Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2009, Article ID 856962, 10 pages
doi:10.1155/2009/856962

Research Article

An Embedded Software Platform for Distributed Automotive Environment Management

Ralf Seepold, Natividad Martínez Madrid, Jesús Sáez Gómez-Escalonilla, and Alvaro Reina Nieves

Departamento de Ingeniería Telemática Escuela Politécnica Superior Universidad Carlos III de Madrid, Avenida Universidad 30, 28911 Leganés, Madrid, Spain

Correspondence should be addressed to Ralf Seepold, ralf.seepold@uc3m.es

Received 2 September 2008; Revised 9 December 2008; Accepted 27 January 2009

Recommended by Markus Kucera

This paper discusses an innovative extension of the actual vehicle platforms that integrate intelligent environments in order to carry out e-safety tasks improving the driving security. These platforms are dedicated to automotive environments which are characterized by sensor networks deployed along the vehicles. Since this kind of platform infrastructure is hardly extensible and forms a non-scalable process unit, an embedded OSGi-based UPnP platform extension is proposed in this article. Such extension deploys a compatible and scalable uniform environment that allows to manage the vehicle components heterogeneity and to provide plug and play support, being compatible with all kind of devices and sensors located in a car network. Furthermore, such extension allows to autoregister any kind of external devices, wherever they are located, providing the in-vehicle system with additional services and data supplied by them. This extension also supports service provisioning and connections to external and remote network services using SIP technology.

1. Introduction

Cars are equipped with an increasing number of sensors mainly integrated (and provided) by car manufacturers and thus directly connected to the proprietary internal bus of a vehicle. While the amount of internal sensors is increasing, also external sensors play a growing important role. Examples are wearable sensor networks, specific devices connected to health care monitoring or sensors connected to PDA devices, like GPS trackers. These external sensors can provide additional data to in-vehicle systems that are not available via in-car sensors, like passengers' health status and health files in case of an accident. All these retrieved data can be used by the system developed within the "intelligent vehicles" in order to deploy intelligent environments. They are based on applications that are able to infer risk situation and act to prevent or mitigate them.

The main target of this work is to develop an open in-car service platform that provides an abstraction layer to all devices accessible through the vehicle area network (VAN). Such layer will allow a seamless integration of on-board sensors as well as the services provided by any kind of connectable device located under the vehicle network coverage on the one hand side and the possibility to establish connections to services located in external (remote) networks. It will deploy a uniform execution environment that allows the applications running over it to access in a standard way to the information, and services offered by the connected sensors, devices and servers. In the same way, the developed platform implements the required routines to manage the connectivity toward both the existing vehicle networks (CAN Bus, FlexRay, LIN, etc.) and the external ones (Wifi, UMTS, and GPRS), as well as to act as an access point toward several kinds of local area network deployed within the car environment (Wifi, Bluetooth, Zigbee, etc.). Thus, when the term "platform" is mentioned, it refers to a *car gateway* implemented within a vehicle including all connectivity and abstraction routines.

This car gateway will provide not only the connectivity module, able to connect to all devices inside the car, but

it will establish also secure connections on demand to any external network. As a result, it will be possible to register services located within foreign destinations (remote devices). Additionally, the car gateway can work in a reverse mode registering any application that runs on the gateway to offer its services also to VANs as within an external network. In this way, it will act as a server so that any client (i.e., home gateway) can access to the functionality implemented and published over this platform.

As an important feature and requirement for the ease of use, sensors and devices need to be discovered and configured without user intervention. This allows adding and removing this kind of peripherics dynamically while it does not impose additional effort in setting up new devices or modifying the current platform configuration.

Besides providing raw data on demand, actuators can be connected to the car gateway, deriving further functionality after extracting information from the pool of data made available via the new platform. It is even possible that sensors located inside the car provide data storage via the integration platform for later analysis of vehicle status or passenger status.

Proprietary solutions are based on a set of sensors directly connected to a process unit that is responsible for the connected actuators supervision (cf. Figure 1). This architecture provides neither openness nor flexibility, because these connections are individually configured for each sensor type, and the process unit operation is considered to be customized. It is obvious that this model does not scale in a dynamic environment that has been mentioned before.

The solution proposed in this study is based on a fully distributed model in which each device is connected to the VAN, both wired and wireless. It also connects to devices attached to the CAN-Bus, and finally the car gateway (e.g., a dedicated light weighted embedded car PC) that can serve as the hardware platform (see Figure 2). This architecture supports a distributed and scalable model, which allows installing additional devices. That is, because the car PC provides all routines required to register and to interconnect most of the devices, it allows dynamic discovery of new devices and it installs the appropriate drivers.

Since additional sensors or actuators do not belong to the initial configuration of the static car infrastructure, it is assumed that they can join or leave the network in a dynamic way. Some device typology examples can be found in e-care environments which are tracking vital signals or other interesting parameters belonging to person monitoring capabilities integrated in wearable devices or as additional accessories [1]. All these target devices have one part in common: they will be incorporated into the car network, acting as servers for new features (services) registered in the car gateway (i.e., A/V cameras, A/V contents server, reproduction devices, smart infant seats, etc.). These will help to develop new added value e-safety applications like enriched emergency calls, driver and passengers wellness monitoring, and so forth.

