Vehicle service middleware based on OSGi

Juan Luo
School of Information Science and Engineering
Hunan University
ChangSha, China
juanluo@hnu.edu.cn

Feng Wu
School of Information Science and Engineering
Hunan University
ChangSha, China
wufeng@hnu.edu.cn

FAHMI AMEEN ABDO ALKUBATI
School of Information Science and Engineering
Hunan University
ChangSha, China

Abstract— **Embedded system for the automotive electronics becomes increasingly powerful, whereas the system structure becomes more complex. There is a high coupling between hardware and software which makes it hard to develop applications. To solve this problem, we proposed a middleware based on OSGi, namely OSGiIV. By embedding the middleware into the ECU (electronic control unit) in the vehicle, we can provide a unified interface for different ECUs, and provides a service-oriented application programming environment for the car platform applications.**

Keywords- OSGi; SOA; middleware; ECU

I. INTRODUCTION

With the development of automotive electronics, it develops from traditional electric into the smart system. To meet the growing demands of people, embedded system functions based on the automotive electronics is becoming increasingly powerful. But its systems structure becomes more complex, resulting in the increasing costs of development and maintenance. And different electronic control units in vehicles have different hardware environments, which makes the coupling between hardware and software very high, reduces reusability of automotive electronics software, and restricts the development of automotive electronics.

Embedded middleware is proposed to be a good solution to the above problems. Embedded Middleware is a software layer between operating system and application software. Putting the embedded middleware into embedded devices is capable of providing a unified operating environment for the upper application layer software, and coordinating functions between lower and upper layer. In fact, embedded middleware plays a role of a common interface. With the interface, the upper software development can be independent of the underlying hardware environment. Compared with general middleware, the embedded middleware has minimal kernel, which makes its operating speed and efficiency not affected after being placed under the embedded device.

OSGi (open service gateway initiative)[1] is not only a typical service-oriented components system, but also a dynamic, light-weighted middleware platform. Applications or components in the form of bundles for deployment can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot. Application life cycle management is done via APIs that allow for remote

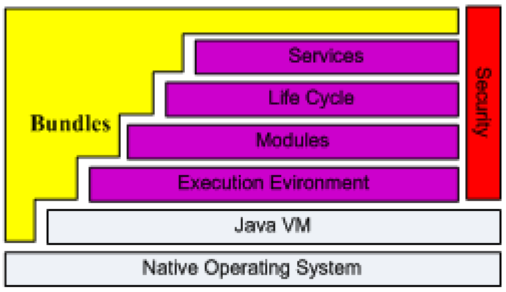


Figure 1. OSGi framework.

downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly, as shown Fig.1. With the help of OSGi, we can reuse the resources that were used in the framework, which will reduce a great deal of cost.

Based on the OSGi middleware, this paper designs an embedded middleware platform of service-oriented architecture (SOA) [2] which named OSGiIV. Embedding it into the electronic control unit of the vehicle provides a unified interface for various automotive electronic means, and facilitates the rapid processing between various tasks and data. The dynamic and modular characteristics of the middleware reduces the coupling degree between electronic control unit and the upper application, provides a service-oriented programming environment , and makes it easier and faster to develop upper applications.

II. RELATED WORK

Many literatures have extended the study and realization of traditional OSGi platform. Rellermeyer[3] achieved the interoperability of OSGi applications by extending the traditional centralized, industry-standard OSGi platform to a distributed middleware platform, which greatly simplifies the development of distributed applications with low overhead of performance. But it is invasive to the OSGi programming model, and cannot interact with non-OSGi system. Shi [4] proposed CORBA-based distributed OSGi model, which supports interoperability among multiple OSGi applications and between non-OSGi and OSGi, and it reaches the goal of low–invasive and high scalability. Lai[5] analyzed the P2P multimedia sharing mechanism home network and he found that the transmission could only be achieved with the use of P2P networks, but when the content server and the client have adopted this system, the transmission speed of the internal network could not increase. To solve this problem, the OSGi middleware was added to the DLNA -based multimedia sharing system expanding the network to be an OSGi-based P2P one, which effectively improves the quality of service for users. For the smart home service network with limited or unreachable resources, Cheng[6] designed a service management mechanism based on priority scheduling algorithm by embedding the middleware into the service gateway to ensure the quality of service and better dealing with emergency situations. But this priority-based service management platform cannot be called among multiple platforms, and thus its scope of application is limited.

III. SYSTEM DESIGN

OSGi-based vehicle service middleware namely OSGiIV (OSGi in Vehicle), and its essence is a communication interface, it can not only masked the underlying different hardware environment, but also facilitate the further development of the upper application. The overall framework of middleware in OSGiIV is shown in figure 2.

