Automotive Cloud Service Systems Based on Service-Oriented Architecture and Its Evaluation

Akihito Iwai
Corporate ePF Deptartment
DENSO CORPORATION
Kariya, Japan
akihito_iwai@denso.co.jp

Mikio Aoyama
Department of Software Engineering
Nanzan University
Seto, Japan
mikio.aoyama@nifty.com

***Abstract*—This article reports our concept and experiences on ACSS (Automotive Cloud Service System) based on SOA (Service-Oriented Architecture) for the next-generation automotive software platform. Along with rapid deployment of cloud computing, we expect, automotive software is evolving to ACSS where vehicles are collaborating with outside cloud computing and a variety of social networks. We propose an ACSS based on SOA named DARWIN, and demonstrate the validity of DARWIN with case studies running on a prototype electric vehicle. This article contributes to reveal key aspects of new software architecture for the next generation automotive software of integrating software in a vehicle and cloud services out of vehicles seamlessly.**

Keywords-component; cloud computing, SOA, Service-Oriented Architecture, REST, automotive software, AUTOSAR, embedded software, telematic services

I. INTRODUCTION

Nowadays, the intelligent safety control systems such as ACC (Adaptive Cruise Control) and LKS (Lane Keeping Systems) are widely deployed in automobiles in many countries. Due to the high demand such advanced services, automotive software systems become large-scale and highly complicated. Moreover, automotive software systems are connected to the traffic infrastructures [10]. Through telematics services, you can remotely control your car from your mobile phone. With rapid deployment of cloud computing[1], we expect, automotive software is evolving to the ACSS (Automotive Cloud Service System), which make vehicles work together with a variety of information services outside the vehicles [2, 11].

We further forecast that the space of service collaboration will be enhanced widely because vehicles will connected to everybody/ everything in our society such as home, office, commerce, and government along with the prevailing of PHV (Plug-in Hybrid Vehicle) and EV (Electric Vehicle) toward a smarter society, which is greener, safer, more convenient and comfortable.

In this paper, we propose DARWIN, the highly safe and reliable SOA-based architecture for the ACSS, and demonstrate the validity of the architecture with case studies running on the DARWIN prototype implemented in an electric vehicle. We discuss DARWIN from an architecture point of view in order to evaluate the architecture for the future value-added orchestration between in-vehicle and out-vehicle services.

This article is organized as follows: Chapter 2 identifies key challenges focused on service integration in the automotive domain. Chapter 3 refers to related works. In Chapter 4, we discuss key requirements to ACSS and introduce architectural solutions. Then, we propose the DARWIN Architecture, a core idea of this work, in Chapter 5. Chapter 6 illustrates a prototype of DARWIN implemented in a passenger car of electronic vehicle, and demonstrates the case studies on the prototype with usage scenarios, followed by discussions in Chapter 7. Future study and conclusions follow in Chapter 8 and 9, respectively.

II. KEY CHALLENGES

As illustrated in Fig. 1, ACSS includes many IT infrastructures, mobile devices, cloud services, traffic infrastructure, home network, and vehicles. They interact with one another in vehicle-centric manner everywhere at anytime. We start to discuss the key challenges of the ACSS.

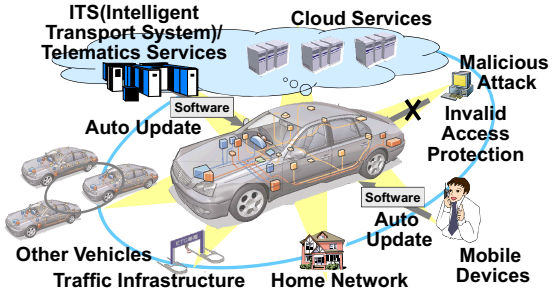

Fig. 1 ACSS: Automotive Cloud Service Systems

A. Guarantee of Real-time Performance

The automotive software needs to assure the strict time constrains [5]. If the specified timing is not met, controllability may be diminished or in the worst case, the system might fail to work. Hence in the ACSS, the guarantee of real-time performance becomes a challenge while, for

978-0-7695-4460-1/11 $26.00 © 2011 IEEE
DOI 10.1109/CLOUD.2011.119

instance, sensing sensor data or controlling actuators via the Web from devices outside the vehicle.

B. Assurance of Safety, Reliability,a dn Security

The automotive software must be safety-critical as faults can be fatal to human life. Therefore safety and reliability are the critical requirements in the development of the software systems. Even if the information devices outside of vehicle are failed, the automotive ECU needs some mechanism to tolerate the impact of the failure. The information security needs to be assured similarly.

