Internet of Things Platform and Services for Connected Cars

Chungki Woo[1,2], Jihyun Jung[2,3], Jang Euitack[2], Jongwoong Lee[2], Jaewook Kwon[1] and Daeyoung Kim[2,4]
[1]*Samsung Electronics, Suwon, Kyungki, Korea, Republic of*
[2]*Graduate School of Software, KAIST, Seoul, Korea, Republic of*
[3]*Mando, Pankyo, Kyungki, Korea, Republic of*
[4]*Auto-ID Lab, KAIST, Daejeon, Chungbuk, Korea, Republic of*

Keywords: Connected Car, IOT, GS1, EPCglobal, Oliot, EPCIS, ONS.

Abstract: In recent years, the connected car market has been expanding. Various car manufacturers are trying to provide Internet of things (IoT) services by collecting and analysing sensing data from cars. However, there is not a well-defined standardized IoT platform to handle the big data for the various car OEM companies or service providers. To resolve this issue, we propose a globally standardized IoT platform for connected cars based on Global Standard 1 (GS1). We extended and remodelled Electronic Product Code global (EPCglobal), one of GS1 standards, and developed a new IoT platform framework called open-language for IoT (Oliot). Then, based on the framework, we modelled car events and developed some hardware and software modules to capture, store, and share the event data. We also implemented demonstration services using the shared data for verification. This research can provide a new ecosystem to the connected car industries and service providers to enable standardized handling and processing of big data. As a result, it will be much easier to create and provide a greater variety of services and combinations of services.

1 INTRODUCTION

The global connected car market is constantly expanding. One study has predicted (Allied Market Research, 2014) that the market will grow to $141 billion by the year 2020.

A ‘connected car’ usually means a car that supports connection and communication between the car and the outside world (Internet, mobile devices, other cars, drivers and other things) using wireless network technologies.

With growth of the market, the number of related IoT services is also increasing. Such as BMW and Volkswagen, are trying to collect and analyse the sensing data from cars and provide various services. BMW provides a car-as-a-sensor (CARASSO) service and dynamically updated map information to drivers. (Investor's Business Daily, 2015)

Volkswagen informs drivers of real-time traffic conditions and car status through Car-Net. (Engadget, 2015) HP has also experimented with such a service by sensing driver behaviour, road quality, and social media in the World Record Race.

Through these efforts, manufacturers are introducing new value into the car market.

In this situation, a problem is that a number of service providers and car OEM companies are building and using their own private data silos. In other words, they are using heterogeneous platforms and different protocols for collecting, processing, storing, and sharing car data. Having such different ways to handle data is a major obstacle to data sharing and to making good services using the big data from connected cars and various things from other domains.

To address this issue, we developed new common IoT platform for Connected Car based on Global Standards 1 (GS1) (Global Standards 1).

For a long time, GS1 was a global common business language that has been used for distribution business. The GS1 provides a few standards having three following abilities. Identify, Capture and Share. These three abilities enable industries to uniquely identify object and capture and share the event in life-cycle of object. We applied this concept into new IoT platform for connected car. In the conclusion, we can expect that generated event data in the life cycles of cars can be standardized, collected, and shared easily.

Woo, C., Jung, J., Euitack, J., Lee, J., Kwon, J. and Kim, D.
Internet of Things Platform and Services for Connected Cars.
DOI: 10.5220/0005952904690478
In *Proceedings of the International Conference on Internet of Things and Big Data (IoTBD 2016)*, pages 469-478
ISBN: 978-989-758-183-0

To make the platform, we modelled four car events, namely, 'Selling', 'Driving', ‘Repairing’ and 'Scrapping', and we constructed a server following concept of GS1 to store and share the event data. After that, we developed a special hardware module for capturing the events and implemented a capturing application running on the hardware. In addition, we developed the demonstration services called the car data \management system (CDMS) and car data monitoring in real-time mobile application service (CDMR) for the purpose of verification.

In this paper, we explain how to the connected car platform was built and how the services were designed in detail.

The rest of this paper is organized as follows. In Section 2, Oliot IoT platform framework including a detailed description of GS1. Section 3 describes the design of the connected car platform in detail. Section 4 describes the development of services. Finally, Section 5 concludes this paper.

2 GS1 & OLIOT

2.1 GS1: Global Standards 1

Global Standards 1 (GS1) supports the identification of objects, capturing and sharing of data for business related to products, services, assets, shipments, physical locations, and so on. GS1 standards have three important abilities.

