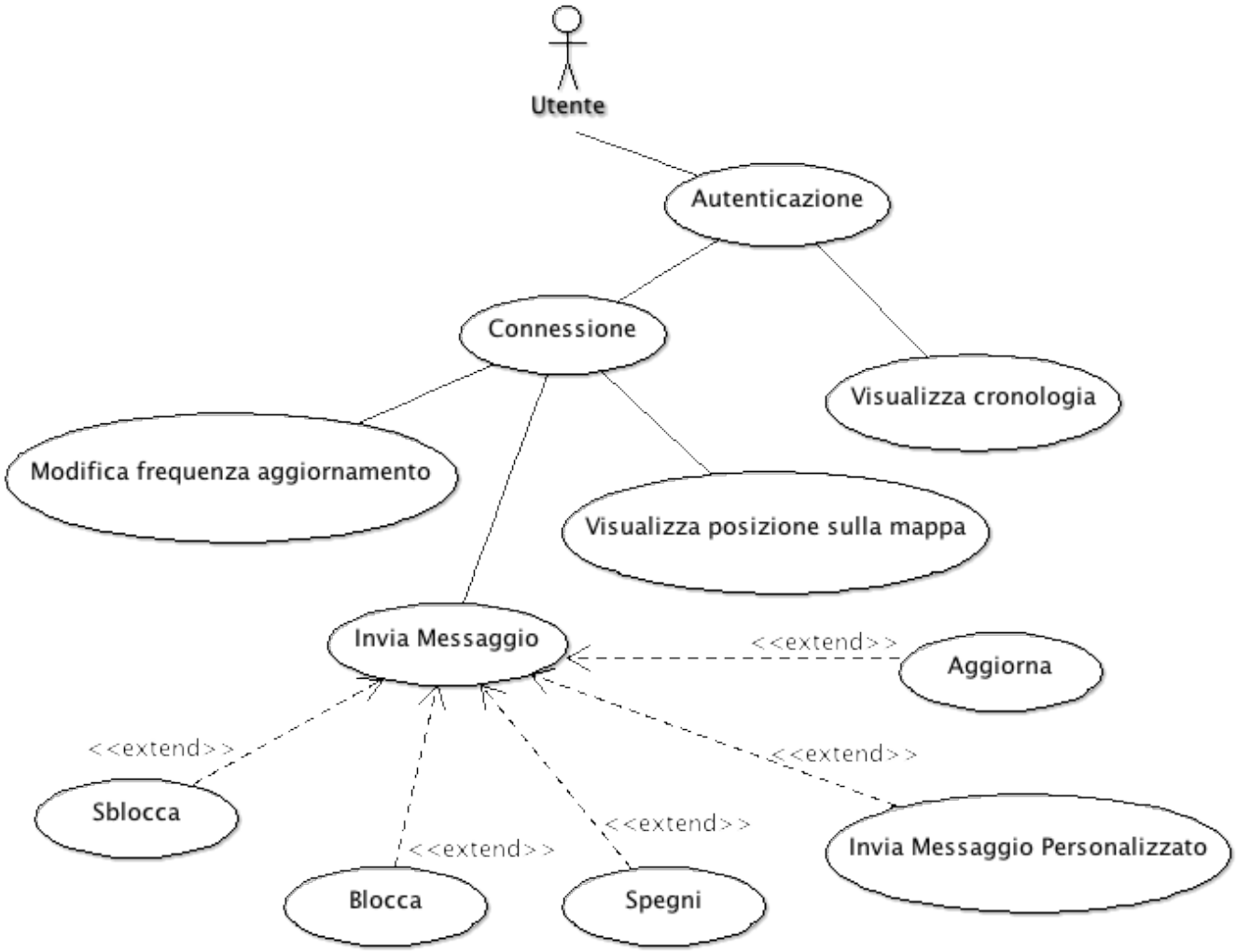
Automobile

Client

Client

USE CASE DIAGRAM

Un “use case diagram” descrive l’interazione tra un utente e il veicolo e serve per comprendere il comportamento di quest’ultimo.
Le interviste con gli utenti portano alla definizione di use case ad alto livello e dell’attore del sistema (Utente).



DESCRIZIONE DEI CASI D'USO

Vengono descritti, di seguito, i principali casi d'uso del sistema. L'analisi degli stessi è stata effettuata utilizzando un approccio di tipo top-down: ciascun caso d'uso viene dettagliato attraverso l'utilizzo di diagrammi delle attività, utili alla rappresentazione di sistemi caratterizzati da un elevato numero di diagrammi di flusso.

CASO D'USO 1: MODIFICA PREFERENZE
Attore: Utente
Pre-Condizione: il sistema visualizza il form di log-in per consentire all'utente l'accesso.
Flusso Principale: Viene visualizzata la finestra di modifica delle preferenze. Il sistema prova a leggere il file "preferenze.xml". Se il sistema riesce a leggere i parametri, li visualizza ognuno nella propria casella e permette all'utente la modifica.
Flusso Alternativo: Nel caso il file non esista o si generi un errore di lettura verrà visualizzato un messaggio a schermo.
Post-Condizione: Nessuna.