C. Rethinking Automotive Software Engineering

The product cycle of the automotive devices is much longer than that of the computing devices. In addition, the software can upgrade its functions after the deployment to the systems. We can provide the services on demand from the service providers on the Web, such as cloud computing and SaaS (Software as a Service) [1]. While ever increasing demand to automotive software systems, conventional software architecture and associated development model may not be effective in the age of cloud computing. We need to rethink the way of the development and deployment of automotive software, while assuring the fundamental requirements of safety, reliability and real-time performance. Along with the maturity and wide use of the SOA and cloud computing, we are developing a new architecture for automotive software.

In the automotive domain, service-oriented technologies are emerging just recently. They include NGTP (Next Generation Telematics Protocol)[14] and GENIVE[9] for telematics services. However, standardization is not yet established, except for AUTOSAR (Automotive Open System Architecture), an in-vehicle software platform [3].

III. Related Works

We discuss the following related works to this study.

A. SOA: Service-Oriented Architecture

SOA has been adopted into enterprise software for integrating the information and communication services seamlessly on the Web [7] For applying SOA to the time-constrained systems, Real-Time SOA, a real-time extension of SOA, is proposed, and applications to automotive software emerged [2][4][5][11][13][18]. However, most of conventional works focused on a certain part of automotive software, especially so-called infotainment software. Looking at the ever increasing demands to software, it's needed to rethink the entire architecture of automotive software systems.

B. AUTOSAR

AUTOSAR (Automotive Open System Architecture)[3] is the open and standardized automotive software architecture for enabling to reuse software components in automotive software systems. As a platform for composing components, an abstract software bus named VFB (Virtual Function Bus) is proposed. AUTOSAR is successfully used to integrate components within an ECU. However, it's still an open problem to integrate software in-and-out of vehicle over multiple different networks.

C. OSGi™

The OSGi™ specifications define a gateway to a service-oriented architecture[15]. It's applied to a gateway converting protocols between conventional in-vehicle networks. However, the scope of networks is limited within an automobile.

D. Telematics Services/ITS and Cloud

Modern automobiles, especially premium cars, are equipped with a data communication module, and can send and receive data over standard protocols include HTTP and SMTP. With the capability, many new services are introduced as telematics services. They include remote control of door and windows, and remote security for disabling engine. Similarly, government-led ITS (Intelligent Transport System), provides an infrastructure services. However, current technologies, application level protocols and interfaces, of telematics services, are not yet standardizes. Moreover, those services are closed in the sense only specific services provided by automobile manufacturers and telematics services providers are accessible. Looking at the rapid evolution and deployment of cloud computing and SaaS [1], it's needed to provide a platform enabling application level interoperability between automotive software and cloud computing.

IV. Key Requirements To ACSS

Key requirements to ACSS are identified by reviewing key technical elements expected in the near future of automotive software.

A. SOA in-and-out of Vehicle

Major requirements to the SOA in-and-out of vehicles include the followings.

(1) The each in-vehicle ECU (Electronic Control Unit) software component or service is able to communicate and collaborate with one another.

(2) The in-vehicle ECU software component and the out-vehicle information service are able to communicate and collaborate with each other safely within time constrains.

(3) The service provider is able to specify the optimized service processes for a variety of user requirements and the vehicle context.

The service continues to run even when the network is disconnected by the momentarily communication or power failure.

B. Information Sharing over Networks

Requirements for service sharing include the followings.

(1) The service provider is able to manage and share information regarding the user (e.g. driver and/or passenger) and the vehicle operations (e.g. vehicle speed, fuel consumption, location, and so on) by invoking the service interfaces via the network in safe.

(2) The service provider is able to add, remove and modify information regarding on the users and the vehicles from outside vehicle in a safe manner after the vehicle is purchased.

C. Non-Functional Requirements

Non-functional requirements include the followings.

(1) The system is intended to provide the prominent ability to be fault tolerant against unpredictable failures.

(2) The in-vehicle components can be equipped at a reasonable cost.

V. DARWIN ARCHITECTURE

We discuss the concept, architecture and core technologies for ACSS named DARWIN.

A. Architecture Overview

The DARWIN Architecture is illustrated in Fig. 2. It presents the conceptual structure of platform for service interactions between in-vehicle and out-vehicle. The DARWIN platform consists of the following two main subsystems:

(1) SPM (Service Process Manager), and,

(2) DSS (DARWIN Service Space).

On the platform, ECU software components within a vehicle and services out of a vehicle can work together. Once the service process is invoked by SPM, DSS calls the out-vehicle services in the service process using the service link protocol like SOAP, REST, or DSP (DARWIN Service link Protocol) which is a specific service link protocol. In parallel, DSS calls the ECU software components in-vehicle, which is specified in the service process, using AUTOSAR VFB protocol via AUTOSAR VFB adapter

B. ECU Software Components

The software on the ECU consists of the several software components which are called ECU software component (ECU SWC). ECU SWC includes an application component like ACC (Adaptive Cruise Control) application, a sensor component like the input signal processing of speed sensor, and an actuator component.

