Green Move Dynamic Applications

Gianpaolo Cugola, Angelo Morzenti, Matteo Rossi and Edoardo G. Vannutelli Depoli

Abstract In this chapter, we describe the middleware that has been defined and implemented in the prototype of the Green Move system to dynamically manage value-added applications that run on the Green e-Boxes of vehicles and that are used to tailor the user experience of Green Move customers. A Green Move dynamic Application (GMA) is a bundle of code that can be installed or removed at run-time (i.e., after the system has been deployed, even while the vehicle is in use), depending on the current situation and on the user preferences. GMAs can be developed by the administrators of the Green Move system, but also by third parties. In this chapter, we describe the primitives offered by the GMA framework to facilitate the development of GMAs. We also show how the system allows GMAs to be installed automatically, when certain conditions are met, thanks to the complex event processing capabilities of the Green Move system prototype.

This chapter includes material previously published in (Bianchessi et al. 2013, 2014).

This work was carried out while this author was at Department of Electronics Information and Bioengineering, Politecnico di Milano.

G. Cugola · A. Morzenti · M. Rossi (✉) · E.G. Vannutelli Depoli
Department of Electronics Information and Bioengineering,
Politecnico Di Milano, Piazza Leonardo Da Vinci 32, 20133 Milan, Italy
e-mail: matteo.rossi@polimi.it

G. Cugola
e-mail: gianpaolo.cugola@polimi.it

A. Morzenti
e-mail: angelo.morzenti@polimi.it

E.G. Vannutelli Depoli
e-mail: edoardo.vannutelli@polimi.it

© Springer International Publishing AG 2017
D.F. Bignami et al. (eds.), *Electric Vehicle Sharing Services for Smarter Cities*,
Research for Development, DOI 10.1007/978-3-319-61964-4_9

1 Introduction

The architecture of the Green Move system described in Chap. 8, which includes the Green Move Center (GMC), the Green e-Boxes (GEBs) installed on vehicles, and the Green Move app installed on users' mobile devices has been designed to support a high level of flexibility and configurability of the services offered to users.

In this chapter, we focus on the mechanisms developed to tailor the functions offered by a vehicle to the preferences of its current user. This is achieved through the notion of *dynamic application*, i.e., a piece of software that can be loaded on the GEB of a vehicle at any time, whether it is in use or not, and which extends the functions offered to the user. Dynamic applications can access the data that are present on the vehicle—speed, acceleration, state of charge of the battery, etc.—to create value-added services for the user.

We present a pair of prototype dynamic applications. The first one addresses the issue of *range anxiety*—that is, the fear that the battery will empty before reaching the destination—by providing feedback to the user concerning his driving style, to nudge him toward a more economical one. The second application provides context-dependent advice to the user, such as commercial or cultural suggestions; this application can also be used to coordinate the fleet, for example, by notifying drivers of points of interest that are in their proximity—e.g., available charging stations. The Green Move platform allows these applications to be dynamically installed on vehicles, while they are in use, depending on the situation they are in. For example, the driving style application can be loaded only for users who have indicated, among their preferences, the desire to receive that kind of feedback; or only when the system determines that the user is exhausting the battery while still far from his destination, so a more economic style of driving is necessary to maximize the driving range. Dynamic applications also support a scenario where multiple implementations are available for a given functionality—for example, with different styles for presenting feedback to the driver—and the one that best fits the user preferences is loaded at the beginning of the trip and unloaded at the end. In addition, a mechanism where applications are installed dynamically after the fleet has been deployed allows administrators to remotely perform updates of the functions offered by the GEBs without physically operating on them.

In the next section, we briefly survey existing solutions that enable the execution of user-oriented software applications on vehicles (especially cars) and highlight their differences with respect to the mechanisms developed for the Green Move platform, in particular with respect to the issues of configurability and interactions of users and vehicles. Then, we describe the mechanisms underlying Green Move dynamic applications and present a pair of prototypical ones.

2 Related Works

In recent years, more and more vehicles—especially cars—have been equipped with devices capable of running applications that enhance the user experience during the drive. In particular, most Original Equipment Manufacturers (OEMs) produce systems for In-Vehicle Infotainment (IVI) and telemetry systems for the monitoring of the state of the vehicle. Most telemetry systems are proprietary based on real-time operating systems; they are typically not integrated with IVI. IVI systems are also usually proprietary, though lately a few OEMs are converging on common solutions.

