Service-Oriented Architecture for Automotive Cloud Service Systems

Akihito Iwai
Software Tech. R&D Dept.
Corporate ePF Division
DENSO CORPORATION
Outline

• Background: Automotive Cloud Service Systems (ACSS)
• Key Challenges for ACSS
• Related Works
• ACSS Architecture
• Case Study
• Discussions
• Conclusions and Future Work
Background: Automotive Cloud Service Systems (ACSS)

- Automotive Software Systems become more large-scale and complex year by year
- ⇒ Evolving to “Automotive Cloud Service Systems: ACSS”
  - Vehicle + “Traffic Infrastructure”
  - + “IT service”
  - + “Power Grid”
  - ................

ITS (Intelligent Transport System)/Telematics Services
Cloud Services

Malicious Attack
Invalid Access Protection
Auto Update
Mobile Devices

Other Vehicles
Traffic Infrastructure
Home Network
Example of Service: Intelligent Parking Service

- Car, service provider and mobile phone work collaboratively to provide parking navigation, remote security and road pricing.
- Car provides appropriate services according to the requirements of the situation autonomously.
Key Challenges for ACSS

• **Guarantee of Real-time Performance**
  – The automotive software system is a real-time embedded system which is imposed strict time constrains
    • signal processing of Sensor/Actuator within the determined time interval

• **Assurance of Safety and Reliability**
  – The automotive software system is a safety-critical system which can be fatal to human life
    • Electronic Control Unit (ECU) needs high safety and reliability

• **Embracing the Short Product Life Cycle**
  – The product cycle of the computing devices in-and-out of vehicle is much shorter than that of the automotive devices
    • Automotive service-related standardizations have not matured to practical use yet
      – NGTP and GENIVE for telematics systems
Related Works

- **SOA: Service-Oriented Architecture**
  - Architecture to integrate the information services and communication services seamlessly
  - Dealing with all information as "Everything is a Service"

- **AUTOSAR (Automotive Open System Architecture)**
  - Open and standardized automotive software architecture
  - **Component based** software design model

![AUTOSAR Architecture Diagram]

- **VFB view**
- **Mapping**
- **Tool supporting deployment of SW components**
- **System Constraint Description**

**ECU Hardware**

**Microcontroller Abstraction**

**Basic Software**
- Transfer layers for different communication technologies (e.g. CAN, LIN, ...)
- Network management
- System services (diagnostic protocols, ...) 
- NVRAM management
- ...

**Operating System**

**AUTOSAR RTE**

**AUTOSAR Interface**

**Virtual Functional Bus**

**SW-C Description**

**RTE Basic Software**

**Gateway**
ACSS Architecture: Requirements to ACSS Architecture

• Service Orchestration in-and-out of Vehicle
  – To collaborate between ECU Software Components and external services in safe and within time constraints
  – To optimize service processes for a variety of user requirements and the vehicle context
  – To continue to run even under unstable network connection
    • momentarily communication or power blackout

• Information Sharing over Networks
  – To provide service interfaces for sharing information
  – To remove and modify information with safe after purchased

• Non-Functional Requirements
  – To keep fault tolerant against unpredictable failures
  – Low cost for in-vehicle terminal
ACSS Architecture: What is DARWIN Architecture?

- DARWIN platform contains two main software modules
  - DSS (DAWIN Service Space)
  - SPM (Service Process Manager)
- Provide service/component Integration/Interoperability
  - ECU Software Components (ECU SWC)
  - Services
  - AUTOSAR VFB Adapter included in DSS
  - Service Link Protocol (see the details later)
**ACSS Architecture: DSS(DARWIN Service Space)**

**Challenges**
- Meet the requirements on the high safety, reliability and real-time performance for automotive applications
  - Exiting technologies such as Java Space, CORBA are not met

**Key features**
- **Cloud based DSS server**
  - Encapsulation of interaction among Real Cars
- **AUTOSAR VFB support**
  - ECU Software components communication with other software components or in-out-vehicle services
- **QoS management mechanism**
  - Prioritized messaging service
Detailed Software Architecture of DSS in Vehicle