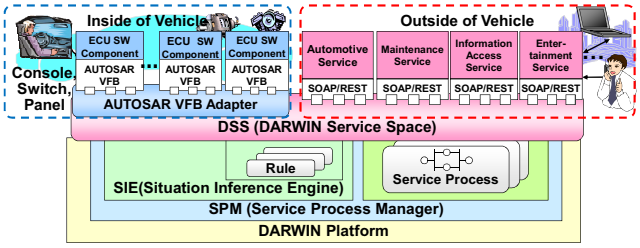

Fig. 2 DARWIN Architecture

In automotive industry, the model of ECU software components has been standardized by AUTOSAR [3]. According to AUTOSAR, application engineer is able to design ECU software components called AUTOSAR software component (AUTOSAR SWC) using AUTOSAR software component template presented in UML. In addition, AUTOSAR SWCs are able to be distributed on several different ECUs over in-vehicle networks using AUTOSAR VFB. Therefore, ECU SWCs are able to interact with each other independent of underlying platform.

C. Services

The DARWIN Architecture enables dealing with a variety of Web services and information from out-vehicle providers, for instance, Web services, messaging services, Audio and Video stream broadcasting, internet shopping, digital electric devices networking, navigation service, parking guide and other automotive specific services.

Even if the Web-based infrastructure is available in service-oriented systems, these kinds of out-vehicle service provisioning and integration may be difficult for service providers when service providers try to develop the integrated services between out-vehicle services and in-vehicle software components.

D. DSS: DARWIN Service Space

There are some technologies like Java Space to share the information and services across the different platforms. However, the existing technologies are not suitable for ACSS, because they are not able to meet the safety, reliability and real-timeness for automotive applications. Therefore, we developed a new shared space model for information and services for automotive, which is called DSS (DARWIN Service Space), illustrated in Fig. 3. All services and information in the DARWIN systems are integrated via the DSS. The DSS includes the QoS management mechanism which performs the services by means of priority order assignment.

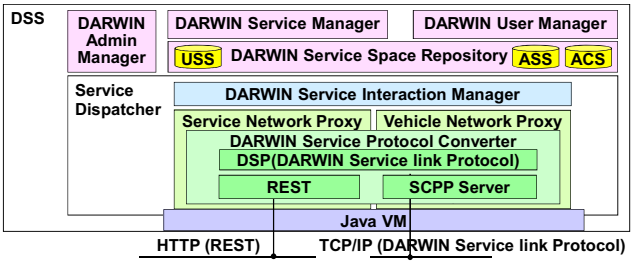

Fig. 3 Architecture of DSS (DARWIN Service Space)

The DSS is based on the publish/subscribe architecture employing the synchronization decoupling[8]. Thus, publishers are not blocked while producing events and subscribers can get asynchronously notified of the occurrence of an event while performing some concurrent activity. The DSS is distributed and implemented on several

devices and servers using repository. The DSS is encapsulated with AUTOSAR VFB adapter to translate the protocol between AUTOSAR VFB and DSS service bus.

E. DSP: DARWIN Service Link Protocol

In DARWIN, two types of service link protocols are supported. One is the REST (REpresentational State Transfer)[16] and the other is DSP (Darwin Service link Protocol). Both protocols play the important role for service interactions and integration in the DARWIN architecture.

(1) REST: We employed REST for service interaction with cloud computing (DSS inside) and Web servers[6].

We are able to operate all resources using only four Web APIs including GET, PUT, POST and DELETE.

However, REST does not support the presentation format of URL to manage automotive specific information on such as user (Driver/Passenger), vehicle, ECU services and out-vehicle services. Therefore we extended REST to specify user ID, car ID, service ID and interface ID in URL. It allows DSS to identify resources in automotive ECUs. For instance, ECU service call interface is specified as following:

```
http://darwin.jp/api/001/vehicle/{car ID}/services/{service ID}/
interfaces/{interface ID}/call
```

(2) DSP: The DSP is a subset of NGTP and is compatible with NGTP protocols [14]. In order to use service interaction between vehicles and cloud computing centers, DSP improves real-time performance capability by data compression technique of data coding.

F. SPM: Service Process Manager

(1) Architecture of SPM

SPM is a subsystem which manages service processes. As illustrated in Fig. 4, it consists of the following two components:

1) Rule repository: it specifies the conditions of service process invocation depending on driving situation, and,

2) SIE (Situation Inference Engine): SPM identifies the driving situation and looks up an appropriate service by retrieves the rule repository, and then invoke the services looked up automatically.