In the following section, the new approach is compared to the state of the art and additional information about selected standards are also given. Section 3 explains the

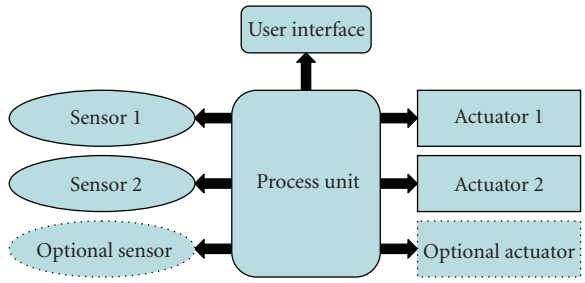

Figure 1: Dedicated sensors.

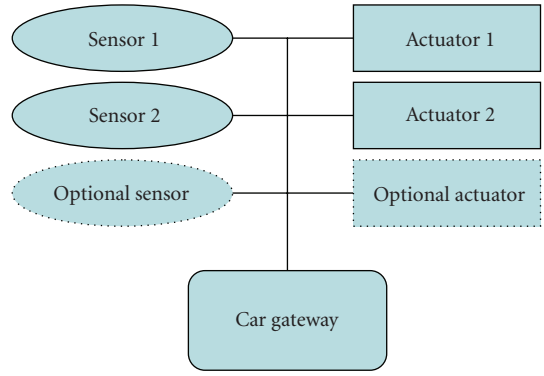

Figure 2: Sensor network.

design of the car gateway architecture and the integration platform embedded into the gateway. Then, Section 4 will describe the prototype implementation of the developed system. Finally, Section 5 summarizes the results obtained and Section 6 gives a brief outlook on the future work.

2. State of the Art

2.1. Current Initiatives. Nowadays, there are several initiatives running that try to develop embedded platforms installed within a vehicle system. They allow integrating the communication with all the different devices located into the car, so that both the information and the control of these devices are available. Furthermore, some of these initiatives also pursue to provide the vehicle with connectivity toward the external world, so that it can connect with any external servers located in Internet.

One of these initiatives is the proposal announced by Visteon, TACNET [2]. This technology is based on a central processing unit in charge of monitoring devices installed into a concrete vehicle. Thus, this solution does not provide any kind of scalability, since it is only compatible with devices previously installed and configured in the system.

Another similar initiative is implemented in the EASIS (electronic architecture and system engineering for integrated safety systems) project [3]. This one tries to develop an integration platform that manages both the sensor and control systems located in the car; in addition, it provides a gateway to connect to external devices. This platform

purpose is to deploy the necessary infrastructure to support a future software environment that will use this platform features in order to implement its functionalities.

There are many other similar initiatives as the "web-based control system" [4] implemented by the department for Information and Communication Engineering of Daegu University, or "DRIVE" [5] implemented by Telefonica I+D that try to achieve the in-vehicle devices integration, in order to homogenize the access to them. As can be seen, all these initiatives are focused on their systems adaptation to the existing devices and protocols, or on the new network architectures development specifically developed for sensor and actuators networks, as the (LIN local interconnect network) project [6]. Therefore, all of these activities will have the same scalability and incompatibility problem because vehicle devices that implement noncompliant protocols will not be accessible via their platforms.

Finally, one of the most important initiatives regarding these issues is (AUTOSAR Automotive open software architecture) [7]. It defines a platform that deploys an open automotive software architecture, capable of managing the heterogeneity of the sensors and devices connected with the vehicle. Furthermore, it is characterized by a run time environment (RTE) where applications defined by the providers will be deployed so that they are abstracted from the complexity under them, being able to access to the device and sensor data in a standard way. Thus, this initiative is able to solve the scalability and compatibility problems, standardizing both the in-car devices access and the applications installed over the RTE. However, this platform is only focused on solving the connection with devices and sensors directly linked to the vehicle using the CAN, FlexRay, or LIN interface. It is not compatible with many other devices that are involved into the car environment, like PDAs, media players, and so forth. Due to this restriction, this initiative has several constrains related to the connectivity and infotainment issues, limiting its capabilities to standard vehicle functions and not being compatible with telematics applications. To solve these restrictions, this initiative proposes to use the OSGi technology in order to develop a parallel framework. Thus, all the connections and devices not compatible with the actual development would be managed by dedicated modules (bundles) installed over this secondary framework. This solution has several drawbacks, first of all is that nowadays there are not many advances in this area and the integration of AUTOSAR and OSGi still have not been defined. Furthermore, this solution does not rely on the compatibility features of the AUTOSAR architecture, being necessary to create specific modules to solve the interaction with any new device.

Due to this, the system developed in this paper search to develop a platform that solves the above mentioned lacks. It will create an abstraction layer that allows the applications installed over this one not only to auto-register and manage the services and data provided by any kind of sensor or device, but also to publish some of their functionalities, acting as a server towards the external word. These features will allow to deploy the infrastructure for the implementation of new added-value applications like enriched emergency calls, environment monitoring, driver status inference, data exchange between two vehicles, and so forth characterized by the cooperation between many different kind of devices and servers. Furthermore, its architecture will be completely compatible with the AUTOSAR initiative, so that both platforms can run in a complementary way.