CASO D'USO 2: AUTENTICAZIONE
Attore: Utente
Pre-Condizione: il sistema visualizza il form di log-in per consentire all'utente l'accesso.
Flusso Principale: Attraverso questa funzione, un utente precedentemente registrato, può autenticarsi al sistema. Una volta effettuato l'accesso , in base alle credenziali fornite, il sistema consente di svolgere determinate funzioni. Questo caso d'uso prevede la visualizzazione di un form, nel quale l'utente deve inserire la propria "username" e "password". Acquisite le credenziali dell'utente, il sistema effettua legge dal file "preferenze.xml" l'indirizzo ip e la porta del server SQL remoto ed effettua un controllo, per verificarne la validità. La convalida dell'utente da parte del sistema attesta l'avvenuta autenticazione e consente allo stesso l'accesso alla finestra di controllo dell'automobile.
Flusso Alternativo: Nel caso in cui l'utente nel tentativo di accedere alla finestra di controllo inserisce credenziali errate, il sistema comunica l'errore tramite la console e permette all'utente ad inserire nuovamente le proprie credenziali.
Post-Condizione: L'autenticazione dell'utente da parte del sistema consente allo di accedere alla finestra di controllo.

CASO D'USO 3: VISUALIZZA CRONOLOGIA
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato il log-in.
Flusso Principale: Viene visualizzata una finestra dove l'utente potrà vedere l'ultima posizione nota dell'auto, la data e la posizione dell'ultimi tre eventi (Avvio motore, blocco, allarme)
Flusso Alternativo: Nessuno
Post-Condizione: A conclusione dell'operazione, l'utente viene rimandato nella finestra di controllo.

CASO D'USO 4: CONNESSIONE
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato il log-in.
Flusso Principale: Viene letto il file "preferenze.xml" e viene effettuato un tentativo di connessione sull'indirizzo e sulla porta precedentemente salvati sul file.
Flusso Alternativo: Il sistema scrive in console l'errore generato.
Post-Condizione: A connessione avvenuta si potrà notare che l'indicatore di connessione passa da "Offline" a "Online", viene quindi richiesto immediatamente un aggiornamento dei parametri dell'automobile.

CASO D'USO 5: MODIFICA FREQUENZA AGGIORNAMENTO
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato la connessione.
Flusso Principale: Il sistema, di default, effettua una richiesta di aggiornamento ogni 5 secondi, l'utente può cambiare questo valore dal menù a tendina presente sulla finestra di controllo.
Flusso Alternativo: Nessuno.
Post-Condizione: A conclusione dell'operazione e dopo il successivo aggiornamento, verrà modificata la frequenza di richiesta degli aggiornamenti.

CASO D'USO 6: VISUALIZZA POSIZIONE SULLA MAPPA
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato la connessione e bisogna aver ricevuto almeno un aggiornamento.
Flusso Principale: Il sistema aprirà una pagina nel browser predefinito visualizzando la posizione corrente dell'automobile.
Flusso Alternativo: Nessuno
Post-Condizione: A conclusione dell'operazione, l'utente viene rimandato nella finestra di controllo.

CASO D'USO 7: INVIA MESSAGGIO
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato la connessione.
Flusso Principale: il sistema invia un messaggio all'automobile.
Flusso Alternativo: Il sistema scrive in console l'errore generato.
Post-Condizione: Nessuna.

CASO D'USO 8: BLOCCA
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato la connessione.
Flusso Principale: il sistema invia un messaggio all'automobile dove richiede il blocco della stessa.
Flusso Alternativo: Il sistema scrive in console l'errore generato.
Post-Condizione: L'automobile viene bloccata.

CASO D'USO 9: SBLOCCA
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato la connessione.
Flusso Principale: il sistema invia un messaggio all'automobile dove richiede lo sblocco della stessa.
Flusso Alternativo: Il sistema scrive in console l'errore generato.
Post-Condizione: L'automobile viene sbloccata.

CASO D'USO 10: SPEGNI
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato la connessione.
Flusso Principale: il sistema invia un messaggio all'automobile dove richiede lo spegnimento della stessa.
Flusso Alternativo: Il sistema scrive in console l'errore generato.
Post-Condizione: Il motore dell'auto viene spento.