We employed the rule-based inference method. A service provider can specify invocation rules independent of the driving condition into the repository in advance. However, sophisticated automotive services require to integrating multiple services depending on the driving conditions. Therefore, the inference engine helps to look up appropriate services, and invoke them in a certain manner.

Fig. 5 illustrates an example of PNS (Parking Navigation Service) comprising multiple services on different platforms. In PNS, the car, service provider and mobile phone work together to provide parking navigation, remote security and road pricing. Within the DSS, there are multiple autonomous services of possibly invoked. DSS looks up necessary services along with the driving and assessed for suitability by the SIE.

The service orchestration model can be specified by BPEL. However, we extended the model for specifying invoking conditions such as the Start Condition used by the SIE. For example, in the SIE, "*Process start Condition*" is described in the follow manner.

Example of "*Process Start Condition*":

```
When(SearchingParkingLotMode ∈
modeOf(ParkingNavigationServiceProcess)),
Who(userOf(ParkingNavigationServiceProcess) ∈
customersOf("TimesParkingCompany"))
```

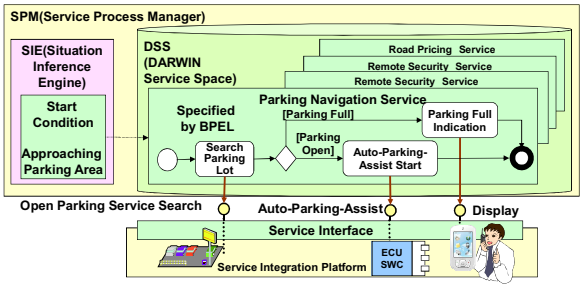

Fig. 4 Service Process Invocation by SPM

(2) Service Process Description

Process of PNS in Fig. 5 is translated to BPEL[17]. This service process in Fig. 5 presents how the PNS performs while collaborating with external Web service and Mobile terminal applications. It assumes that this process is pre-designed and registered into DSS with the corresponding rule repository by a service provider. In case of this rule, a process is started when a driver is searching a parking lot and a customer of Times Parking Company. While driving, if this rule is formed, SIE invokes the process of Parking Navigation Service.

(3) BPEL Description: The Parking Navigation Service

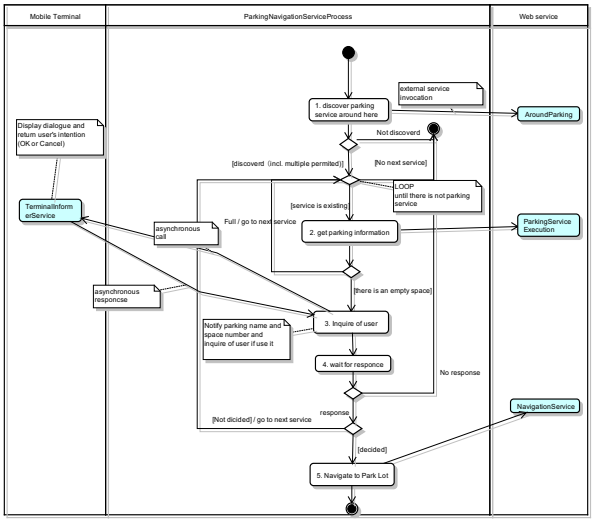

Fig. 5 Process of PNS (Parking Navigation Service)

Process is described by BPEL consists of three sections; pre-definition for performing process, <eventHandlers> and <sequence>. For example, <eventHandlers> section deals with variety of events such as wait event caused while interacting between mobile terminal, Web service and vehicle. It calls a sequence specified in <sequence> section including each collaboration services such as "AroundParking" in Web service.

VI. Ovewview of Prototype And Evaluation

We discuss a case study for proof-of-concept of DARWIN architecture. We evaluated the feasibility of the DARWIN architecture by running usage scenarios on the prototype system. We demonstrate functionalities and evaluate performance of DARWIN from the following two aspects.

(1) Performance of DSP: We measure performance capability of protocol itself by comparison of those of SOAP and REST in a specific evaluation environment.

(2) Real vehicle prototype and its performance evaluation: The case is aimed to estimate system level performance of this architecture. Overhead of a sequence of main service call is measured with above two scenarios in prototype system.

VII. Prototype And Its Evaluation

Fig. 6 illustrates the performance evaluation environment for DSP. SOAP and REST was implemented for the virtual service call. The environment is implemented with Java.