2.2. Technologies Overview. The main technologies needed to develop the platform are as follows. Firstly, a universal plug and play protocol that provides automatic discovery and configuration of new devices brought into the car network. Due to several reasons, those will be exposed in the next subsections; UPnP [8] has been selected as the discovery and autoconfiguration protocol.

Additionally, the platform is going to have a framework that implements a dynamic component model, so the sensors and any other devices could be attached to the local network but without the necessity to touch the original car configuration. In this way, the components can be added or removed dynamically and will be recognized by the system transparently. The platform is going to be installed on an OSGi [9] framework, so that OSGi UPnP services can map vehicle devices. As a result, the platform can register and offer services as OSGi services.

Finally, the session initializing protocol SIP [10, 11] will be used to provide communication to external networks capability, and mobility features, so that it can be located and accessible at any moment.

More details about these technologies and why they have been selected will be discussed in the following subsections.

2.2.1. UPnP. UPnP is an open and distributed software architecture that defines common protocols and procedures sequences in order to install interoperability between devices connected to the same local network. These features are as follows:

(i) easy management of heterogeneous devices,
(ii) automatic and dynamic discovery of devices,
(iii) management and support of services,
(iv) standardized behavior by the UPnP forum,
(v) small footprint,
(vi) UPnP provides a particular architecture to AV,
(vii) UPnP is compatible with QoS techniques,
(viii) UPnP offers a client/server architecture.

As mentioned, UPnP is an expanding technology, since more and more commercial devices are compliant to this standard. Furthermore, this technology maintains a small footprint so that it is assumed that UPnP is suited well for a prototype implementation of new sensors.

UPnP specification implements architectures and procedures to offer value-added services. The relevant parts cover the QoS subsystem, security subsystem, and AV services. All of them are well defined and some are well implemented as the UPnP standard architecture and the AV subsystem [12, 13].

Together with the OSGi framework, UPnP allows to publish registered services like framework services which can be made available by any application installed on top of it. In the same way, any internal service implemented by a developed application can be published as an UPnP service using this technology.

DPWS (devices profiles for web services) [14] also offers a dynamic component model like UPnP, and it also provides the necessary routines which implement a similar dynamic UPnP behavior. Furthermore, this technology is not limited to a local domain, and thus DPWS offers a possibility to publish devices-services registered by its infrastructure to the external world. But on the other hand, this technology is not sufficiently expanded yet in case it is compared with UPnP. Additionally, regarding the footprint of both technologies, UPnP is more suitable for constrained environment, because while the most lightweight implementations of DPWS are using around 100 KB of dynamic memory, UPnP can run correctly using less than 45 KB.

Given this, DPWS technology has been discarded in the first phase of the project, and UPnP technology will be integrated into the embedded platform.

2.2.2. OSGi. OSGi technology will be used in order to constitute the services platform infrastructure, since this technology allows developing a flexible and lightweight environment, which is aimed at developing dynamic service platforms.

This specification defines a framework that implements a dynamic component system. It possesses necessary primitives, and it allows to use modules already deployed in the same environment. In this way it offers new applications services which can be used by others. Based on Java, this framework makes it possible to deploy a multiplatform solution, which is independent of the underlying operating system as well as the kind of hardware where the platform is running.

All independent modules are called bundles, and they implement small, reusable, and collaborative components. These are managed by the OSGi platform that provides routines permitting to realize a dynamically discovery in order to establish a collaboration between them, as well as to start, stop, and remove these services via a specific bundle without the necessity of restart the platform.

The OSGi framework offers routines in charge of implementing the UPnP features, which are able to publish a specific service offered as a bundle of an UPnP service towards a local domain.

Another possible platform alternative would have been the .NET framework, but finally it was discarded due to the restrictions in the communication between applications, and in addition to the lack of a robust support mechanism to maintain extensible and dynamic systems as it has been analyzed by the article "Developing an OSGi-like Service Platform for .NET" [15]. Furthermore this decision allows this platform to be fully compatible with AUTOSAR initiative, so that both system can be running and cooperating in the same ECU or car-PC.

2.2.3. Session Initiation Protocol (SIP). SIP is a signaling protocol developed to initialize, modify, and terminate a user interactive sessions characterized by the intervention of multimedia elements like video, voice, instant messaging, online games and virtual reality.

This protocol offers personal and terminal mobility over WANs, so that any device that incorporates an SIP agent can be located and connected wherever it is. In addition, this protocol solves problems related to the device handover and multihoming.

Furthermore, the SIP protocol offers extended features related to authentication, encryption, and quality of service control that supplies capabilities to establish an external communication between two different sub domains.

Other technologies as alternatives to SIP are DDNS [16] and MOBIKE [17]. DDNS is based on the use of dynamic DNSs. In contrast, this technology does not support both handover and multihoming. On the other hand, (IKEv2 mobility and multihoming MOBIKE) is based on the use of the signaling protocol that integrates IPsec and IKEv2. This protocol is based on the security associations that are created by the device's IPs. This one provides handover and multihoming support, but the technology is not able to locate a specific device into different domains.