A. *Overall Framework*

In order to simulate a real-time vehicle network conditions, we use the Linux operating system in this paper, and use SocketCAN as a CAN bus network interface which is encapsulated by Linux, it can effectively simulate the real-time communication mechanism between upper network and underlying hardware devices. The vehicle service middleware must rely on java virtual machine (JVM) runtime environment, and the upper part is the core of OSGi framework, its various functions and application modules are presented in the form of bundle components.

Schedule center is a primary interface provided by OSGiIV middleware. When the network has multiple information needs to be passed, we should determine the order of information transfer by appropriate rules designed by the Schedule center.

Service scheduling is a service management, which is also an important component specification in OSGi SPR4. We could introduce a priority scheduling method to OSGi framework to improve the performance of framework. In the OSGi framework, the coordination between service schedule component and multiple components could reduce collision rate when multiple tasks are simultaneously transmitted in bus. When the access collision rate is decreased, then the number of repeated request will be reduced, and so does the number of service request. In addition, service scheduling management can efficiency help the whole system quickly complete the task.

CAN socket and CAN data Listener work well to allow underlying SocketCAN framework to be a service in OSGi framework. This not only provides a pure java implementation for CANSocket, but also exchanges data in CAN.

Vi Admin is an information translator. It is mainly used for processing data from the underlying CANSocket, and translating data into java language that can be recognized by

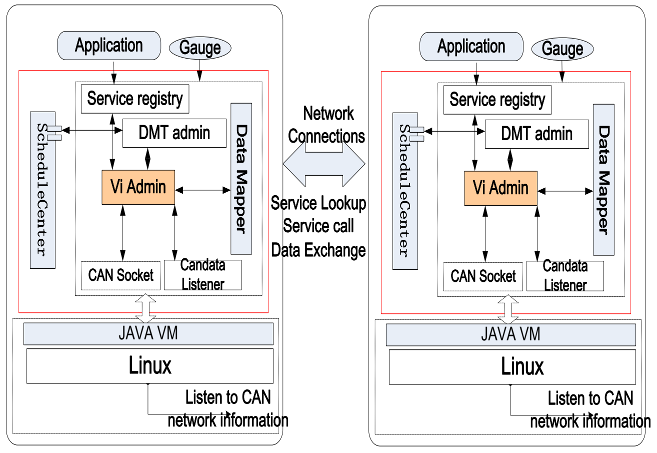


Figure 2. Overall Framework of OSGiIV.

various components of the OSGi framework. It also implements the upper application to send and to receive vehicle information without considering the underlying structure of the vehicle.

DMT admin is a data management center. This bundle manages data by OSGi DMT (OSGi Date management tree), and it provides a pattern of access to vehicle data. DMT (Device Manager Tree) is used to store and manage vehicle information. And DMT provides appropriate metadata for the upper application. The application running in a local environment can also get the vehicle sensor data through data center module.

Data Mapper Bundle is a data mapping component. This component transforms byte array information into data which can be identified by specific application, namely it provides a function of transmitting arrays into java objects.

Service registry is the native components of the inner framework.

B. *Working mechanism*

In figure 2, once local OSGi monitors the corresponding sensor information, the functional unit SocketCan of Linux operating system will be used to transmit original data information coming from the bottom. The virtual machine JAVA provides a good execution environment for keeping the middleware running this OSGiIV. In OSGi, however, Candata Listener listens for the underlying data information.

When it gets access to the corresponding data flow, the interface of CAN socket component will be used to contact with the underlying CAN network session. At this time, the transmission data is a kind of original data which cannot be identified by multiple components within the OSGi framework. So CAN socket components will transfer some corresponding information to translation center vi Admin, making the upper application send and receive information without considering the architecture of the bottom. In order to manage efficiently a large amount of Data, vi Admin will do two operations simultaneously. The first step is to transfer corresponding Data to the Data mapping center Data Mapper for a particular format, and send it to the Data management center DMT admin, storing and managing the corresponding data in a tree structure. Then, the corresponding service information will be registered in the service registry. Lastly, core information will be dispatched in the form of services by the schedule center.

C. Service requesting and monitoring mechanism

Assuming that when a certain application needs some latest information service, first of all, it will ask the schedule center for help. Schedule center at this time will transfer the asking information to DMT data management center DMT. At this time, DMT will register the service information to the service registry as the latest service information, realizing a binding between the requester and the service consumer. At the same time, the service requesting is synchronized to the VI admin, which is used to get the data translated by the VI admin. Then the original data obtained from the underlying will be translated and mapped by VI admin to the DMT data management center. Management center DMT will transmit the request data to the schedule center. Finally, the schedule center will send the request information in order according to certain priorities.