Mobile solutions for hardware platforms and operating systems are the most commonly used basis for building IVI systems The iOS-based *CarPlay*[1] IVI by Apple offers the possibility to install third-party applications—though not dynamically—and to interact with the driver's phone to share the data and audio channels; at the moment it is not clear if it offers any access to vehicle data.

General Motors have developed a proprietary solution,[2] which allows for a closer integration with the vehicle. In particular, it exposes two sets of application programming interfaces (APIs): In-Vehicle APIs and Remote APIs. In-Vehicle APIs allow developers to access a small set of vehicle data, audio/video capabilities, navigation information, user interfaces, and communication channels. Remote APIs offer access to the vehicle data and the ability to send commands to the vehicle. Through these commands, one can lock/unlock the vehicle doors, retrieve diagnostic information, or retrieve the vehicle's position. This system offers functions similar to those that have been developed for the Green Move platform; however, unlike the latter, it does not seem to offer any capability of dynamically changing the services offered onboard.

The *GENIVI Alliance*[3] has developed an open-source infrastructure for in-vehicle infotainment. The *Tizen*[4] software platform, originally born as a mobile operating system, is now shifting its focus to the IVI market through the GENIVI Alliance open-source platform.

All the solutions above focus on creating an operating system that offers access to multimedia and network functions, and in some cases to vehicle data. Applications developed for these platforms have to be manually installed and started by the user.

OSGi[5] is a service-oriented component-based framework that allows developers to create and manage dynamically loadable applications. The OSGi infrastructure is built upon three basic abstractions: modules, life cycle management, and services. A *module* is a single portion of functional code, wrapped in a deployable unit called

[1] www.apple.com/ios/carplay.

[2] developer.gm.com.

[3] www.genivi.org.

[4] www.tizen.org.

[5] www.osgi.org.

bundle. The *OSGiContainer* provides the bundles' execution environment and primitives to manage their life cycle. Bundles can be dynamically downloaded, installed, and started. *Services* are built upon modules. Every module can offer and consume services. OSGiContainer provides standard modules to manage security issues. Knopflerfish[6] and Apache Felix[7] are two available implementations of the OSGi framework. The OSGi architecture targets generic Java virtual machines rather than Android's Dalvik, although Android implementations are available. The Green Move mechanisms for managing dynamic applications, instead, have been specifically designed for vehicle sharing systems: They allow for the installation of new components without driver intervention and for the access to vehicle data. In addition, the Green Move APIs for programming dynamic applications are simple and lightweight, but nevertheless they fit different kinds of applications, as shown later in this chapter.

To summarize, finding new solutions for an optimal management of electric, flexible, and heterogeneous fleets is one of the main challenges facing today's urban mobility. The Green Move project has tackled this issue through a platform that allows both a high level of automation in the interaction between users and system, and powerful customizations of onboard services through the notion of dynamic applications.

3 Overview

The Green Move system allows administrators to modify the functions offered by GEBs installed on vehicles on-the-fly, at any time, while the system is running. This is achieved by dynamically loading and unloading applications that run on GEBs. These applications on one side are built on top of the basic functions offered by GEBs that are described in Chap. 8, and on the other side extend the services offered by GEBs, in particular to users.

Consider, for example, the following scenario, typical for electric vehicles, which need a long time to recharge. The user, by driving aggressively in town, is quickly consuming the reserve of energy stored in the battery and runs the risk of not being able to reach his destination; the system, which is monitoring the status of the vehicle, and in particular the battery charge, through the mechanisms described in Chap. 8, notices the depletion of the battery charge, and decides to aid the user by suggesting a kinder driving style, one which can make the battery last longer; then, to help the user in this regard, the system dynamically installs an application which, by elaborating the data retrieved by the GEB concerning speed, acceleration, etc., of the vehicle—i.e., parameters that affect the rate with which the battery is

[6]www.knopflerfish.org.
[7]felix.apache.org.

depleted—presents the user with a measure of the aggressiveness of his driving style, so as to steer him toward a gentler style, less demanding on the battery.

As a second scenario, consider an organization— for example, Politecnico di Milano—whose employees use the vehicle sharing system when they have to move within the city for institutional reasons. Imagine that the organization and the management of the vehicle sharing system have an agreement that grants the former some privileges such as preferred access to the vehicles during certain days and time slots, or the possibility to have customized functions available for its employees. Then, when an employee of the institution—say, a courier—takes one of the vehicles to visit various places in the city, the system automatically loads on the vehicle an application for managing not only the route followed by the employee (which might be optimized according to the list of tasks to be carried out), but also the tasks themselves (e.g., to keep track of the deliveries and the pickups).