All the previous technologies are located at the application level of the OSI stack. Additionally, there exist some other solutions based on network and transport level protocols, like MIP (Mobile IP) [18] and mSCTP [19] respectively. Both ones are suitable solutions for our problem, but they are based on external infrastructures that support their functioning. At this point, and given that SIP has the best features and requires the least infrastructure, all these technologies were discarded, and SIP has been selected as the appropriate protocol.

3. Platform Design

The main objective of this project is based on developing a platform that deploys a homogeneous OSGi-UPnP environment. This platform will allow to implement applications in an OSGi environment and to hide the complexity related to the device control and information obtaining. In this way, the car gateway provides an open platform capable of autoregistering all devices located in the VAN, as well as their services and events, and it converts them into OSGi services accessible by any applications developed over this platform. In the same way, it allows to connect with any device or server located at the external network, so that it can register them as local services, like in the previous case.

Furthermore, this platform will implement the required routines, so that any application installed over the OSGi framework can publish its services as UPnP services towards the VAN and external network in order to make them accessible by any UPnP client.

The platform has been designed to be a fully modular system capable of deploying each of its functionality in blocks that are totally disjoint. In this way, the design option provides the ability to replace any block without any

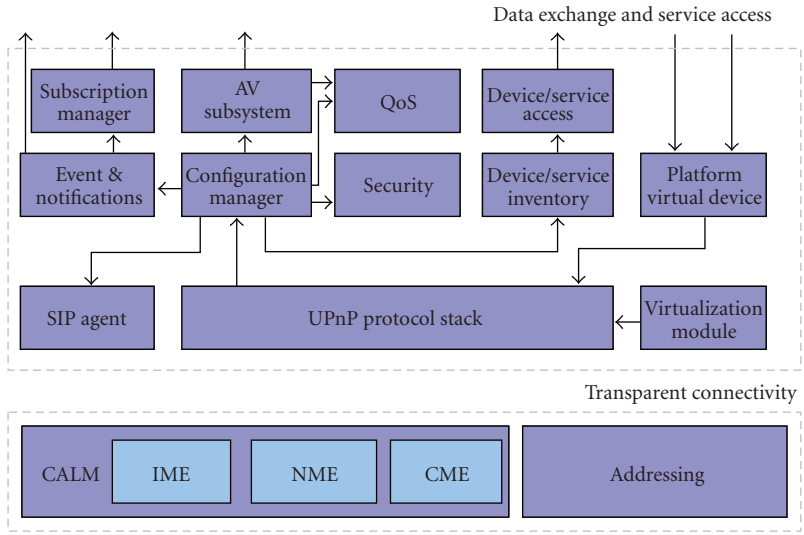

Figure 3: Platform architecture.

inconsistency in the system, so that the platform can be updated continuously.

The modular structure proposed can be seen in Figure 3. In the following subsections each layer of the system will be presented and exhaustively described.

3.1. Transparent Connectivity. Transparent connectivity is in charge of managing the communication between the vehicle and the external servers, as well as any device or sensor located within the vehicle environment. To this end, this module analyzes and selects the best communication technology possible or available (GPRS, UMTS, Wifi, Bluetooth. Zigbee, Ethernet, etc.) for the given service requirements and user preferences (*always best connected* (*ABC*)). This layer tries to adapt the networking process to the vehicular environment. To do this, its architecture is based on the CALM proposal (continuous air interface for long and medium range) [20].

Based on TCP/IP stack, this layer is formed by two submodules, as it can be seen in Figure 3. Firstly, the "CALM" module, which is formed by the next three coordinated entities: "*interface manager entity*" (IME), "*network manager entity*" (NME), and "*CALM manager entity*" (CME). IME entity is in charge of asking about the communication medium status, find the available resources at each moment considering the signal properties (signal strength, data rate, etc.) and maintain updated the stored information about the interface status. On the other hand, NME is responsible for considering the quality of service requirements for each application and compares them with each communication interface status. So, NME entity determines the optimum network selection, based on certain selection rules. Finally, CME entity acts as this module brain, supervising and managing the process execution and offering towards the upper layer the API to interact with this system.

Secondly, the "addressing" module is in charge of implementing the addressing of all the connected devices. This layer acts as an "access point" for any device located in the VAN, providing private IP addresses to all of them. Therefore, any connected device is going to be addressed in the same way.

All these modules have been developed as kernel modules, which act directly over the operative system routines in charge of carrying out the connectivity tasks.

3.2. Data Exchange and Service Access: Functionality Description. This subsystem defines the main part of the platform, which manages the heterogeneity of the connected car gateway devices and it provides a homogeneous environment to higher levels so that they can access in a transparent way to all the services offered. This is possible thanks to the routines implemented in order to publish and control all services provided by each device that is located both in the VAN and in any other external network.

Based on UPnP and OSGi, this allows incorporating, removing, and configuring new devices in a dynamic way without any human intervention.

Since not all connected devices will be compliant to the UPnP specification, this subsystem is responsible for implementing all required routines allowing the final platform to be abstracted from any proprietary protocols provided by these sensors and devices. In this way, this layer will emulate the UPnP behavior of each device that does not support this technology, and thus acting as a driver for non-UPnP compliant components. Additionally, an SIP agent has been developed in this layer, providing all required routines to interconnect the car gateway to external devices. This feature supports mobility of the platform in different networks.