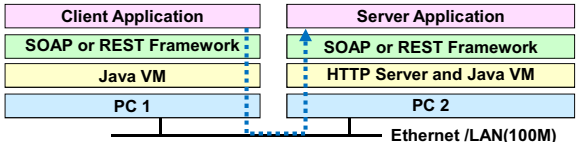

Fig. 6 Configuration of Evaluation Environment

Table 1 summarizes specifications of the environment. The end-to-end processing time from client application to server application, from just after service call request until just before the service execution, and the payload of communication data were measured for comparing performance of SOAP and REST.

For the performance evaluation, the API parameter in XML is defined by referring NGTP parameters, and assuming the minimum parameter set.

Table 1 Specification for Prototype

		PC1 (Client)	PC2 (Server)
CPU		Intel Pentium M 1.70GHz	Intel CoreDuo 1.83GHz
Memory		1.0 GB	1.5 GB
OS		Linux: Ubuntu 9.0.4 (2.6.28-generic)	
Middle ware	SOAP	Axis 1.4.1	
	REST	Restlet2.5	
	HTTP	-	Apache Tomcat 6.0.2
Java VM		Sun Java1.6.0_14	

The statistics of performance evaluation of service link protocol for SOAP and REST are illustrated in Fig. 7 and Fig. 8. Fig. 7 presents the message processing time and Fig. 8 presents the payload.

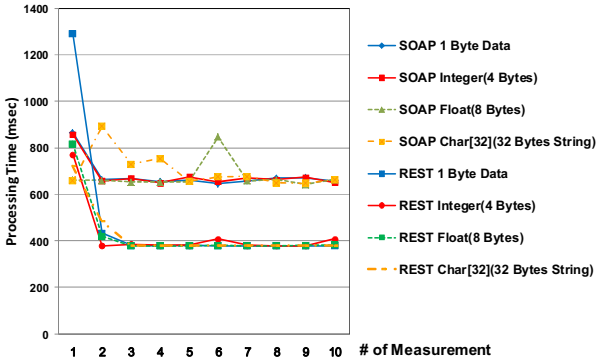

Fig. 7 Message Processing Time for Service Call

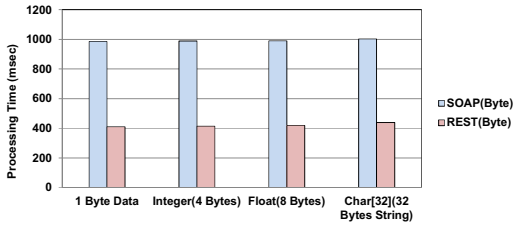

Fig. 8 Payload of Communication Data for Service Call

As the result of experiment illustrated in Fig. 8, there was some 40% increase in performance in REST over SOAP. The API compatibility between SOAP and REST was taken into consideration, and data of unneeded vehicles and services were included in the REST parameters. However, in REST, the additional information can be eliminated from the request parameters, and the performance may be further improved.

When handing one integer parameter, REST also had an advantage in terms of the payload of communication data as illustrated in Fig. 8. Furthermore, REST can handle the calling parameters in binary, and further reduction of communication data may be possible.

VIII. Real Vehicle Prototype And Its Evaluation

Fig. 9 (left) shows an electronic vehicle passenger car in which we implemented a DARWIN prototype. The car is in a retrospective body commemorating the 60th anniversary of DENSO CORPORATION. Fig 9 (right) shows a smart phone with Android.

Fig. 9 Prototype DENSO Electronic Vehicle (left) and Android Smart Phone for Remote Control (right)

A. Prototype System

The configuration of the prototype system is illustrated in Fig 10. The whole system consists of two subsystems; in-vehicle subsystem and out-vehicle one. The specifications of components in Fig. 10 are summarized in Table 2.

(1) In-vehicle subsystem: It consists of a DARWIN gateway, two ECUs, a Light actuator and a Battery sensor. The DARWIN Gateway is responsible for the exchange of the messages between in-vehicle services and out-vehicle services. For DARWIN gateway, DSS and other DARWIN Basic Software (DARWIN BSW) are developed.

(2) Out-vehicle subsystem: It consists of the DSS server emulated for cloud computing center and the Mobile terminal. An Android mobile phone is used for the mobile terminal which a driver can handle. Android phone is able to access ECUs via the DSS server and DARWIN gateway through GUI and applications.

In addition, the DSS includes AUTOSAR adapter. And other DARWIN BSWs are deployed in the DSS server. It allows us to perform the service process which orchestrates in-vehicle functions and out-vehicle services.

SPM is implemented as a stub. A commercial embedded database is employed for a rule repository. SPM is responsible for recovery and transaction process when communication blackout is occurred.