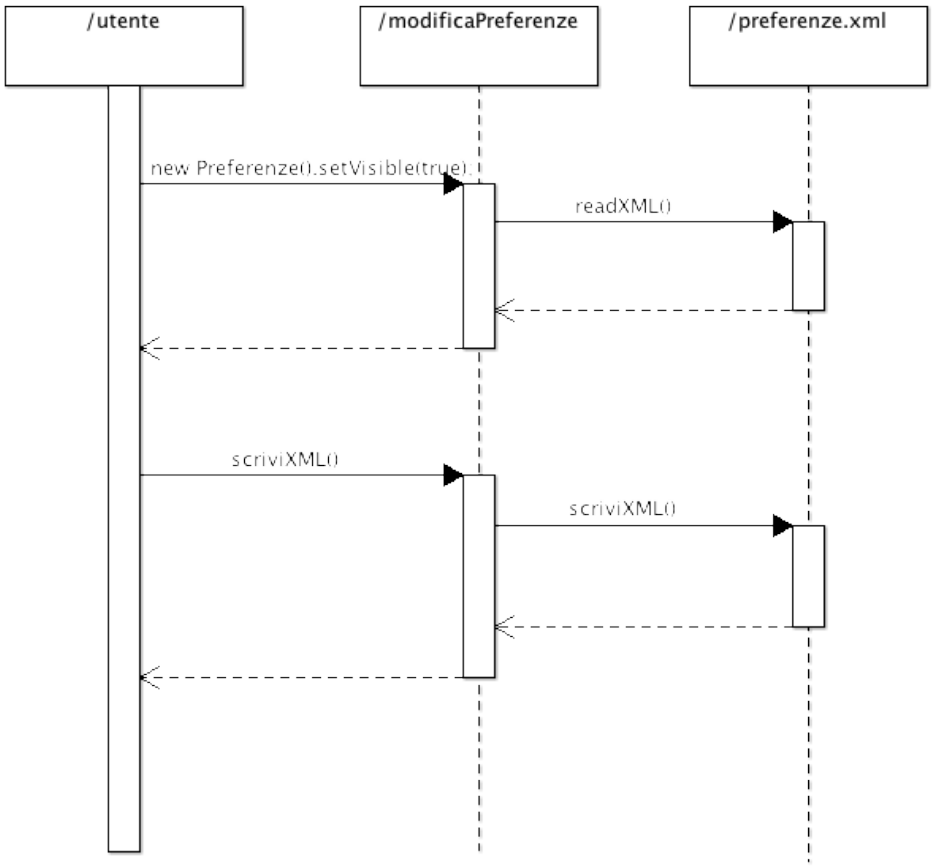
CASO D'USO 11: INVIA MESSAGGIO PERSONALIZZATO
Attore: Utente
Pre-Condizione: L'utente deve avere effettuato la connessione.
Flusso Principale: il sistema visualizza una finestra dove l'utente potrà digitare il proprio messaggio personalizzato.
Flusso Alternativo: Il sistema scrive in console l'errore generato.
Post-Condizione: L'automobile visualizza tempestivamente sul display il messaggio ricevuto.

SEQUENCE DIAGRAM

Un “sequence diagram” descrive uno scenario d’uso mettendo in evidenza la sequenza delle azioni intraprese e le relazioni che intercorrono tra le varie entità del sistema.
Gli elementi principali sono le lifeline e i messaggi. Una lifeline descrive un periodo di vita di un elemento del modello e viene rappresentato tramite un riquadro che riporta un identificativo dell’elemento stesso.
I messaggi sono rappresentati attraverso delle frecce che collegano le varie lifeline.

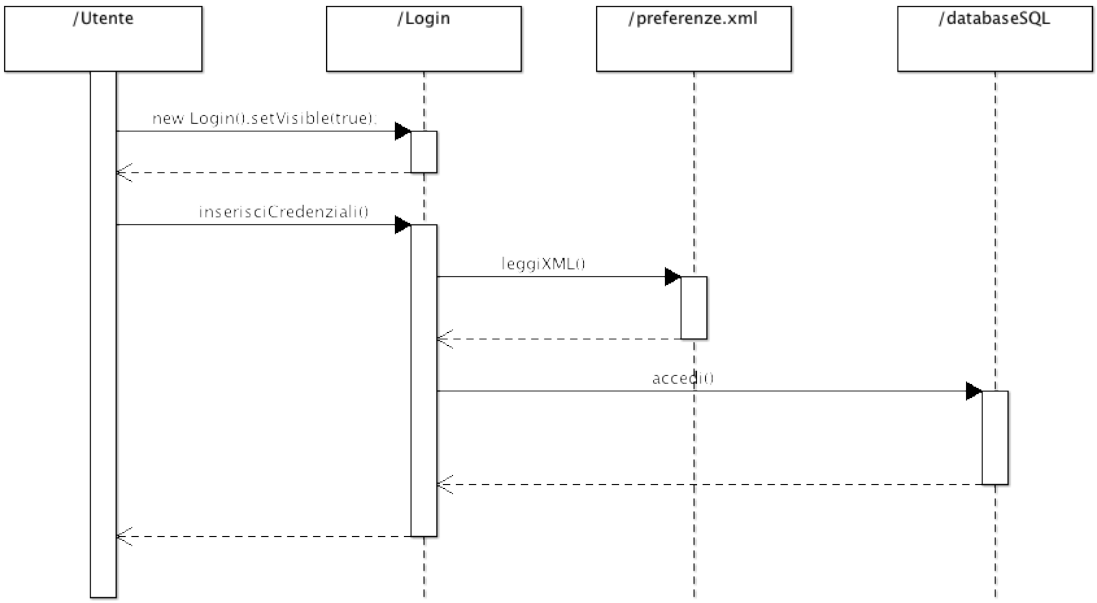
Modifica Preferenze

La corretta modifica delle preferenze (ip, username, password, porte) è essenziale per il corretto funzionamento del software. In questa fase l’utente può visualizzare le attuali preferenze e modificarle se necessario.