Additionally, several modules are deployed to provide local and external car gateway communication. These management tasks include the following duties.

(i) Check for services offered by the connected devices and publish them as an OSGi services.

(ii) Publish OSGi services implemented over the framework as UPnP services towards local or external networks.

(iii) Manage events belonging to the connected devices.

(iv) Manage quality of service in the connections between the car gateway and each device, located in both foreign or local networks.

(v) Manage traffic security generated between components.

(vi) Manage AV connections between all AV components.

Each module implements an interface so that any other entity can configure it or access to its functionality. All of them will be described in more detail in the next section.

3.3. Data Exchange and Service Access: Layer Modules Description. The "data exchange and access service" layer (see Figure 3) is described in this sub-section. It is important to note that since the platform is based on OSGi framework, each module is implemented as a bundle installed and initialized in the OSGi framework. The communication methods, employed to connect the different modules and the upper layers (applications installed over this platform), are based on the mechanisms the bundles have in order to publish their services. Concretely, while the interaction between modules is going to be based on the Java libraries published by them; the interfaces offered by each module towards the upper levels will be based on OSGi services they publish and install over the framework.

UPnP Protocol Stack. This module is an implementation of the UPnP protocol stack. Apart from the protocols within the UPnP specification, this module implementation provides with higher abstraction Java libraries for developing generic UPnP devices and control points.

UPnP technology has been developed to be suitable in local domains. This is due to the mechanisms used to carry out the devices auto-discovery stage, because it is based on broadcasting. The UPnP stack implemented within this module tries to solve this constraint, extending such discovery mechanism. This extension is based on an on demand registration, sending unicast discovery messages (SSDP) towards the devices located at external networks. Therefore, this allows our platform to register and use the services implemented by any device, anywhere it is located.

Configuration Manager. This module implements the UPnP client, the "control point." This one is in charge of registering automatically all devices and services located inside the VAN. It sends this information to the correspondent module in charge of storing and publishing it ("devices/services inventory"). This one is based on the services the "UPnP Protocol Stack" module publishes internally.

Devices/Services Inventory. This module consists of a repository where the descriptions of all the devices connected to the platform are stored, containing all the information about the services and data published by them. In order to reach this target, it publishes an interface that is used by the "configuration manager" in order to add and remove all detected services descriptions, once a determinate device has been registered. Specifically, the information is stored in this module is:

(i) device/service identification

(ii) device/service capabilities (video, audio, data, etc.)

(iii) device/service services interfaces

(iv) device/service configuration interfaces

(v) data formats supported by the device (XML, codec, streaming, etc.).

The persistence mechanism that uses this module is based on a lightweight database where a simple relational model has been defined. Once all information has been stored, this module is in charge of publishing it towards the other modules, so that they can use it to carry out their objectives.

Devices/Services Access. This module is in charge of converting all the UPnP services, registered by the UPnP control point, into OSGi services. It defines the presentation layer of the services offered by the platform to the upper layer. Its functioning is based on the information retrieved from the previous module. Therefore, each OSGi service implemented is based on the UPnP description that each UPnP device has published.

Once the conversion has been done, this module registers the new services within the OSGi framework, where they will be able to be accessed by any application that runs over our platform.

Subscription Manager. This will publish services that allow upper layer applications to subscribe to asynchronous events coming from the device network. This module will offer the following services.

(i) Subscribe and unsubscribe to certain notifications coming from the sensor and device network. This service will permit to realize a subscription according to several criteria such as device family, device type, specific capacity, and so forth.

(ii) Subscribe and unsubscribe to events published when certain new devices are attached to the network (e.g., camera, specify sensor).

(iii) Query the system about the ontology of devices (families, types), provides capacities the callback channel on which the application wants to be notified.

This module use the asynchronous communication mechanism implemented by UPnP protocol to define high-level functionalities mentioned above. Furthermore, it is also

in charge of managing the keep-alive messages sent by the devices. In case this module notices a possible drop of a node, it tries to contact the device, but in case it is not possible, it proceeds to notify this event to the "configuration manager" which will proceed to eliminate the stored references to this device.

Events and Notifications. Based on the services published by the previous module, it completes the asynchronous communication mechanism of the platform. This module defines a confluence point responsible for centralizing the management of all events produced within the devices and sensors network. It acts as events router, in charge of receiving the events published by any connected device and routing them to the correspondent interested parties (platform modules or upper applications).

Its functioning is based on the eventualization mechanism implemented by Java. Each interested party registers its listener in this module, so when an UPnP event is received it is converted into a Java event by this module and notified to the interested entities.

QoS. Module in charge of managing the platform communications QoS. It is based on the QoS specification done by UPnP forum [21] (see Figure 4). This module implements the quality of service manager (QoS manager) and the policy manager (QoS policy Holder), offering the interface that allows the upper applications and platform modules (i.e., AV Subsystem) to invoke the VAN QoS control routines. In this way, before realizing a data transfer, the quality of service requirements are tested on the transmission channel, so if the quality is not ensured the connection will not be established.

To carry out this operation, this module makes use of a service published by any UPnP device that supports quality of service control (QoS device service). Such service provides the "QoS manager" with the necessary information about the device status as well as its connected networks.