Table 2 Specification of Prototype System

Device	CPU	Memory	Software
D1: Mobile Terminal	QUALCOMM MSM7201a, 528MHz	Flash 512MB, SRAM 192MB	Android R1.6
D2: DSS Server	Intel Core DUO2 3GHz	2.5G	Debian5.0, SQLite, Sun Java Ver. 1.6.0_14, Apache Tomcat Ver. 6.0.20
D3: DARWIN Gateway	VIA C7 1GHz	1GB	Debian4.0 Linter
D4: ECU1	CUSTOM CPU	512KB	AUTOSAR Basic Software Release 3.0
D5: ECU2			

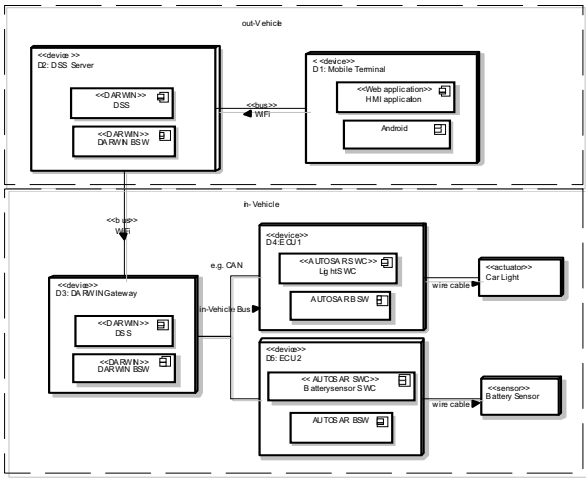

Fig. 10 Configuration of Prototype System

B. Usage Scenarios

Two usage scenarios below were performed for evaluation.

***Scenario 1*: Car light remote control**
Service Type: Calling in-vehicle service from out-vehicle service
Service Description: A user remotely controls car light on/off from the mobile terminal (Android phone).

***Scenario 2*: Battery charging monitor**
Service type: Providing the vehicle information from in-vehicle to out-vehicle
Service Description: A user remotely monitors Battery sensor information in-vehicle from the mobile terminal (Android phone)

(1) Scenario 1
The sequence of scenario 1 is illustrated in Fig 11. Once HMI App in Mobile Terminal (D1) detects the push event of "Car Light" (E1) by a user, it calls "Car Light" service process (M1). The URI of this call API is described as following.

http://darwin.jp/api/001/vehicle/PRI0001/services/SwcLoc/interfaces/MainLight/call

In the URL above, PRI0001 is "Car ID", SwcLoc is "Service ID", and MainLight is "Interface ID" as described in Section 5.5.

(2) Scenario 2
The sequence of scenario 2 is illustrated in Fig 12. Similar to scenario 1, "Battery Sensor Information" service process is invoked and performed by DSS and DSP.

C. Performance of Service Call in Prototype System

We present the performance statistics of DARWIN prototype in Table 3 and Fig. 13. The measurement time periods, T1 to T3, are defined in the sequence diagrams in Fig. 11 and Fig. 12.

(1) Time T1 is the reading time from service or application from outside vehicle (in this case, HMI application

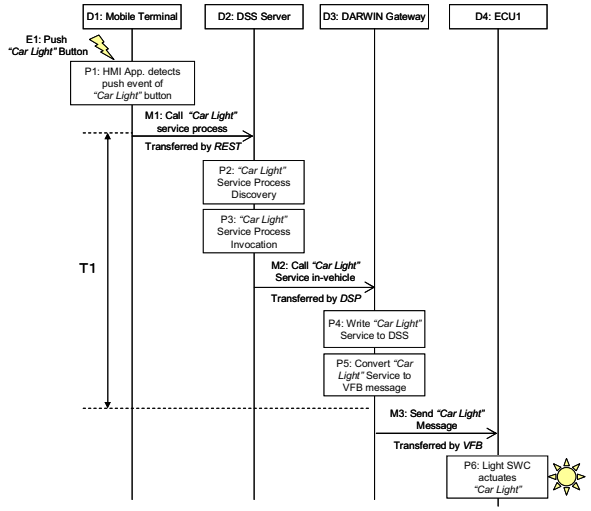

Fig. 11 Remote Control of Car Light

from an android cell phone) to in-vehicle service (in this case, a lighting service from ECU1). So the overhead is the total time of the DSS server searching and reading the lighting service, writing a request to the DSS of DARWIN Gateway, and VFB message conversion in DARWIN platform.

(2) Time T2 is the reading time of the vehicle information in an in-vehicle service (in this case, a battery sensor monitor from ECU2). So the overhead is the total time of data conversion and writing to DSS from DARWIN Gateway, and writing to DSS from DSS server.