- DARWIN Admin Manager
- DARWIN Service Manager
- DARWIN User Manager
- DARWIN Service Space Repository
  - USS
  - ASS
  - ACS
- Service Dispatcher
  - DARWIN Service Interaction Manager
- Service Network Proxy
- Vehicle Network Proxy
- DARWIN Service Protocol Converter
  - DSP (DARWIN Service link Protocol)
  - REST
- SCPP Server
- Java VM
- HTTP (REST)
- TCP/IP (DARWIN Service link Protocol)
• Two kinds of service link protocols are supported
  – For communication between Datacenter and in-vehicles
    • DSP (*Darwin Service Link Protocol*)
      – Original Protocol
      – Compatible with subset of NGTP (*New Generation Telematics Protocol*)
      – High performance and lightweight protocol by data compression
  – For communication between Datacenter and out-vehicle services
    • REST (*REpresentational State Transfer*)
      – Lightweight vs. SOAP
      – HTTP base, resource identified by URL
      – Simple basic API: GET, PUT, POST and DELETE
      – Extension for management of in-out-vehicle service base on REST
        » Car ID, Service ID, Interface ID

URL description example:
http://darwin.jp/api/001/vehicle/{car ID}/services/{service ID}/interfaces/{interface ID}/call
REST API structure

- Vehicle
  - VehicleId

- In-Vehicle Service
  - ServiceId

- Service IF
  - InterfaceId

1. Service Property IF
   - Property Value
     - PropertyValue
2. Service Method IF
   - Parameter Value Type
     - ParameterValueType
3. Service Method IF
   - ReturnValue Type
     - ReturnValueType
## Example of DARWIN API

<table>
<thead>
<tr>
<th>ServiceName</th>
<th>Service Id</th>
<th>API</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Application Service</td>
<td>AppDevice</td>
<td>setMainLight</td>
<td>Control front main light.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setFogLight</td>
<td>Control front fog light.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getMainLightStatus</td>
<td>Collect current main light status.</td>
</tr>
<tr>
<td>Auto driving Service</td>
<td>AppAuto</td>
<td>setAutoDrive Mode</td>
<td>Set automatic driving mode. (e.g. Auto-Parking)</td>
</tr>
<tr>
<td>Probe Information Service</td>
<td>AppProb</td>
<td>getSpeed</td>
<td>Collect current speed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getBattery</td>
<td>Collect remain battery capacity.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getBatteryOutput</td>
<td>Collect battery output voltage.</td>
</tr>
<tr>
<td></td>
<td>AppSon</td>
<td>getClearanceSensors</td>
<td>Collect vehicle clearance conditions.</td>
</tr>
</tbody>
</table>
Performance Evaluation

Configuration of Evaluation Environment

PC1

Client Application

SOAP or REST Framework

Java VM

PC2

Server Application

SOAP or REST Framework

HTTP Server

Java VM

Ethernet /LAN(100M)
# Specification on performance evaluation

<table>
<thead>
<tr>
<th></th>
<th>PC1 (Client)</th>
<th>PC2 (Server)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>Intel Pentium M 1.70GHz</td>
<td>Intel CoreDuo 1.83GHz</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>1.0 GB</td>
<td>1.5 GB</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>Linux: Ubuntu 9.0.4 (2.6.28-generic)</td>
<td>Linux: Ubuntu 9.0.4 (2.6.28-generic)</td>
</tr>
<tr>
<td><strong>Middleware</strong></td>
<td><a href="#">SOAP</a> Axis 1.4.1</td>
<td>Axis 1.4.1</td>
</tr>
<tr>
<td></td>
<td><a href="#">REST</a> Restlet2.5</td>
<td>Restlet2.5</td>
</tr>
<tr>
<td></td>
<td><a href="#">HTTP</a> -</td>
<td>Apache Tomcat 6.0.2</td>
</tr>
<tr>
<td><strong>Java VM</strong></td>
<td>Sun Java1.6.0_14</td>
<td>Sun Java1.6.0_14</td>
</tr>
<tr>
<td>#</td>
<td>Field</td>
<td>Size*</td>
</tr>
<tr>
<td>----</td>
<td>-------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>MESSAGE_VERSION</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>MESSAGE_ID</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>MESSAGE_TYPE</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>SERVICE_CTRL_SIZE</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>BODY_FIELD_SIZE</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>TIME</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>SOURCE_USER_ID</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>SOURCE_CAR_ID</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>TARGET_SERVICE_ID</td>
<td>7</td>
</tr>
<tr>
<td>11</td>
<td>TARGET_SERVICE_INTERFACE_ID</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>DATA_FIELD</td>
<td>Variable</td>
</tr>
</tbody>
</table>

*Note: Bytes
This information is the exclusive property of DENSO CORPORATION. Without their consent, it may not be reproduced or given to third parties.
Payload of Communication Data for Service Call

- SOAP (Byte)
- REST (Byte)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>SOAP (Bytes)</th>
<th>REST (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Byte Data</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Integer (4 Bytes)</td>
<td>800</td>
<td>800</td>
</tr>
<tr>
<td>Float (8 Bytes)</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>Char[32] (32 Bytes)</td>
<td>400</td>
<td>400</td>
</tr>
</tbody>
</table>

This information is the exclusive property of DENSO CORPORATION. Without their consent, it may not be reproduced or given to third parties.
• Situation Matching
  – Car moves through various situations.
  – Service Integration platform executes appropriate services according to the requirements of the situation autonomously.