The functioning of this module is independent of the connection typology (local domain or external network). While in the local domain case this module ensure by itself the quality within the communication, this system makes use of the routines implemented by the "transparent connectivity" layer in the connection with external devices to carry out the quality ensuring between networks.

Security. This module is in charge of providing the necessary routines to ensure the confidentiality and integrity in the platform communications. It works over the UPnP protocol stack, encrypting the messages exchanged into the UPnP control phase. The discovery and the description phases are intended to be public phases so they will not be encrypted.

Its objective is filtering the communications between the UPnP client that resides into this platform ("configuration manager") and any connected device, using an encryption system based on public key cryptography algorithms. Thus, the information delivered trough the network cannot be unencrypted by any other UPnP peer but the invoking client and the receiver device.

This module introduces a little overhead in the communication process. This is due to the necessary key negotiation in order to prepare the system to do the encryption process. Once this negotiation is done, the module is ready to encrypt any message.

Platform Virtual Device. This module contains the required routines to publish on demand any functionality implemented into a certain application (implemented over the platform) as an UPnP service. According with the (model view controller MVC) paradigm, while the business logic is implemented by each application, this module will define the presentation logic of these published services.

This procedure allows any UPnP client located within the VAN coverage to register and make use of these services implemented by such applications in a transparent way.

AV Subsystem. This module, based on the UPnP AV specification mentioned above, is in charge of handling all A/V communication. It implements the AV control point functionalities, extending the capabilities of the UPnP client deployed in the "configuration manager." This module offers a range of services to the upper layer that allows controlling the AV flows between media servers and renders. Thus, applications installed in the framework only have to determine which devices want to communicate to, and what action they want to execute (run, stop, rewind). As a result, the control point will provide the functionality required.

Additionally, this makes use of the "QoS" module functionality in order to ensure the quality of service of all the flows it manages.

SIP Agent. This module justification is mainly due to the platform mobile nature, and the local character of the UPnP technology. UPnP is oriented towards local networks, so it does not implement any mechanism that allows a client to register services published in external networks. Besides that, the platform will be located in a mobile environment, so it is necessary to take this into account. The reason is that the connectivity to the platform needs to be maintained always.

To solve both two problems, the SIP module implements an agent in charge of maintaining the connectivity between devices residing in an external server and in the car gateway, using an external SIP server (SIP proxy). As long as a connection between the car gateway and a device located in an external network is active, an SIP session will be established between both, using the external SIP server to manage the platform mobility feature. Once the SIP session has been established, a dialogue between both devices is possible.

Finally, it is necessary to clarify that the SIP agent always maintains updated the information within the SIP server about the vehicle location, which allows any external client to retrieve the network level address of the vehicle, enabling to access to the platform.

Virtualization Module. This module is responsible for implementing a virtualization of each non-UPnP device located

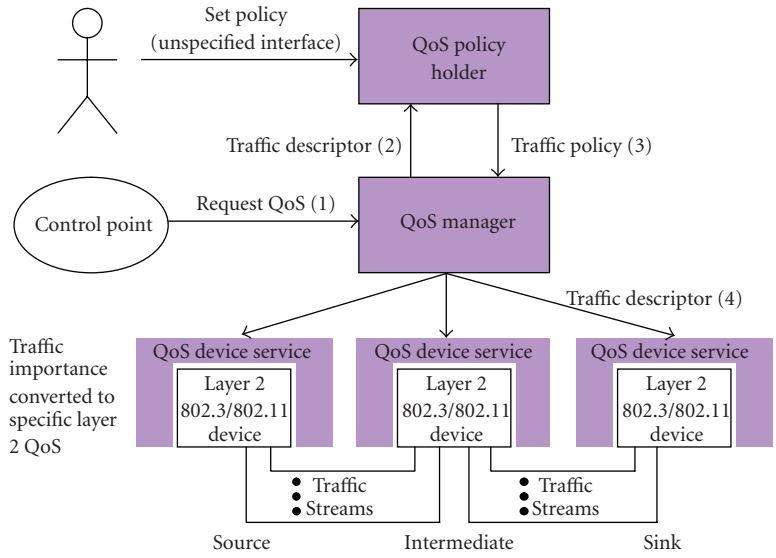

Figure 4: Local QoS subsystem.

in the platform's VAN. It acts as a concentrator for the information received from the different non-UPnP sensors and devices attached to the vehicle. It defines the interfaces towards them, using the IEEE 1451.X smart transducer standards [22]. Specifically, it makes use of IEEE 1451.5 protocol that covers the wireless communication protocol standards such as 802.15.1 (Bluetooth), 802.15.4 (Zigbee) and 802.11b/g (Wifi). On the other hand, it use the IEEE 1451.6 protocol to implement the interface towards the CAN Bus.

Over the previous interfaces, it implements the IEEE 1451.0 protocol to provide a uniform set of commands to access any sensors or device in the 1451-based networks. This protocol will act as a concentrator of all the other ones, redirecting the request done to a determinate device towards the correspondent IEEE 1451.X module.