(3) Time T3 is the response time to call the battery status, public on the DSS Server. Normally, the service is called periodically (in this evaluation, once in 600ms) within the response time. The service search time of DSS inside the DSS Server is included.

In the evaluation, overhead in service collaboration is around the targeted 500ms, and proves that the service collaboration function with this architecture is viable for actual use. However, viewing the distribution and standard deviation from Table 4, it can be seen that (a) in the first trial, all T1, T2, T3 have large latency. (b) Dispersion (standard deviation) of T2 is very high.

By analyzing detailed data, the following reasons are assumed;

(a) Long response time in the first measurement: The initial loading of the middleware, from disc to memory may be consuming time.

(b) The high dispersion (standard deviation) in T2:

The main reason is that, there is a high dispersion in performance (writing time) of the DSS server. In this prototype system DSS, both DARWIN Gateway and DARWIN DSS exist, and DARWIN Gateway is implemented using Linter Embedded DB. However, on the DSS Server side, SQLite, a popular freeware used in business is used.

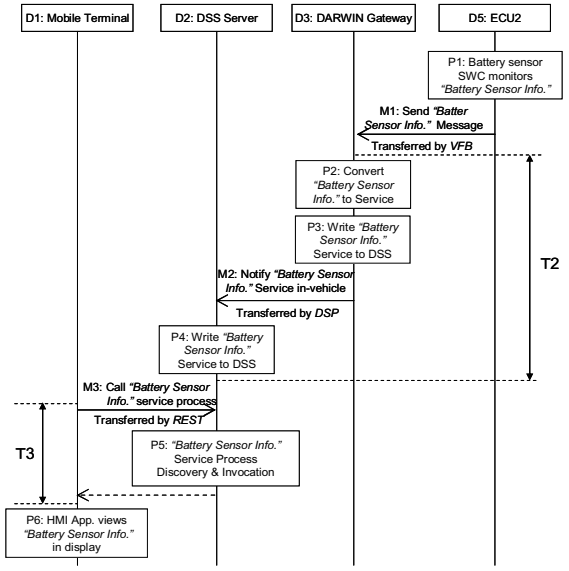

Fig. 12 Battery Charging Monitor

This means, that in an embedded DB, writing time is short and stable, but SQLite is long and unstable. In case of T2, when compared to T1, there is writing on the DSS Server, causing dispersion in the writing time of SQLite, and as a result, dispersion time in the measurement results.

Therefore, this time, improvement in the initial loading time, and exchanging of the server database to high-performance commercial software, can reduce the maximum and dispersion time.

Also, the evaluation result in this phase is based on result from simple scenarios, and further verification with practical scenarios is needed.

Table 3 End-to-End Response Time of Service Call

Scenario	Min.	Max.	Ave.	STD
T1	139	544	162.8[205.1]	15.9[120.7]
T2	91	524	192.9[212.4]	131.1[135.4]
T3	105	433	259.3[266.6]	93.2[90.3]

Note: [] indicates the statistics including the first attempt.

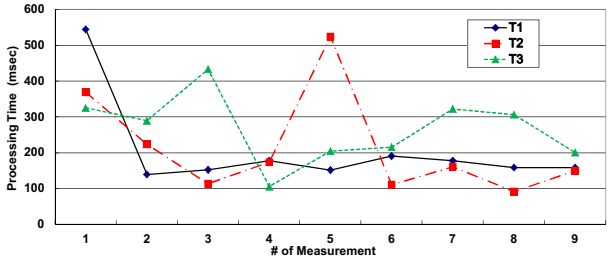

Fig. 13 Propagation Time in Prototype System

IX. DISCUSSIONS

In this case study, we have conformed if these scenarios behaved as we expected in this prototype system based on our DARWIN concept.

(1) Feasibility of DARWIN Architecture: We believe the major contribution of this study is to prove the concept of SOA-based virtual service platform connecting seamlessly between in-vehicle and out-vehicle software platforms, while assuring real-time performance and reliability. The core technologies of DARWIN architecture include DSS, SPM and SIE together with DSP and support for SOAP and REST. Each technology has been implemented in a certain domains. However, orchestrating those technologies, while extending the technologies for adapting to automotive software systems, is challenge. This study demonstrates the feasibility of orchestrating those technologies in the automotive domain, which is considered as a highly complicated and safety-critical distributed embedded software system. Currently, many ECUs in a vehicle are developed independently by different vendors. However, more and more functionalities are required to collaborate across multiple ECUs. Furthermore,

automotive software is expected to collaborate with outside services and social infrastructure in order to further reduce fuel consumption and emissions, as well as to enhance capability in information society. This study contributes to drive the way of software development and provisioning of automotive software into the next stage of software for the era of cloud computing and green society.