At the same time, in order to capture the service registry events of the remote framework and put forward the service request for remote service center, every situation in the remote framework needs to be monitored, and then the future action needs to be determined according to the results.

D. Date Management mechanism

After the internal components DmtAdmin obtain the session request information， it will get access to data and information by getNodes, getSessionId and getType methods from internal data centers , and find its relevant information in a order of starting from root node to itself. By matching the node name and Access Control List (ACL) it can find itself. When adding or deleting a data, it not only increases or deletes corresponding data in the tree nodes, but also register this event in the registration. It will pass it to other components through the session after obtaining the information.

In general, the framework provides a remote node management model to manage bundle’s cycle and monitor framework’s state. In order to change its state, such as install a new bundle, it must at least have one session with the framework node. The established model is used to reflect the status of the requesting information. When the session is transmitted out, the component on the bottom should change the status into real-time status.

E. Factory pattern mechanism

Factory pattern is commonly used as a design pattern. It usually instantiates a class which contains a lot common interfaces, without having to know which class is to be instantiated in advance. The OSGi framework uses this method while expanding the service.

While creating a Service Management bundle, we definite service factory API in order to allow the application build an object parser tree from an XML document. This is the application of a simple factory pattern, usually when we start and initialize a bundle, we will use it. And some services consume a lot of system resources, so we hope this phenomenon will not last for a long time, and then you need to destroy them when you do not apply it. Implementation of Service Factory interface is a good way to solve this problem.

F. Tracker mechanism

The middleware use tracker mechanism to avoid some unnecessary problems and track the event of registry and uninstalling.

OSGi framework is a dynamic, multi-threaded environment, so the callback mechanism can occur between different threads at the same time. However, the dynamics also brings many difficulties. For example, it is difficult in this environment effectively monitor service and bundle real-time status. Because BundleListener and ServiceListener interfaces can only access the status change, not the state of existence. This leaves developers a big problem, in which the state of existence cannot be combined with the change of state for a bundle. This phenomenon is obvious under multithreading environment. The bundle tracker and service tracker can be a good solution to this problem.

IV. EVALUATION

In this section, we will describe our experiment and show the display to prove the system’s use and functions.

We should install a virtue machine in our pc. And then installing a Linux operation is necessary for our experiment to simulate a real-time CAN BUS in vehicle network. What is more, we should install the eclipse software to provide a running environment for the OSGiIV.

When we start the SocketCAN in Linux to send message and start the OSGiIV which should be configured in advance in eclipse to receive the relevant information, the demo bundle will display the information in the screen.

Figure 4 shows the velocity gauge demo. When the system finds the change that occurs in the DMT velocity’s node through the relevant functional component, it will display the change on the screen. This proves our platform work well with real time system, and we can develop applications without knowing the underlying structure.

At the same time, we put two algorithms into our platform to make sure our middleware is able to work well in the real time system, as shown in figure 5. In figure 5, there are 4 demos which can select dates from our platform and compare them in the demo. After we go through 3steps, we will see the demo4, which tells us NOPA algorithm is better than OPA algorithm. Meanwhile, this comparison experiment tells us our middleware platform can work well with different algorithm in real time system.