In the second scenario, the tracking application must be available only for a subset of the users of the system; in addition, it might not be available all the time, as agreements between institutions can change or expire. Hence, it is crucial to have a flexible mechanism that allows the system to dynamically configure, in real-time and on-demand—i.e., at irregular, unscheduled moments in time—the services offered to users by the system. Even in the first scenario, which in some cases could be served by an application preinstalled on all vehicles and activated only when necessary, a dynamic mechanism for the loading/unloading of applications on GEBs would be very useful, as different "driving style coaches" might be available to users, and the one to be actually installed could be a real-time decision made based on the preferences of the user.

The Green Move platform supports the scenarios described above through the notion of Green Move dynamic Applications (GMAs for short), which can be installed on and removed from GEBs while the system is running. Figure 1 shows the steps through which a GMA is uploaded on a GEB. The mechanisms for the management of GMAs leverage the architecture and the technologies that are at the core of the Green Move system, and which are described in Chap. 8. First (Fig. 1a) a T-Rex event is sent to the GEB, which notifies the latter that a GMA is to be installed; the event contains the address on the GMC where the GMA can be retrieved. Then, the app running on the GEB accesses the GMC (Fig. 1b) and retrieves the code of the GMA (Fig. 1c). Finally, the interfaces implemented by the GMA allow the GEB to instantiate the application and start it (Fig. 1d). Similar mechanisms are used to uninstall a GMA from a GEB, or to reload a running GMA.

The GMC gives the administrators different options to select the vehicles on which a GMA is to be uploaded: single vehicles (unicast), groups of vehicles (multicast) or the whole fleet (broadcast).

The next chapter details the main mechanisms and components of the middleware that supports the distribution and management of GMAs.

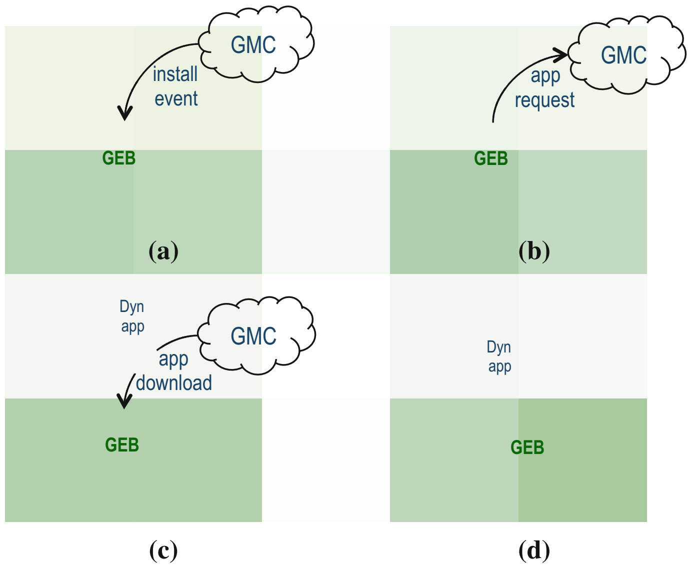

Fig. 1 Overview of the steps for the installation of a dynamic application

4 Green Move Dynamic Applications Middleware

The middleware for managing GMAs is based on two main components: the *Code Server*, which resides on the GMC and the *GMcontainer*, which instead is part of the GEB. In the following, we introduce their main features.

4.1 Code Server

The Code Server is a module of the GMC that allows trusted parties to upload their applications, to verify them and to distribute them to GEBs through the Green Move middleware. It also allows administrators to get the current list of devices running a certain application, to stop any running instance and uninstall it, or to deploy it a second time. Applications are uploaded as signed JAR files, which are verified for authenticity before being made distributable.