First, IDENTIFY is related to identification using unique codes called GS1 identification keys.

Second, CAPTURE is related to data capture which support bar code and radio frequency identification (RFID) data carriers and specify consistent interfaces. Software components that connect the data carriers to business applications are also included here.

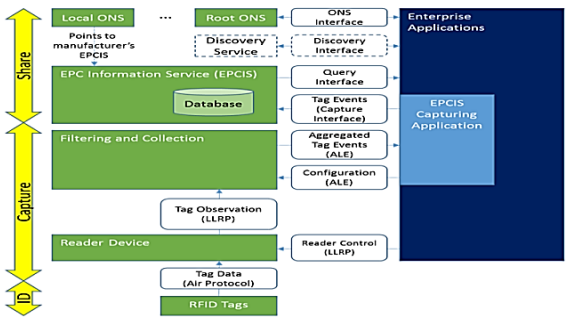

Figure 1: GS1 EPCglobal Architecture (Traub, 2005).

Third, SHARE is related to information sharing that supports standards for data, communication, and discovery.

Figure 1 shows the architecture of GS1 EPCglobal and three steps (IDENTIFY, CAPTURE and SHARE) from bottom to up.

GS1 EPCglobal (Traub, 2005) is one of the GS1 standards and for using RFID-based identification. It also supports the sharing of information about the status of products and global visibility of items through EPCIS.

EPC Information Service (EPCIS) (GS1, 2007) is for capturing and sharing events. This is enabled by code systems, such as GS1 identification keys, global product classification (GPC) and Electronics Product Code(EPC). These universal identifiers provide a unique identity for any physical object in the world. The identification of objects is done by reading RFID tags. The data from objects is filtered and grouped by an event capturing application. Finally, the EPCIS-level events are transferred to an EPCIS server and stored and shared.

Object Name Server (ONS) (Mealling, 2004) is for finding and sharing data which are related with specific object.

GS1 Rail Vehicle example (GS1, 2014) shows the importance and strength of GS1. The example explains how to implement EPCIS for Rail Vehicle visibility. The example shows that the event data from a rail vehicle can be effectively captured and stored into the EPCIS.

We thought that making a new standardized IoT platform using GS1 would be very powerful, so we extended and remodelled the GS1 standard as an IoT platform. We finally developed an open-source IoT platform framework called open-language for IoT (OLIOT), and we constructed a connected car platform based on it.

2.2 Oliot: Open Language for IoT

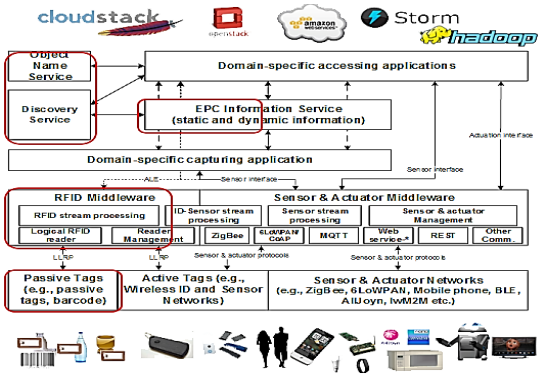

Figure 2: Oliot IOT platform architecuture (Oliot 2015).

Oliot (Oliot, 2015) is an open-source project to build an ID-based framework to identify, capture, control, and share information about smart things.

The overall architecture of the Oliot IOT platform is shown in Figure 2.

It was implemented by referencing the newest standard of GS1. We tried to extend the GS1 concept to a standardized IoT platform framework. As a result, Oliot extended and remodeled the basic architecture of the GS1 EPCglobal standard.

The EPCIS of EPCglobal was extended and handled as 'Internet things global data repository' called Oliot Object Name Service (Oliot-EPCIS) (Byun and Kim 2015) complying with the GS1 standard EPCglobal and supporting various sensor devices not only RFID.

Also, ONS of EPCglobal was extended as 'Internet things service discovery' called Oliot Object Name Service (Oliot-ONS).

In Oliot, various sensors can be used, unlike GS1 EPCglobal, which can only use RFID tags. These include passive tags, such as barcodes; active tags, such as wireless IDs; as well as sensor and actuator networks, such as Zigbee, LoWPAN, mobile phone, BLE, and so on.