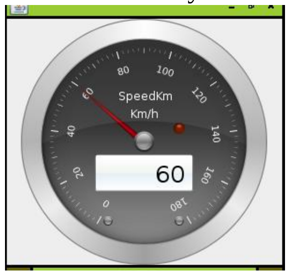
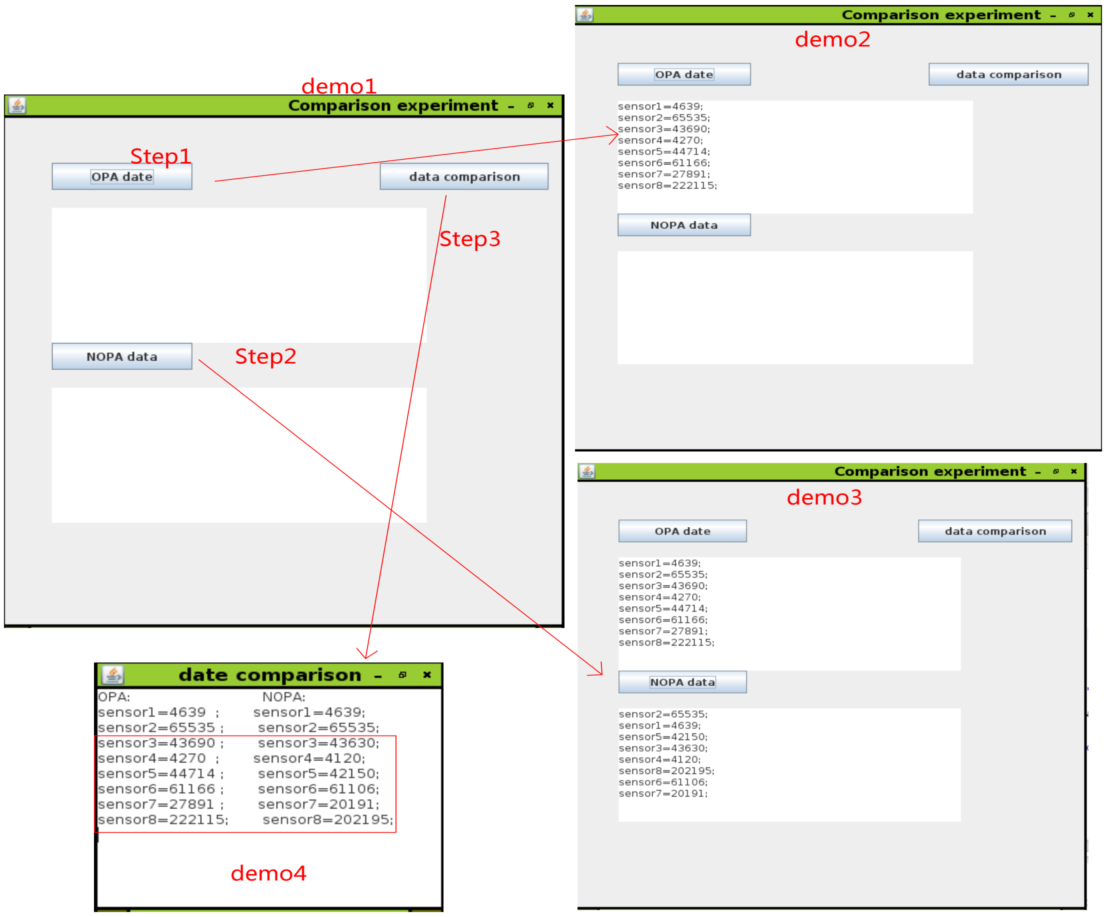


Figure 4. Velocity Gauge Demo



```
sensor1=4639;
sensor2=65535;
sensor3=43690;
sensor4=4270;
sensor5=44714;
sensor6=61166;
sensor7=27891;
sensor8=222115;
```

```
sensor1=4639;
sensor2=65535;
sensor3=43690;
sensor4=4270;
sensor5=44714;
sensor6=61166;
sensor7=27891;
sensor8=222115;
```

```
sensor2=65535;
sensor1=4639;
sensor5=42150;
sensor3=43630;
sensor4=4120;
sensor8=202195;
sensor6=61106;
sensor7=20191;
```

```
OPA:                  NOPA:
sensor1=4639  ;       sensor1=4639;
sensor2=65535 ;       sensor2=65535;
sensor3=43690 ;       sensor3=43630;
sensor4=4270  ;       sensor4=4120;
sensor5=44714 ;       sensor5=42150;
sensor6=61166 ;       sensor6=61106;
sensor7=27891 ;       sensor7=20191;
sensor8=222115;       sensor8=202195;
```

Figure5. Comparison experiment in our platform.

V. CONCLUSIONS AND FUTURE WORK

In this paper we implemented a middleware based on OSGi. By designing its contructure and adding some functional components into the OSGi, we provide a unified interface for upper application. The experiment shows that this middleware is of the function we need, and we can develop upper applications easier and faster. The future work is to embed it into ECU in vehicle.

ACKNOWLEDGMENTS

This work is partially supported by Program for New Century Excellent Talents in University (NCET-12-0164); National Natural Science Foundation of China (61370094); Natural Science Foundation of Hunan(13JJ1014).

REFERENCES

[1] http://en.wikipedia.org/wiki/OSGi

[2] Guinard D, Trifa V, Karnouskos S, et al. "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services,"Services Computing, IEEE Transactions on. 223-235,2010.

[3] Rellermeyer J S, Alonso G, Roscoe T. "R-OSGi: distributed applications through software modularization," Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware. Springer-Verlag New York, Inc.1-20,2007.

[4] Shi Dianxi, Wu Yuanli, Ding Bo,et al. StarOSGi: A distributed extension middleware for OSGi[J].Computer Science,162-189,2011.

[5] Lai C F,Chen M, Vasilakos A V,et al. "Extending the DLNA-Based Multimedia Sharing System to P2P Network on OSGi Frameworks," Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE.1-5,2010.

[6] Cheng S T, Chou C L, Horng G J. "Priority-Oriented Architecture Service Management on OSGi Home-service Platform," Wireless personal communications, 611-628, 2013.

[7] Tian Jing, Huang Yalou, Wang Liwen, et al. "Application of Fixed Priority Schedule Algorithm in CAN Bus," Computer Engineering.94-96, 2006.