(2) Performance of Prototype: A series of performance evaluation on the DARWIN prototype running on passenger car with two usage scenarios demonstrated the behavioral characteristics of DARWIN architecture. Although we need further study, the performance of prototype is good enough to convince the feasibility of the architecture.

(3) Comparison with Related Works: Comparing with previous work [1]2] and related works [5][6][7], we can claim that the following two advantages of DARWIN architecture.

   1) Seamless integration between in-and-out vehicle services.
   2) Evolutional collaboration with conventional architecture such as AUTOSAR [4].

(4) From the experiments, we are convinced that our DARWIN concept is feasible and effective. More evaluations are underway.

X. FUTURE WORKS

We will study the following real-time performance, and verification of safety and reliability. Moreover, we plan to study the detail mechanisms of context-aware lookup and invocation services for SPM.

(1) Improvement of Real-time Performance: The performance improvement of DSP and DSS should be addressed. We plan to reduce network traffic by grid computing including ECU in-vehicle on cloud computing environment [15].

(2) Verification for Safety and Reliability: Verification of safety and reliability should be addressed. In particular, verification of security issues such as prevention from invalid access from out-vehicle and protection of privacy will be needed. In addition, the verification methodology of non-deterministic services will be studied.

(3) SIE for autonomous Automotive service: For SIE, we will study the mechanism of context-aware lookup and invocation of services. We consider a machine learning based approach might be promising other than a static rule-based approach in this study.

XI. CONCLISIONS

In this article, we first discuss the challenges of our concept of ACSS. We proposed DARWIN architecture for ACSS based on SOA, and validated the feasibility of the architecture with case studies. DSS is core technology in DARWIN architecture, which is responsible for service orchestration and information sharing over in-out-vehicle networks. This technology makes it easy for service provider to develop new automotive cloud services.

This study on DARWIN contributes to reveal key aspects of new software architecture for the next generation automotive software, which needs to integrate software in a vehicle and cloud services out of vehicles seamlessly. We will further study non-functional requirements of safety, reliability and security for ACSS and make ACSS real to the development of automotive software systems.

REFERENCES

[1] A. A. Ahson and M. Ilyas (eds.), *Cloud Computing and Software Services*, CRC Press, 2010.

[2] M. Aoyama, et al., Attribute-Based Architecture Patterns for Lightweight Service-Oriented Architectures, *Proc. of APSEC 2009*, IEEE Computer Society, Dec. 2009, pp. 119-126.

[3] AUTOSAR, http://www.autosar.org/.

[4] L. Bocchi, et al., Service-Oriented Modelling of Automotive Systems; *Proc. of IEEE COMPSAC '08*, IEEE, Jul./Aug. 2008, pp. 1059-1064.

[5] M. Broy, et al. (eds.), Automotive Software- Connected Services in Mobile Networks, LNCS Vol. 4147, Springer, 2006.

[6] J. H. Christensen, Using RESTful Web Services and Cloud Computing to Create Nest Generation Mobile Applications, *Proc. of ACM OOPSLA 2009*, Oct. 2009, pp. 627-634.

[7] T. Erl, *Service-Oriented Architecture*, Prentice Hall, 2005.

[8] P. Th. Eugster, et al., The Many Faces of Publish/Subscribe, *ACM Computing Survey*, Vol. 35, No. 2, Jun. 2003, pp. 114-131.

[9] GENIVI, http://genivi.org/.

[10] ITU, *The Fully Networked Car Workshop 2010*, Mar. 2010 http://www.itu.int/ITU-T/worksem/ict-auto/201003/.

[11] A. Iwai, et al., Experiences with Automotive Service Modeling, *Proc. of 10th Workshop on Domain-Specific Modeling, ACM SPLASH 2010,* Oct. 2010, 6 pages.

[12] D. Jordan and J. Evdemon (eds.), Web Services Business Process Execution Language Version 2.0, OASIS, Apr. 2007.

[13] I. H. Krűger, et al., Service-Based Software Development for Automotive Applications, *Proc. of Convergence 2004*, No. 2004-21-040, CTEA/SAE, Oct. 2004, 9 pages.

[14] NGTP, *Application Services Layer*, Version 1.0, Jan. 2008, http://www.ngtp.org/.

[15] OSGi, *OSGi Service Platform Core Specification*, Release 4, Version 4.2, Jun. 2009, http://www.osgi.org/.

[16] L. Richardson, et al., *RESTful Web Services*, O'Reilly, 2007.

[17] S. Weerawarana, et al., *Web Services Platform Architecture,* Prentice Hall PTR, 2005.

[18] M. Wirsing, et al., *A Systematic Approach to Developing Service-Oriented Systems*, SENSORIA White Paper, Oct. 2007.