4.2 GMcontainer

Figure 2 shows the main elements of GMcontainer, the component that is part of the GEB and that is in charge of managing GMAs on vehicles. In particular, the GMcontainer listens to the events sent by the GMC depicted in Fig. 1; it reacts

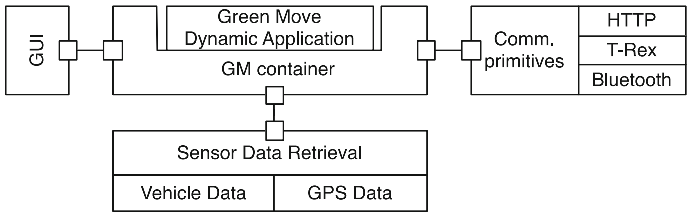

Fig. 2 Schematic view of the Green Move dynamic application container

to them by downloading the GMA code, instantiating it, and removing it when it is no longer needed. In addition, the GMcontainer keeps a persistent registry of the GMAs installed.

The interface implemented by a GMA allows it to retrieve from the GMcontainer references to other components of the Green Move environment running on the GEB. In particular, a GMA can use primitives that allow it to: (i) send messages through a variety of communications channels; (ii) acquire the vehicle data (speed, acceleration, state of charge, etc.) from the sensors; (iii) display information on the graphical user interface (GUI) of the GEB if one is available.

A GMA can communicate with remote components in several different ways. It can use an HTTP channel that is made available and mediated by the GEB. Such a channel can be used to interact with external services to send/retrieve information (e.g., weather, traffic, advertisements) through standard protocols. For security reasons, the HTTP connection is managed on the GEB by a software proxy which has the possibility to check the traffic and, for example, block access to blacklisted servers.

A standard HTTP channel, however, is ill-suited for transmitting (or receiving) streams of data, such as telemetry or composite events. For this reason, the GMcontainer offers GMAs access to a T-Rex channel, through which they can publish or subscribe to events. The T-Rex channel allows a GMA to distribute information to other vehicles, if the latter subscribe to the events published by the former.

A GMA can also use the Bluetooth channel mediated by the GMcontainer (see Fig. 2) to send messages to the smartphone of the user during the rental. This allows a GMA to communicate with the Green Move client application, or another companion application on the user's smartphone, as explained in Sect. 5.2.

The GMcontainer offers GMAs the possibility to interact with the vehicle through a high-level interface. The interface is very general, and it can be used to read the status of the vehicle, such as the state of charge of the battery or the speed of the vehicle. However, as mentioned in Chap. 8, the Green Move platform targets fleets made of heterogeneous vehicles, so different kinds of vehicles might offer different kinds of data; for example, not all vehicles have doors and, for those that do not, a "door status" is not available. Through this high-level interface a GMA can also send commands to the vehicle. For example, "open/close door" commands can be issued if the GMA resides on the GEB of a vehicle that has doors. As before, the sending/receiving of commands is mediated by the GEB, which can monitor the interaction.

If the GEB has a screen (not all do, such as those of scooters), a GMA can use it to provide information to the user. For example, a GMA might display suitable diagrams to give the user feedback on the driving style. The GMcontainer offers GMAs two different possibilities to show information on the GEB GUI: display a short, purely textual message on the main view of the GEB app; or build a full-screen, graphical view that is under the control of the GMA. The availability of different primitives allows GMA programmers to select the one that better fits the needs of the application, for example from the point of view of the energy consumption.

4.3 Implementation

From the implementation point of view, the GMcontainer is realized as an Android Service. GMAs are Android components that adhere to the single entry point convention enforced by the GMcontainer. This means that applications could be coded in any language that can be run on top of the Android Java Virtual Machine (Dalvik), such as Ruby or Python. The current implementation assumes that the application code is sealed in a standard JAR file.

To allow the system to uniformly manage heterogeneous applications, each GMA must implement a standard interface, which exports primitives that are used by the GMcontainer to set-up, start and stop the application. More precisely, the primitive for setting up and starting the GMA is used by GMcontainer to pass the application the reference to the GEB components that it can use during its execution, and which are depicted in Fig. 2. The primitive that is used by GMcontainer for stopping the application, instead, is responsible for the disposal of the GMA's own resources (sockets, running threads, and so on).

As mentioned in Sect. 3, GMAs are installed and uninstalled by sending GEBs suitable T-Rex events (Cugola and Margara 2012). These events can be generated in several ways, possibly as a consequence of other events that occur in the system. In the latter case, they are defined as rules expressed in the TESLA language (Cugola and Margara 2010). For example, the following TESLA rule, upon the release of the vehicle by the user (marked by event *Released* described in Chap. 8), generates an *Uninstall* event for each GMA that has been loaded during the last rental, thus cleaning up newly installed GMAs from the GEB.

```
define Uninstall (String: GreenBox_id, String: class)
from Released(GreenBox_id = $a) and
each SendApp(GreenBox_id = $a) within 5 days from Released
consuming SendApp
where Uninstall.GreenBox_id = SendApp.greenBox_id and
      Uninstall.class = SendApp.class
```

Notice that the *Unistall* event has two fields: GreenBox_id and class, where the latter is a string identifying the main class of the application. SendApp is the event that is used by the Green Move middleware to notify GEBs of the need to load a GMA; it has three fields: GreenBox_id, class, and appUrl, where the first two are as for the Uninstall event and the third one points the GEB to the URL on the GMC from where the GMA is to be downloaded.