Finally, this module makes use of the services published by "UPnP protocol stack" module to implement a generic UPnP server. This UPnP device publishes a set of generic services which maps all the commands offered by the IEEE 1451.0 protocol. Furthermore, this device is characterized by a set of state variables that store the value of each sensor at any time and publish UPnP events when this value changes. In this way, all the iteration with non-UPnP devices is done using this module, which virtualizes the UPnP behavior.

4. Prototype Implementation

The platform described in this paper has been deployed in a prototype. The main goal of this prototype is to check the suitability and adaptation of this system for the vehicular environment, as well as to test its correct functioning. The architecture of the implemented system, which can be seen in Figure 5, is characterized by the next layers.

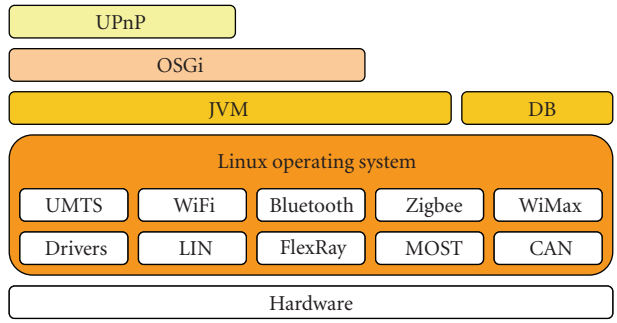

Figure 5: Platform structure.

Hardware. Composed of a Via EPIA CN 1300 mother board with a C7 processor. This module has 1 GB of RAM. It will be connected to a Xenarc 700TXV-B screen.

These hardware specifications are proper to the constrained computers that are installed into the vehicles. This is going to allow to check if the developed software is suitable for these kinds of environments.

Operating System. Linux kernel version 2.6 has been selected to implement the platform operative system because it is free software and it is an open source initiative. It provides the robustness, potency, and reliability required by the system. Furthermore, it allows to execute several services and multiple applications like simultaneous processes running in parallel, essential requirement for the developed platform.

JVM. Java has been selected to separate the platform from the OS and to provide portability. For the development, the JVM 1.6 version implemented by SUN has been used. It has been necessary to use this version because it provides new

routines that are required by some functions, like the QoS management.

Framework. OSGi has been selected to act as the development framework because it provides versatility and modularity needed by the platform. Specifically, the implementation selected for the prototype has been the Equinox framework version 3.2.2. This distribution is a free implementation of the OSGi specification release 4.

Plug & Play Technology. The UPnP protocol stack implementation developed by "Ciberlink" [23] has been taken as reference to create the extended UPnP protocol stack used in order to create the UPnP client (control point) as well as the servers (devices) which are implemented into the car gateway.

Database. The persistence mechanism has been implemented using SQLite version 3.5.6. This database manager has been selected in order to achieve the balance between a satisfactory throughput and a small footprint. This will facilitate the integration in small portable devices.

Carriage Return Introduced. Once the complete system has been implemented and run correctly, platform behaved as expected. The platform is completely suitable to the vehicle sensor and actuators network already deployed, being capable of retrieving all data and services offered by them and translating them into OSGi services and Java events. The same result has been obtained with several *new-age* UPnP devices (e.g., the pocket PC Nokia 880 or mobile phone Nokia N80) and with servers implemented in both and attached to the VAN and in an external network. Finally, regarding the platform services publication, it was possible to verify that the developed system is able to act as a server towards the VAN and the external world, so that any UPnP client can register such services and make use of them, independently of its location and the vehicle mobility.

This implementation helped us to detect some lacks. The main issue is the integration of vehicle sensors and actuators with respect to their compatibility. Even though, the platform offers standard access to the vehicle network, each vehicle company uses proprietary standards to access to its devices. Due to that, it was necessary to tune the platform towards sensors of a limited number of companies. Furthermore, due to security reasons the platform will not actively interact with the car network, but being able to access to the sensor status in a read-only way. The current prototype does not support real-time features, which might be an upcoming issue when further proceeding with the integrations tasks with AUTOSAR and OSGi. For the current application area real-time features are beyond the scope of the implementation.

5. Conclusion

It is more and more common that intelligent environments are present in daily lives. This project tries to establish the fundaments to integrate different isolated intelligent environments that are located in a vehicle network as well as to deploy a system capable of recognizing and registering all kind of devices and sensors not belonging to the vehicle in order to provide the in-car system with the value-added services and data captured by them. The main objective is to offer a platform that allows connecting all kinds of devices, independent from their technology, localization, and communications interface they use, while maintaining the ease of plug and play. Furthermore, the platform developed allows the vehicle to play a server role, being able to publish a set of services towards the external world, so that any client can make use of them.

The system deployed in this approach combines the use of technologies such as UPnP, OSGi, and SIP to reach this target. This integration provides a common protocol to interact with registered devices in a dynamic environment in which it is possible to install, remove, or control any sensor or device without prior configuration, in case the device is accessible. The developed platform implements the necessary functionality able to resolve disadvantages arising from the mentioned heterogeneity, as far as communication protocols are concerned. The platform mobility support is maintained with the help of routines that manage changes of vehicle addressing when it is moving, and furthermore, provide access to external networks.

Additionally to all these functionality, the described abstraction layer also hides problems that may occur in proprietary solutions, maintaining the flexibility on a device's compatibility as well as on the platform's scalability.