Situation = Space (Where) & Time (When)
ACSS Architecture: SPM (Service Process Manager)

- Easy Integration of services using Service Interface
Attempt to minimally extend BPEL

Example of BPEL description

(1)  
<invokeAbstractService when="always" where="area:Osaka"  
what="Search parking" execute="all" timing="start">  
<params>  
   <param type="int">latitude</param>  
   <param type="int">longitude</param>  
</params>  
<return type="string">ParkingServiceName</return>  
</invokeAbstractService>

(2)  
<invoke name="InvokeNotifyEmptySpaceNumber"  
partnerLink="ParkingServer" operation="GetEmptySpaceNumber"  
portType="GetEmptySpaceNumberPT"  
inputVariable="ParkingServiceName"  
outputVariable="ParkingNumber">  
</invoke>

(3)  
<invoke name="InvokeCheckParkingCar" partnerLink="CAR"  
operation="CheckParkingCar" portType="CheckParkingCarPT"  
inputVariable="ParkingNumber" outputVariable="bParkCar">  
<toParts>  
   <toPart part="partnerLinkName" toVariable="ParkingServiceName"/>
   <toPart part="partnerLinkName" toVariable="ParkingNumber"/>
</toParts>
</invoke>
1. discover parking service around here
2. get parking information
3. Inquire of user
4. wait for response
5. Navigate to Park Lot

Display dialogue and return user's intention (OK or Cancel)
Notify parking name and space number and inquire of user if use it
Not discovered / go to next service

Not discovered / go to next service
Loop until there is not parking service
Full / go to next service

[service is existing]
[discovered (incl. multiple permitted)]
[there is an empty space]

[decided]
[Not decided] / go to next service

External service invocation
AroundParking
ParkingServiceExecution
NavigationService
<?xml version="1.0" encoding="UTF-8"?>
<process

/* Define name space, import file, desired service and etc */
name="IntelligentParkingService"
targetNamespace="http://enterprise.netbeans.org/bpel/IntelligentParkingService/
DefaultServiceName"
xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"

import namespace="http://j2ee.netbeans.org/wsdl/TerminalInformerService"
location="TerminalInformerService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"

/* Specify EventHandler */
<eventHandlers>
<onEvent partnerLink="IPS_PL"
operation="EventOperation"
portType="ns4:IntelligentParkingServicePortType"
variable="Event"
messageType="ns4:IntelligentParkingServiceOperationNotify">

<sequence name="Sequence9">
<assign name="AssignTerminateMsg">

<reply name="ForceTerminateReply"
partnerLink="IPS_PL"
operation="IntelligentParkingServiceOperation"
portType="ns4:IntelligentParkingServicePortType"
variable="IntelligentParkingServiceOperationOutTerminate"/>
</sequence>
</onEvent>
</eventHandlers>

/* Specify Sequence */
<sequence>
<receive name="Start"
createInstance="yes"
partnerLink="IPS_PL"
operation="IntelligentParkingServiceOperation"
xmlns:tns="http://j2ee.netbeans.org/wsdl/IntelligentParkingService"
portType="tns:IntelligentParkingServicePortType"
variable="IntelligentParkingServiceOperationIn"/>

</receive>
</sequence>
Case Study: Evaluation of DARWIN with a Prototype

Configuration of Prototype System

Two usage scenarios

**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 using 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 using the mobile terminal (Android Phone).
### Table 1 Specification of Prototype System

<table>
<thead>
<tr>
<th>Device</th>
<th>CPU</th>
<th>Memory</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1: Mobile Terminal</td>
<td>QUAL COMM MSM7201a, 528MHz</td>
<td>Flash 512MB, SRAM 192MB</td>
<td>Android R1.6</td>
</tr>
<tr>
<td>D2: DSS Server</td>
<td>Intel Core DUO2 3GHz</td>
<td>2.5G</td>
<td>Debian5.0 Sun Java Ver. 1.6.0_14 Apache Tomcat Ver. 6.0.20, SQLite</td>
</tr>
<tr>
<td>D3: DARWIN Gateway</td>
<td>VIA C7 1GHz</td>
<td>1GB</td>
<td>Debian4.0 Linter</td>
</tr>
<tr>
<td>D4: ECU1</td>
<td>CUSTOM CPU</td>
<td>512KB</td>
<td>AUTOSAR Basic Software Release 3.0</td>
</tr>
<tr>
<td>D5: ECU2</td>
<td>CUSTOM CPU</td>
<td>512KB</td>
<td>AUTOSAR Basic Software Release 3.0</td>
</tr>
</tbody>
</table>
Case Study: Scenario 1: Car light remote control