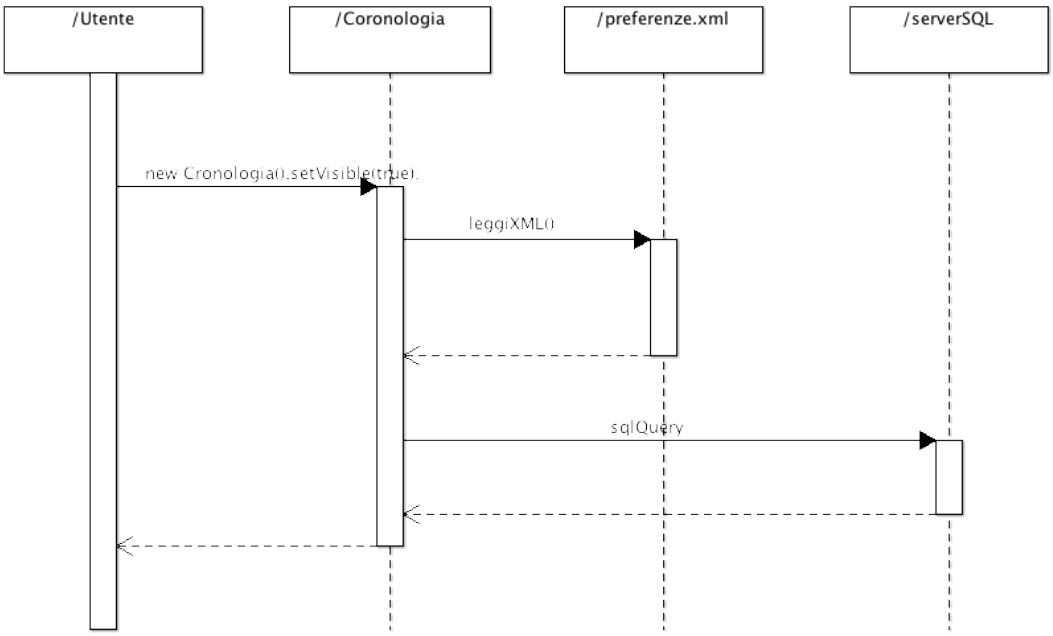
Autenticazione

Viene di seguito illustrato lo scenario di autenticazione da parte di un utente.



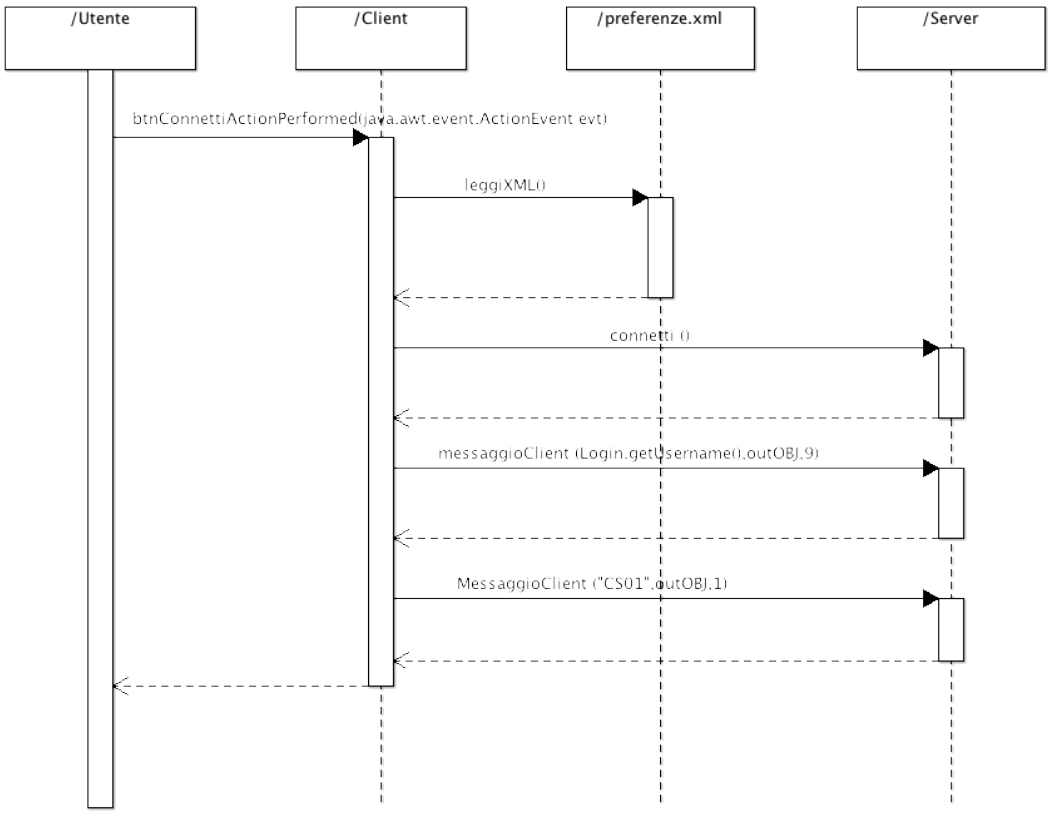
Visualizza Cronologia

La consultazione della cronologia permette all'utente di conoscere lo storico delle posizione e degli stati dell'automobile. In questa fase l'utente può visualizzare tutte le informazioni tramite una query eseguita dal software.



Connessione

Viene di seguito illustrato lo scenario di connessione del client al server.
Ricevuto il comando dall'utente il client procede leggendo i parametri di connessione ai server dell'automobile e effettua un tentativo di connessione.