6. Future Work

One of the main future works planted after this project is regarding the real-time capabilities. Nowadays, the developed platform does not support real-time requirements, not being suitable for certain applications that require this kind of features. In these cases, these applications have to be developed over another system like AUTOSAR, so that both platforms can run in a complementary way. Therefore, the principal research line is based on the adaptation to a middleware that supports hard and soft real-time requirements as well the provision of the necessary routines to manage them.

Related to the plug and play requirements, the platform must be able to register all devices and publish them in the vehicle's PAN network. Currently, the platform is based on UPnP as plug and play technology, and this implies that all devices need to be compliant to UPnP. In other cases, it is necessary to make use of the "device virtualization" module that acts as a driver and is in charge of translating the proprietary communication protocol. A future work line is to develop a module parallel to UPnP in the "data exchange and service access", that allows to register and interconnect these devices automatically, as well as to improve the "device virtualization" module so that it can be compatible with any vehicle kind of devices making use of the standards developed by AUTOSAR.

Acknowledgments

The Car Gateway system has been partly developed inside the Caring Cars [24] project, financed by the Spanish Ministry of Industry under project number FIT-330215-2007-1 and the InCare project, financed by the Spanish Ministry of Education, and Science under project number TSI2006-13390-C02-01 respectively.

References

[1] "Intelligent Healthcare Monitoring based on Semantic Interoperability Platform (SAPHIRE)," Deliverable 3.1.1, 2008, http://www.srdc.metu.edu.tr/webpage/projects/saphire/index.php.

[2] TACNET by Visteon Aftermarket Operations: Innovative-Law Enforcement Equipment, April 2008, http://www.evisteon.com/.

[3] EASIS, April 2008, http://www.easis-online.org/wEnglish/news_easis/news.shtml?navid=1.

[4] T.-H. Kim, S.-I. Lee, Y.-D. Lee, and W.-K. Hong, "Design and evaluation of in-vehicle sensor network for web based control," in *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS '06)*, pp. 209–218, Potsdam, Germany, March 2006.

[5] C. Pinart, I. Lequerica, I. Barona, P. Sanz, D. García, and D. Sánchez-Aparis, "DRIVE: a reconfigurable testbed for advanced vehicular services and communications," in *Proceedings of the 1st Workshop on Experimental Evaluation and Deployment Experiences on Vehicular Networks (WEEDEV '08)*, pp. 1–18, Innsbruck, Austria, March 2008.

[6] "Local Interconnect Network (LIN) / SAE J2602," April 2008, http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=02WcbfNZnLnRys.

[7] AUTOSAR (AUTomotive Open Sortware ARchitecture), January 2009, http://www.autosar.org/.

[8] Universal Plug and Play (UPnP) Forum, April 2008, http://www.upnp.org/.

[9] Open Service Gateway Initiative (OSGi) Alliance, April 2008, http://www.osgi.org/.

[10] SIP, Session Initialization Protocol, Internet Engineering Task Force (IETF), Network Working Group RFC: 3261, 2002, http://tools.ietf.org/html/rfc3261.

[11] SIP Forum (a Swedish non-profit association) and SIP Forum LLC, April 2008, http://www.sipforum.org.

[12] Universal Plug and Play (UPnP) Forum, UPnP AV Architecture:1, June 2002, http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v1- 20020622.pdf.

[13] Universal Plug and Play (UPnP) Forum, MediaServer V 2.0 and MediaRenderer V 2.0, March 2006, http://www.upnp.org/specs/av/.

[14] Devices Profiles for Web Services (DPWS), February 2006, http://schemas.xmlsoap.org/ws/2006/02/devprof/.

[15] C. Escoffier, D. Donsez, and R. S. Hall, "Developing an OSGi-like service platform for .NET," in *Proceedings of the 3rd IEEE Consumer Communications and Networking Conference (CCNC '06)*, vol. 1, pp. 213–217, Las Vegas, Nev, USA, January 2006.

[16] DynDNS, April 2008, http://www.dyndns.com.

[17] P. Eronen, "IKEv2 Mobility and Multihoming Protocol (MOBIKE), Internet Engineering Task Force," RFC 4555 (Proposed Standard), June 2006.

[18] C. Perkins, "IP Mobility Support," RFC 2002, IETF, October 1996.

[19] Ł. Budzisz, R. Ferrús, A. Brunstrom, et al., "Towards transport-layer mobility: evolution of SCTP multihoming," *Computer Communications*, vol. 31, no. 5, pp. 980–998, 2008.

[20] ISO TC 204 Working Group, "CALM project," September 2008, http://www.calm.hu/.

[21] Universal Plug and Play (UPnP) Forum, Quality of Service V 2.0, October 2006, http://www.upnp.org/specs/qos/.

[22] K. Lee, "IEEE 1451: a standard in support of smart transducer networking," in *Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference (IMTC '00)*, vol. 2, pp. 525–528, Baltimore, Md, USA, May 2000.

[23] S. Konno, Tokyo, Japan, April 2008, http://www.cybergarage.org/.

[24] Caring Cars, MEDEA+ project 2A-403, financed by the Spanish Ministry of Industry under project Num. FIT-330215-2007-1, 2008, http://www.medeaplus.org/.