The next sections present meaningful examples of prototype GMAs that illustrate the features available to dynamic applications and what benefits can be gained from them.

5 Examples of Green Move Dynamic Applications

In this section, we present a pair of GMAs that have been implemented to show the capabilities of the Green Move middleware. The first one realizes the "driving style coach" scenario introduced in Sect. 3, whereas the second one is an application that provides personalized advice to users (for example, about commercial offers in their proximity) while they are moving around the city. In this chapter, we focus on the mechanisms provided by the Green Move middleware that allowed us to realize the applications. Chap. 11 (for the driving style application) and Chap. 10 (for the personalized advice application) present the algorithms and principles that are at the basis of the two applications.

5.1 The Driving Style Dynamic Application

The driving style GMA helps the user save battery during the trip, by nudging him toward a smoother style of driving. To achieve this, it computes several indexes that provide an estimation of the driving style of the user and communicates them to the GMC; in addition, the application displays the information to the driver, to induce him to keep a driving profile that is conducive to saving energy.

The application takes advantage of the features offered by the Green Move middleware supporting dynamic applications, such as the possibility of accessing vehicle data (e.g., speed and acceleration), the primitives for accessing the GUI on the GEB, and the communication mechanisms—in particular, the publish-subscribe primitives offered by T-Rex—to interact with the GMC. Figure 3 shows a general overview of the driving style GMA and its interactions with both the vehicle and the user.

The GMC can use the information sent by the driving style GMA to provide users with an enhanced service or additional features (e.g., personalized fees, user alerts, better prediction of the vehicle's range, fleet optimization).

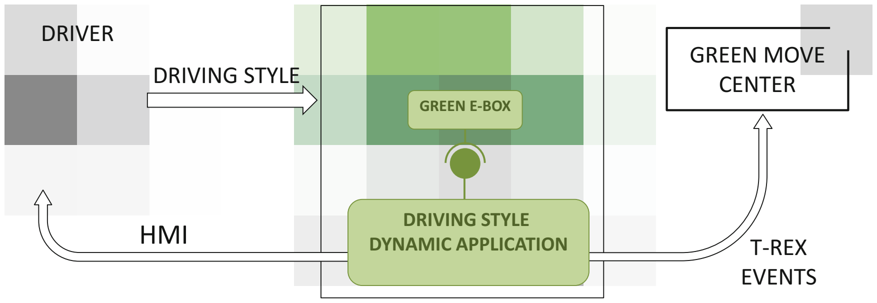

Fig. 3 Overview of the driving style GMA

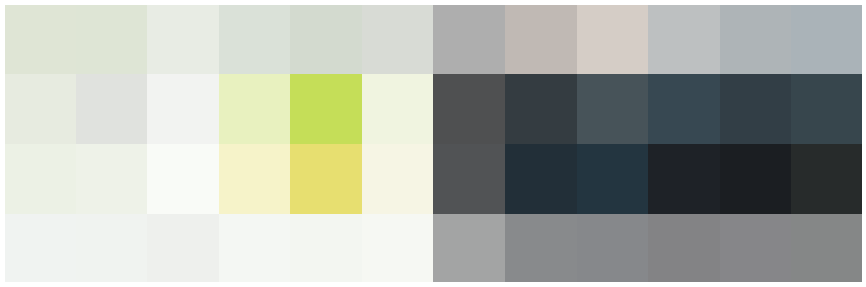

Fig. 4 User interface of the driving style GMA **a** and the application running on a GEB **b**