Modifica frequenza aggiornamento

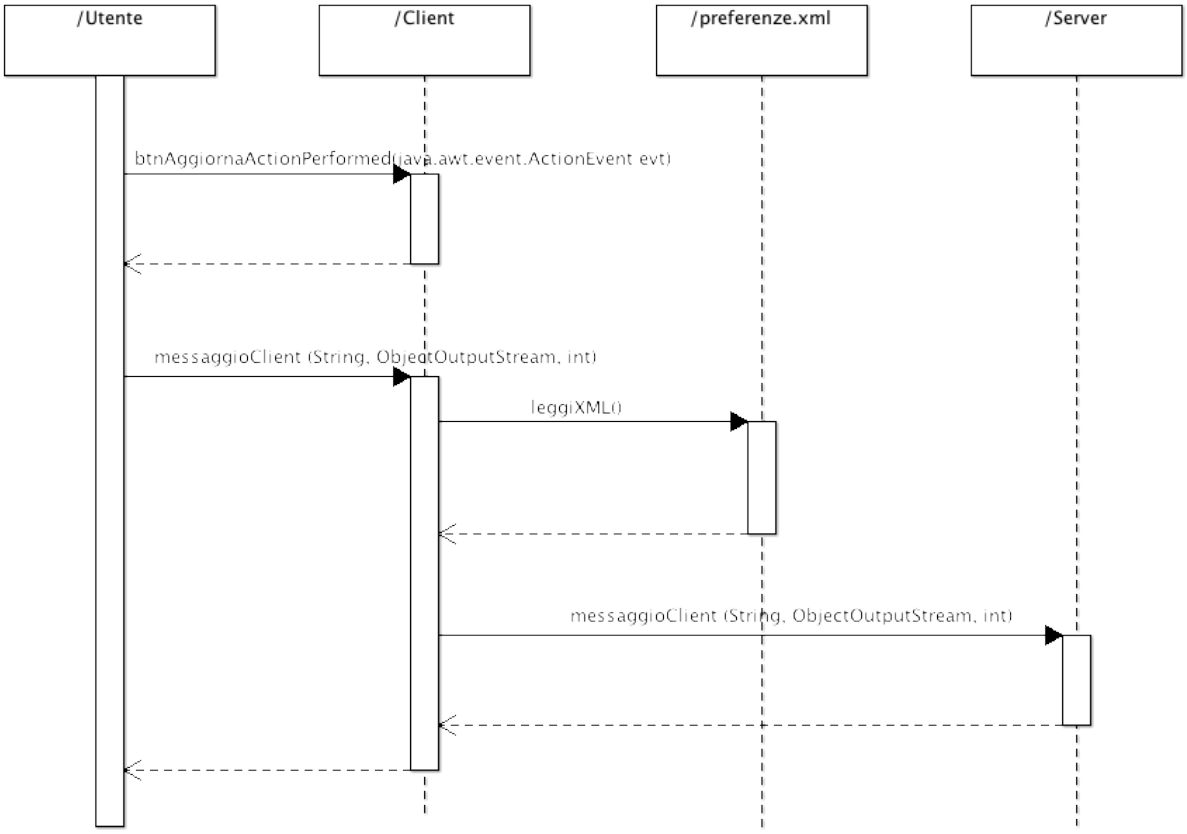
La modifica della frequenza permetta all'utente di sceglie con quale frequenza ricevere gli aggiornamenti.

Visualizza posizione sulla mappa

L'utente, tramite questa funzione, può visualizzare in tempo reale la posizione dell'automobile. Viene di seguito illustrato lo scenario di visualizzazione della posizione dell'automobile.