**Scenario 1:** Car light remote control

**D1: Mobile Terminal**

**D2: DSS Server**

**D3: DARWIN Gateway**

**D4: ECU1**

**M1: Call “Car Light” service process Transferred by REST**

**P2: “Car Light” Service Process Discovery**

**P3: “Car Light” Service Process Invocation**

**M2: Call “Car Light” Service in-vehicle Transferred by DSP**

**P4: Write “Car Light” Service to DSS**

**P5: Convert “Car Light” Service to VFB message**

**M3: Send “Car Light” Message Transferred by VFB**

**P6: Light SWC actuates “Car Light”**

**E1: Push “Car Light” Button**

**P1: HMI App. detects push event of “Car Light” button**

**“Car Light” API:**

http://darwin.jp/api/001/vehicle/PRI0001/services/SwcLoc/interfaces/MainLight/call
Case Study: Scenario 2: Battery charging monitor

D1: Mobile Terminal
D2: DSS Server
D3: DARWIN Gateway
D5: ECU2

P1: Battery sensor SWC monitors “Battery Sensor Info.”

M1: Send “Battery Sensor Info.” Message

P2: Convert “Battery Sensor Info.” to Service

P3: Write “Battery Sensor Info.” Service to DSS

M2: Notify “Battery Sensor Info.” Service in-vehicle

P4: Write “Battery Sensor Info.” Service to DSS

M3: Call “Battery Sensor Info.” Service to DSS

P5: “Battery Sensor Info.” Service Process

Discovery & Invocation

P6: HMI App. views “Battery Sensor Info.” in display

© DENSO CORPORATION All rights reserved.

This information is the exclusive property of DENSO CORPORATION. Without their consent, it may not be reproduced or given to third parties.
Performance of service call in prototype system

Table 4  End-to-End Response Time of Service Call

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Min.</th>
<th>Max.</th>
<th>Ave.</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>139</td>
<td>544</td>
<td>162.8[205.1]</td>
<td>15.9[120.7]</td>
</tr>
<tr>
<td>T2</td>
<td>91</td>
<td>524</td>
<td>192.9[212.4]</td>
<td>131.1[135.4]</td>
</tr>
<tr>
<td>T3</td>
<td>105</td>
<td>433</td>
<td>259.3[266.6]</td>
<td>93.2[90.3]</td>
</tr>
</tbody>
</table>

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

Fig. 15  Propagation Time in Prototype System
Discussions

- **Two advantages of DARWIN architecture**

  1. **Seamless integration between in-vehicle and out-vehicle services**
     - DSS supports integration by standard service messaging protocols such as SOAP and REST
     - SPM provides a mechanism how to integrate and manage different service processes seamlessly

  2. **Evolutional collaboration with conventional architecture such as AUTOSAR component architecture**
     - DSS works as a service bus like ESB (Enterprise Service Bus), enabling to coordinate protocols and interfaces
     - To collaborate with existing software components through AUTOSAR VFB
Conclusions and Future Work

• Conclusions
  – Challenges of ACSS
    • Challenges of our concept of ACSS (Automotive Cloud Service System) is discussed
  – DARWIN Architecture and its Proof of Concept
    • We propose SOA (Service-Oriented Architecture) for ACSS named DARWIN
    • DSS (DARWIN Service Space) makes it easy for service provider to develop new automotive cloud services
    • Validated the DARWIN architecture with a prototype and two case studies
  – Contribution of our ACSS Concept
    • New automotive software platform based on SOA and Cloud Computing
    • Enabling automotive to work with greener society such as smart grid

• Future Work
  – Remaining some challenges: non-functionality aspect, safety-critical issues like QoS and dependability
  – Make ACSS more usable to the development of automotive software systems
Acknowledgements

• Special thanks for external collaborators:
  – Norio Oohashi, NEC Corporation
  – Osamu Takahashi, Eiwa System Management, Inc.
  – Teruyuki Nakazawa, DENSO CREATE Inc.
  – Steven Kelley, Metacase
  – Prof. Aoyama, Nanzan University

• Additional reference:
Thank you for your attention!

Prototype DENSO Electronic Vehicle
for commemorating the 60th anniversary of DENSO CORPORATION