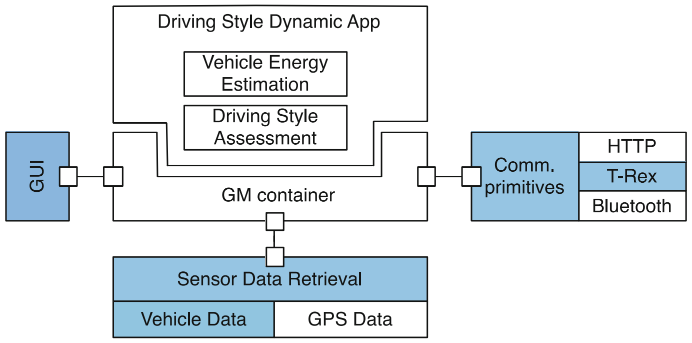

Fig. 5 Features of GMcontainer exploited by the driving style GMA

As mentioned above, the driving style GMA provides feedback to the driver in real-time. To achieve this, the application exploits the primitives provided by GMcontainer to access to the GUI of the GEB. Figure 4 shows its user interface, which provides an immediate and simple representation of the user's driving style.

As shown in Fig. 5, the driving style GMA exploits a number of the features offered by the dynamic application framework, and in particular:

- *the GUI's primitives*, to display the driving style indexes on the GEB;
- *the T-Rex event channel*, to send messages to the GMC;
- *the vehicle data retrieval interface*, to gather data about the vehicle (e.g., speed and acceleration).

According to the scenario outlined in Sect. 3, the application is automatically deployed on a GEB when the user is at risk of not reaching his destination because of an aggressive driving style that too rapidly discharges the battery. This is achieved through the definition of suitable T-Rex events. In particular, the GMC knows the initial state of charge (*SoC*) of the battery, thanks to T-Rex event SocWhenTaken defined by the following TESLA rule:

```
define SocWhenTaken (String: greenBox_id, int: soc)
from Taken(greenBox_id = $a) and
last VehicleStatus(greenBox_id = $a) within 5 days from Taken
where SocWhenTaken.greenBox_id = VehicleStatus.greenBox_id and
      SocWhenTaken.soc = VehicleStatus.soc
```

More precisely, a *SocWhenTaken* event is generated when a rental starts (which corresponds to event *Taken* described in Chap. 8); it carries the information concerning the last value of the SoC that is retrieved from the vehicle.

At regular intervals while the vehicle is moving, the GEB sends *VehicleStatus* events to the GMC (see also Chap. 8); when the residual charge of the battery is less than a given threshold, a *SendApp* event is generated containing the information needed to load the driving style GMA. *SendApp* is a composite event, whose definition in the TESLA language is the following:

```
define SendApp (String: greenBox_id, String: appUrl, String:class)
from VehicleStatus(greenBox_id = $a) and
last SocWhenTaken(greenBox_id = $a) within 1 day
from VehicleStatus and
not Released(greenBox_id = $a) between VehicleStatus and SocWhenTaken
and VehicleStatus.soc < SocWhenTaken.soc*k/(1 + k)
where SendApp.greenBox_id = VehicleStatus.greenBox_id and
      SendApp.appUrl = "<GMAurl>" and SendApp.class = "<GMAclassName>"
```

In particular, the *SendApp* event is generated when the current value of the SoC is less than the SoC consumed since the beginning of the rental multiplied by a constant factor k (which is empirically determined); that is, the event is triggered when constraint $SoC_{\mathrm{curr}} \leq k(SoC_{\mathrm{init}}—SoC_{\mathrm{curr}})$ holds.

5.2 *Dynamic Application for Personalized Advice*

As a second example, we introduce an application, called *GMadvisor*, that provides users with customized, context-dependent advice, while keeping user preferences private from the rest of the Green Move system. The application, whose underlying mechanisms and algorithms are presented in Chap. 10, is an example of GMA that mixes data collected from different sources (users, vehicles, external players) and shows how the Green Move middleware can support distributed architectures involving components other than GMC and GEBs. In fact, as depicted in Fig. 6, the application is based on three components:

- A server storing the set of advice that can be sent to users (e.g., position of charging stations, points of interest).
- The *GMadvisor client app*, which is installed on the mobile device of the user and is responsible for storing the user preferences—necessary for customizing the advice—that should not be shared with the rest of the system; the app also displays advice to the user if the GEB does not have a GUI.
- The *GMadvisor GMA* dynamically deployed on the vehicle's GEB, which sends the id of the user and the position of the vehicle to the advice server, receives the advice and—if it has a GUI—displays those that, according to the client app, match the user preferences.