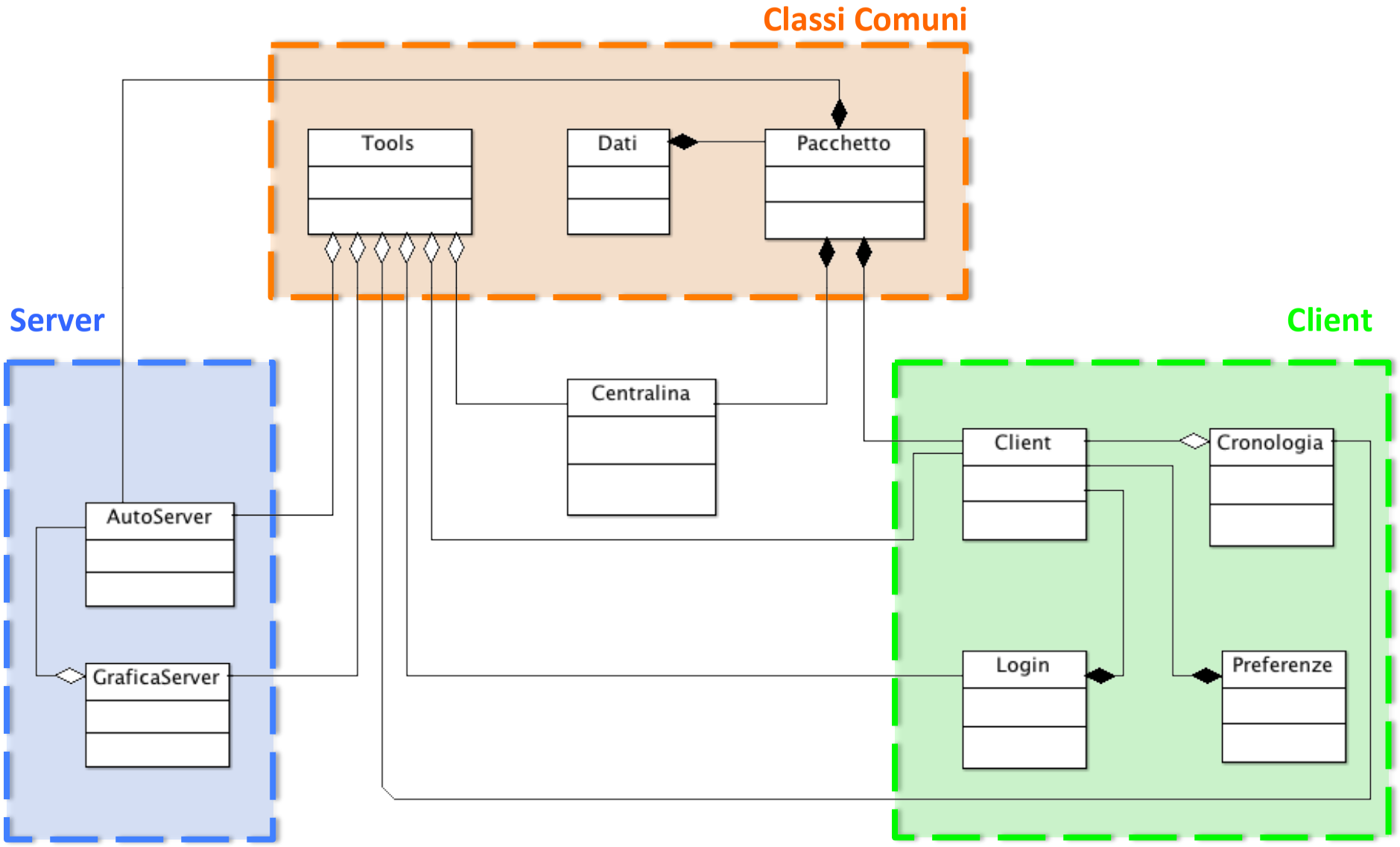
Invio messaggio

L'invio di messaggi permette la comunicazione di informazioni tra client e server e permette inoltre all'utente di inviare un messaggio personalizzato che verrà visualizzato sul display dell'automobile.



CLASS DIAGRAM 1/2

Un “class diagram” descrive le classi e le interfacce che vengono usate per stanziare gli oggetti del sistema e rappresenta il diagramma maggiormente usato dagli attori. L’obiettivo di questo diagramma si concretizza nel definire la visione statica del sistema. Un class diagram è composto da classi di oggetti e dalle loro relazioni.



CLASS DIAGRAM 2/2

Classi Comuni

Pacchetto

- tipo :int
- oggetto :Object

+ getTipo() :int
+ setTipo (int Tipo) :void
+ getOggetto() :Object
+ setOggetto (Object obj) :void

Tools

+ scrivi(): void
+ colora(): void
+ getData(): String
+ boolToInt(boolean valore): int
+ stringaNumero(String stringa) :boolean

Dati

- accensione :boolean
- allarme :boolean
- blocco :boolean
- kmTotali :int
- carburante :int
- gomme :int
- olio :int
- batteria :int
- latitudine :String
- longitudine :String

+ getAccensione() :boolean
+ setAccensione(boolean accensione) :void
+ getAllarme(): boolean
+ setAllarme(boolean allarme) :void
+ getBlocco() :boolean
+ setBlocco(boolean blocco) :void
+ getKmTotali() :int
+ setKmTotali (int kmTotali) :void
+ getCarburante() :int
+ setCarburante (int carburante) :void
+ getGomme() :int
+ setGomme (int gomme) :void
+ getOlio(): int
+ setOlio (int olio) :void
+ getBatteria() :int
+ setBatteria (int batteria) :void
+ getLatitudine() :String
+ setLatitudine (String latitudine) :void
+ getLongitudine() :String
+ setLongitudine (String longitudine) :void

Centralina

- socket :Socket
- inOBJ :ObjectInputStream
- outOBJ :ObjectOutputStream

- reset () :void
- connetti () :void
- messaggioClient (String mess, ObjectOutputStream out, int tipo) :void
- oggettoClient (ObjectOutputStream out) :void
- messaggioDisconnetti (ObjectOutputStream out) :void

Server

GraficaServer

- cont :boolean

+ getIp () :String
+ getUsername () :String
+ getPassword () :String
+ setConsole (String testo) :void
+ setDisplay (String testo) :void
- reset () :void

AutoServer

- accensione :boolean
- allarme :boolean
- blocco :boolean
- kmTotali :int
- carburante :int
- gomme :int
- olio :int
- batteria :int
- latitudine :String
- longitudine :String
- numeroClient :int
- serverSocket :ServerSocket
- socket :Socket

+ getAccensione() :boolean
+ getAllarme(): boolean
+ getBlocco() :boolean
+ getKmTotali() :int
+ getCarburante() :int
+ getGomme() :int
+ getOlio(): int
+ getBatteria() :int
+ getLatitudine() :String
+ getLongitudine() :String
+ getNumeroClient () :int
- connetti () :void
- messaggioServer (String mess, ObjectOutputStream out, int tipo) :void
- oggettoServer (ObjectOutputStream out) :void

Client

Cronologia

- sqlQuery() :void

Login

- username :String
- password :String

+ getUsername () :String
- accedi () :void

Login

- indirizzoSql :String
- usernameSql :String
- passwordSql :String
- indirizzoAuto :String
- portaAuto :String

+ getIndirizzoSql () :String
+ getUsernameSql () :String
+ getPasswordSql () :String
+ getIndirizzoAuto () :String
+ getPortaAuto () :String
- creaXML () :void
- leggiXML () :void

Client

- accensione :boolean
- allarme :boolean
- blocco :boolean
- kmTotali :int
- carburante :int
- gomme :int
- olio :int
- batteria :int
- latitudine :String
- longitudine :String
- socket :Socket
- inOBJ :ObjectInputStream
- outOBJ :ObjectOutputStream
- frequenza :int
- connected :boolean

- reset () :void
- connetti () :void
- messaggioClient (String mess, ObjectOutputStream out, int tipo) :void
- messaggioDisconnetti (ObjectOutputStream out) :void

CODICE SORGENTE

Vengono di seguito riportati le porzioni di codice più importanti del software.