The event data from sensing devices can be processed and delivered to the Oliot-EPCIS system by sensor middleware and domain-specific capturing applications which developed with domain knowledge. A domain-specific accessing application can access the information stored in the Oliot-EPCIS and provide services to end-users. Finally, end-users can find the supported services for specific things using Oliot-ONS. The Oliot-ONS server has task of providing a service list regarding the request of an end-user with a thing identification key. Thus, the end-user can receive a service list and get the services.

Oliot makes it easy to construct IoT services. Our connected car platform was also built based on this platform framework. The collected event data from a car using this platform can be shared and used very easily for various service privoders.

3 CONSTRUCT PLATFORM

Figure 3 shows that simple overall platform architecture based on Oliot framework and the data stream from a car to Oliot-EPCIS. When event data is generated from a car, the sensing hardware module receives the event data. The data is processed and sent to the Oliot-EPCIS server using the HTTP POST protocol by the event capturing application. Oliot-EPCIS stores the event data and shares it. The following subsections explain each component in detail.

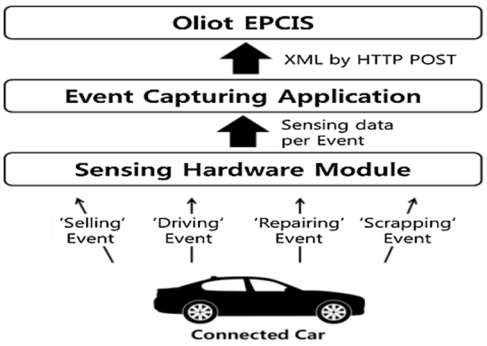

Figure 3: Overall Architecture of the Proposed Platform.

3.1 Modelling Car Events

Modelling car events follows the format of Oliot-EPCIS event type.

To describe an event formally, we describe it with 'WHEN', 'WHAT', 'WHERE', 'WHY' of the event. 'WHEN' means the timestamp at which the event occurred. 'WHAT' means objects related to the event. 'WHERE' means the location at which the event happened. 'WHY' explains the business context of the event.

What events can be generated for a car? We imagined the following scenario.

"A car is sold in a retail shop and moved. The car is driven by its owner and repaired when the car has a physical problem. Finally, the car will be old and scrapped after some time."

Hence, we defined four events in the lifecycle of a car, namely,

'Selling', 'Driving', 'Repairing', 'Scrapping'

Table 1: Data for car events.

NO.	Event	Data
1	Selling	From location to location
2	Driving	Speed
3		Longitude
4		Latitude
5		Altitude
6		RPM
7		Brake ON/OFF
8		The degree of steering wheel
9		The degree of accelerator
10	Repairing	IDs of repaired components
11	Scrapping	IDs of output components

We decided to sense all 11 types of data related to these events. Table 1 shows the four events and the data generated by each event.

Tables 2 to 5 in each show detailed descriptions of the four events.

Table 2: TransactionEvent for car 'Selling' event.

EPCIS event	Event type	TransactionEvent	
	Action	ADD	
WHEN	eventTime	2015-11-20 13:06:23.999 +09:00	
WHAT	epcList	4012345.077889.1111 (SGTIN)	
WHERE	bizLocation	0614141.07346.1235 (SGLN)	
	geo extension	37.4836,127.044 (GPS)	
WHY	bizStep	retail_selling (CBV)	
	dispostion	sold (CBV)	
	Extension	VIN number	KMHJF32J7TU123456
		sourceList	urn:epc:id:sgln:4012345.00001.0
		destinationList	urn:epc:id:sgln:4012345.00002.0

Table 2 is the description of 'Selling' event for the below example situation.

"A car (WHAT-epcList) was sold (WHY-disposition) and moved from one location (WHY-Extenson-sourceList) to another location (WHY-Extension-destinationList) in the retail shop (WHERE-bizLocation) at the moment (WHEN-eventTime)."

In the table, the type of event appears in the first row in each table. Each event is classified as one of the three event types following Oliot-EPCIS event type.

'ObjectEvent','TransactionEvent','TransformationEvent'

In the case of a 'Selling' event, the event type is 'ObjectEvent'.

The second row gives a description of the action. Three actions were already defined in the EPC event definition.

'ADD', 'DELETE', 'OBSERVE'

The action of a 'Selling' event is 'ADD'. We also should note that SGLN (Serialized Global Location Number) and SGTIN (Serialized Global Trade Item Number) which are the 'GS1 standard identification keys (GTIN or GLN) + unique serial number' of the car are used to identify the car and its location. This is charm point about using GS1 concept. We can use GS1 code system for identification of things.