Similar to the driving style GMA, the GMadvisor GMA exploits various features of the Green Move middleware for dynamic applications, shown in Fig. 7. More precisely:

- It uses the GEB GUI to show suggestions on the device's display, if one is available.
- It communicates with the user's mobile device through the Bluetooth channel, to exchange information that allows it to select the advice to show.

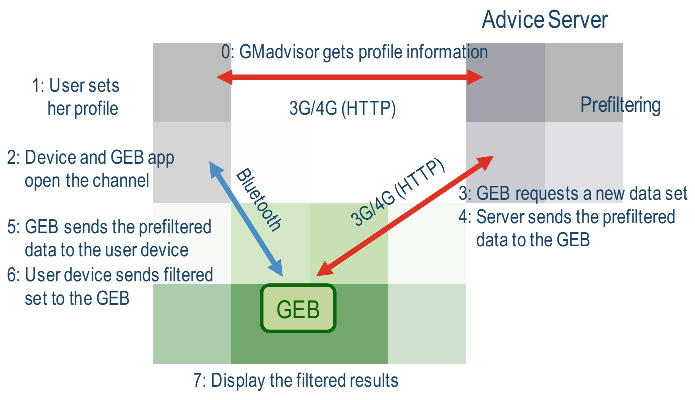
Advice Server
0: GMadvisor gets profile information
1: User sets her profile
3G/4G (HTTP)
Prefiltering
2: Device and GEB app open the channel
Bluetooth
3G/4G (HTTP)
3: GEB requests a new data set
4: Server sends the prefiltered data to the GEB
5: GEB sends the prefiltered data to the user device
6: User device sends filtered set to the GEB
GEB
7: Display the filtered results

Fig. 6 Components of the GMadvisor application and interactions among them

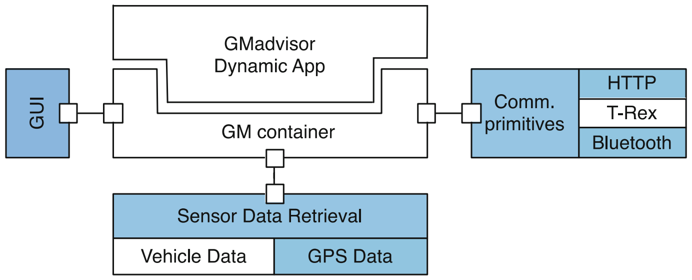

Fig. 7 Features of GMcontainer exploited by the GMadvisor GMA

- It communicates with external servers through the HTTP channel, and in particular it retrieves from an advice server the information to be distributed.
- It retrieves real-time data from the vehicle to determine, for example, the current position of the user.

The GMadvisor GMA can be deployed automatically by the system. This is achieved by generating a suitable T-Rex event upon a *Taken* event triggered by a user starting a rental (if she selected in her profile the preference to receive advice). It can be removed from the GEB through the *Uninstall* event shown in Sect. 4.3.

6 Conclusions

The Green Move platform is highly configurable, as it includes a middleware that allows applications—GMAs—to be dynamically loaded onto vehicles' onboard computers—the GEBs—before and during trips, and then to be removed when they are no longer needed. In this chapter, we have outlined the core features and mechanisms implemented in the Green Move prototype system to support the execution of GMAs. The flexibility offered by the Green Move platform through GMAs allows system managers to tailor the services offered during the vehicle rental to the user's needs and preferences, as witnessed by the prototype applications described in Sect. 5.

Future improvements of the middleware presented in this chapter will focus on the strengthening of the security of the framework in aspects concerning the distribution of GMAs to GEBs and their execution on the target devices, in particular for what relates to the isolation of applications between each other.

References

Bianchessi A, Ongini C, Rotondi S, Tanelli M, Rossi M, Cugola G, Savaresi SM (2013) A flexible architecture for managing vehicle sharing systems. IEEE Embed Syst Lett 5(3):30–33

Bianchessi AG, Cugola G, Formentin S, Morzenti A, Ongini C, Panigati E, Rossi M, Savaresi SM, Schreiber FA, Tanca L, Vannutelli Depoli EG (2014) Green move: a platform for highly configurable, heterogeneous electric Vehicle Sharing. IEEE Intell Transp Syst Mag 6(3):96–108

Cugola G, Margara A (2010) "TESLA: a Formally Defined Event Specification Language". Proceedings of the ACM International Conference On Distributed Event-Based Systems (DEBS)

Cugola G, Margara A (2012) Complex event processing with T-REX. J Syst Softw 85(8): 1709–1728