Hosting del server (Classe AutoServer):

```
private void connetti (){
    try{

        insersciCronologiaSQL.start ();
        serverSocket=new ServerSocket(11111);

        System.out.println ("In attesa di un client sulla porta: 6789...");

        grafica.setVisible (true);
        grafica.setConsole ("In attesa di un client sulla porta: 6789...");

        while (true){
            socket=serverSocket.accept();
            Thread connesso = new Connesso(socket);
            connesso.start();
        }

    }catch(Exception e){
        System.out.println (e);
        grafica.setConsole ("Errore nell'hosting del server.");
    }
}

private class Connesso extends Thread{

    private String identità = "";

    private Socket s;
    private Connesso(Socket s) {
        this.s = s;
    }


    @Override
    public void run(){
        Pacchetto ricevuto = new Pacchetto ();
        try{

            grafica.setConsole ("Client connesso su -> 11111.");
            numeroClient ++;

            ObjectInputStream inOBJ;
            ObjectOutputStream outOBJ;


            inOBJ = new ObjectInputStream(s.getInputStream());
            outOBJ = new ObjectOutputStream(s.getOutputStream());

            messaggioServer ("Benvenuto.", outOBJ,1);

            boolean connected = true;
```

```
while (connected){

    ricevuto = (Pacchetto)inOBJ.readObject();

    int tipoPacchetto = ricevuto.getTipo();
    // 0 = Disconnetti
    // 1 = Messaggio
    // 2 = Dati

    // 9 = Identità

    switch (tipoPacchetto) {

        case 0: // Testo

            connected = false;
            grafica.setConsole (identità+": Client disconnesso.");
            numeroClient --;
        break;


        case 1:  // Testo

            String mess = "";

            mess = (String) ricevuto.getOggetto();

            if (mess.equals("CS01")){
                grafica.setConsole (identità+": Richiesta aggiornamento.");
                oggettoServer (outOBJ);

            } else
                if (mess.equals("CS02")){
                    grafica.setConsole (identità+": Richiesta blocco.");
                    blocco = true;

                } else
                    if (mess.equals("CS03")){
                        grafica.setConsole (identità+": Richiesta sblocco.");
                        blocco = false;


                    } else
                        if (mess.equals("CS04")){
                            grafica.setConsole (identità+": Richiesta spegnimento.");
                            accensione = false;

                        } else {
                                grafica.setDisplay (identità+": "+mess);
                            }

            if (allarme){
                messaggioServer ("TENTATIVO DI FURTO", outOBJ,1);
            }

        break;


        case 2:  // Dati


            Dati dati = new Dati ();

            dati = (Dati) ricevuto.getOggetto();
```

```
                    accensione = dati.getAccensione();
                    allarme = dati.getAllarme();
                    blocco = dati.getBlocco();

                    kmTotali = dati.getKmTotali();
                    carburante = dati.getCarburante();
                    gomme = dati.getGomme();
                    olio = dati.getOlio();
                    batteria = dati.getBatteria();

                    latitudine = dati.getLatitudine();
                    longitudine = dati.getLongitudine();

                    grafica.setConsole (identità+": Dati ricevuti.");

                break;


            case 9: // Identità

                identità = (String) ricevuto.getOggetto();

                break;



            }

        }


    } catch(Exception e){
            System.out.println (e);
            grafica.setConsole (identità+": Errore CLIENT.");
        }

    }
}
```

Invio di un messaggio (Classe Client):

```
private void messaggioClient (String mess, ObjectOutputStream out, int tipo) {

    try
    {
        Pacchetto invia = new Pacchetto ();

        invia.setTipo(tipo);
        invia.setOggetto (mess);


        out.writeObject(invia);
        out.flush();
        Tools.scrivi (console,"Messaggio Inviato ("+mess+")");
     } catch(Exception e){
         Tools.scrivi (console,"Errore invio messaggio.");
         System.out.println (e);
     }

}
```

Classe Pacchetto :

```
package utility;

import java.io.Serializable;


public class Pacchetto implements Serializable {


    private int tipo;
    private Object oggetto;

    public Pacchetto () {}

    public int getTipo() {
           return tipo;
        }

    public void setTipo (int Tipo) {
           tipo = Tipo;
        }

    public Object getOggetto() {
           return oggetto;
        }

    public void setOggetto (Object obj) {
           oggetto = obj;
        }


}
```

Classe Dati (1/2):

```
package utility;

import java.io.Serializable;



public class Dati implements Serializable  {

        private boolean accensione;
        private boolean allarme;
        private boolean blocco;

        private int kmTotali;
        private int carburante;
        private int gomme;
        private int olio;
        private int batteria;

        private String latitudine;
        private String longitudine;

        public Dati () {}

        public boolean getAccensione() {
           return accensione;
        }

        public void setAccensione(boolean Accensione) {
           accensione = Accensione;
        }
```

Classe Dati (2/2):

```
    public boolean getAllarme() {
        return allarme;
    }

    public void setAllarme(boolean Allarme) {
        allarme = Allarme;
    }

    public boolean getBlocco() {
        return blocco;
    }

    public void setBlocco(boolean Blocco) {
        blocco = Blocco;
    }

    public int getKmTotali() {
        return kmTotali;
    }

    public void setKmTotali (int KmTotali) {
        kmTotali = KmTotali;
    }

    public int getCarburante() {
        return carburante;
    }

    public void setCarburante (int Carburante) {
        carburante = Carburante;
    }

    public int getGomme() {
        return gomme;
    }

    public void setGomme (int Gomme) {
        gomme = Gomme;
    }

    public int getOlio() {
        return olio;
    }

    public void setOlio (int Olio) {
        olio = Olio;
    }

    public int getBatteria() {
        return batteria;
    }

    public void setBatteria (int Batteria) {
        batteria = Batteria;
    }

    public String getLatitudine() {
        return latitudine;
    }

    public void setLatitudine (String Latitudine) {
        latitudine = Latitudine;
    }

    public String getLongitudine() {
        return longitudine;
    }

    public void setLongitudine (String Longitudine) {
        longitudine = Longitudine;
    }


}
```

Lettura XML (preferenze.xml):

```
    public static void leggiXML() {

        try {

            File xmlFile = new File("Preferenze.xml");

            DocumentBuilderFactory documentFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder documentBuilder = documentFactory.newDocumentBuilder();

            Document preferenze = documentBuilder.parse(xmlFile);

            preferenze.getDocumentElement().normalize();
            NodeList nodeList = preferenze.getElementsByTagName("Preferenze");


             Node node = nodeList.item(0);


                        Element Preferenze = (Element) node;


                        indirizzoSql = Preferenze.getElementsByTagName("Indirizzo").item(0).getTextContent();

                        usernameSql = Preferenze.getElementsByTagName("Username").item(0).getTextContent();

                        passwordSql = Preferenze.getElementsByTagName("Password").item(0).getTextContent();

                        indirizzoAuto = Preferenze.getElementsByTagName("Indirizzo").item(1).getTextContent();

                        portaAuto = Preferenze.getElementsByTagName("PortaAuto").item(0).getTextContent();



            System.out.println("FILE XML LETTO CON SUCCESSO");
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null,"Errore nella lettura delle preferenze");
            System.out.println(e);
        }
    }

}
```

sqlQuery (Classe Cronologia):

```
private void sqlQuery (){

        Preferenze.LeggiXML();
    try {

            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.getConnection("jdbc:mysql://"+Preferenze.getIndirizzoSql()+"/AutoMonitoring?" +
                    "user="+Preferenze.getUsernameSql()+"&"+String.valueOf(Preferenze.getPasswordSql())+"=");
            Statement stmt;
            ResultSet rs;

            stmt = conn.createStatement();

            rs = stmt.executeQuery("SELECT * "
                    + "FROM Cronologia "
                    + "WHERE data = (SELECT MAX(data) FROM Cronologia)");

            while(rs.next())
            {

                lblCronologia.setText ("Cronologia posizioni automobile "
                        + "(Aggiornata al: "+rs.getString("data")+").");

                lblPosizione.setText ("Ultima posizione nota: "+rs.getString("latitudine")+"  /  "+rs.getString("longitudine"

            }

            ////////////////////////////////////////////////////////////////////////////////////////////////////

            rs = stmt.executeQuery("SELECT * "
                    + "FROM Cronologia "
                    + "WHERE data = (SELECT MAX(data) FROM Cronologia AS C WHERE C.avvioMotore = 1)");


            while(rs.next())
            {

                lblDataV1.setText(rs.getString("data"));
                lblLongitudineV1.setText(rs.getString("latitudine"));
                lblLatitudineV1.setText(rs.getString("longitudine"));

            }

            ////////////////////////////////////////////////////////////////////////////////////////////////////

            rs = stmt.executeQuery("SELECT * "
                    + "FROM Cronologia "
                    + "WHERE data = (SELECT MAX(data) FROM Cronologia AS C WHERE C.blocco = 1)");


            while(rs.next())
            {

                lblDataV2.setText(rs.getString("data"));
                lblLongitudineV2.setText(rs.getString("latitudine"));
                lblLatitudineV2.setText(rs.getString("longitudine"));

            }

            ////////////////////////////////////////////////////////////////////////////////////////////////////

            rs = stmt.executeQuery("SELECT * "
                    + "FROM Cronologia "
                    + "WHERE data = (SELECT MAX(data) "
                    + "FROM Cronologia AS C WHERE C.allarme = 1)");


            while(rs.next())
            {

                lblDataV3.setText(rs.getString("data"));
                lblLongitudineV3.setText(rs.getString("latitudine"));
                lblLatitudineV3.setText(rs.getString("longitudine"));

            }

            stmt.close(); // rilascio le risorse
            conn.close(); // termino la connessione
        }
        catch(ClassNotFoundException e)
        {
            System.out.println(e);
        }
        catch(SQLException e)
        {
            System.out.println(e);
            //Tools.scrivi (console,"Errore: "+e);
        }

}
```