'Extension' is additionally defined here. This is an extension of the basic EPCIS event and describes extended data. The first vehicle information number (VIN number) is always included to be used by various applications. After that, in this event, the location to and from which the sold car moved are described in WHY-Extension-sourceList, WHY-Extension-destList in the table.

One more important thing is that the 'WHY-bizStep', 'WHY-disposition' are described by core business vocabulary (CBV) (GS1, 2014).

The detailed meaning of event types, actions, and how to describe events are officially given in the GS1 EPCIS standard (GS1, 2007).

Table 3: ObjectEvent for car 'Driving' event.

EPCIS event	Event type	ObjectEvent	
	Action	OBSERVE	
WHEN	eventTime	2015-11-20 13:06:23.999 +09:00	
WHAT	epcList	4012345.077889.1111 (SGTIN)	
WHERE	bizLocation	0614141.07346.1235 (SGLN)	
	geo extension	37.4836,127.044 (GPS)	
WHY	bizStep	driving (User Defined Vocabulary)	
	Extension	VIN number	KMHJF32J7TU123456
		Speed	2
		Longitude	127.044
		Latitude	37.4836
		Altitude	45.3
		RPM	785
		Brake ON/OFF	on
		The degree of steering wheel	-540
		The degree of accelerator	0

Table 3 gives the description of an ObjectEvent for 'Driving' and it explains the following example event.

"A car (WHAT-epcList) is being driven (WHY-bizStep) at the moment (WHEN-eventTime) and its current speed, position, and car status are like this. (WHY/Extension-Speed, Longitude, Latitude, Altitude, RPM,Brake ON/OFF, The degree of Steering Wheel, The degree of accelerator)."

We should note that the WHY-bizStep is not described by CBV here. We defined a new word (user-defined vocabulary)'driving' and used it.

Table 4: TransactionEvent for car ‘Repairing’ event.

EPCIS event	Event type	TransactionEvent	
	Action	ADD	
WHEN	eventTime	2015-11-20 13:06:23.999 +09:00	
WHAT	epcList	4012345.077889.1111 (SGTIN)	
WHERE	bizLocation	0614141.07346.1235 (SGLN)	
	geo extension	37.4836,127.044 (GPS)	
WHY	bizStep	repairing (CBV)	
	Extension	VIN number	KMHJF32J7TU123456
		repairedList	urn:epc:id:sgtin:4012345.077889.21 urn:epc:id:sgtin:4012345.077889.22 urn:epc:id:sgtin:4012345.077889.23

Table 4 gives the description of a TransactionEvent for ‘Repairing’, and it explains following example event.

“A car (WHAT-epcList) was repaired (WHY-bizStep) at the moment WHEN-eventTime), and the repaired components are these things (WHY/Extension-repairedList). The location of the repair shop is here (WHERE-bizlocation)”.

Table 5: Transformation for car ‘Scrapping’ event.

EPCIS event	Event type	TransformationEvent	
	Action	DELETE	
WHEN	eventTime	2015-11-20 13:06:23.999 +09:00	
WHAT	inputEPCList	4012345.077889.1111 (SGTIN)	
	outputEPC List	4012345.077889.18 (SGTIN) 4012345.077890.19 (SGTIN) 4012345.077891.20 (SGTIN)	
WHERE	bizLocation	0614141.07346.1235 (SGLN)	
	geo extension	37.4836,127.044 (GPS)	
WHY	bizStep	destroying (CBV)	
	Extension	VIN number	KMHJF32J7TU123456

Table 5 is the description of a TransformationEvent for ‘Scrapping’, and it explains following example event.

“In the junkyard (WHERE-bizLocation), a car (WHAT-epcList) was scrapped and destroyed (WHY-bizStep) at the 2moment (WHEN-eventTime), and the output components are these things (Extention-outputEPCList).”

We can model not only above four events but any events using this description mechanism.

3.2 Sensing Car Events

After modelling and describing car events, we considered how to obtain event data from a car. For ‘Driving’ events, we decided to sense the data using a self-produced hardware module. It virtually emulates a connected car on an ordinary car with wireless communication. Here, we explain how the hardware was designed to obtain event data.

Figure 4: OBD (II) Interface (Top Left), OBD Cable (Top Right), Raspberrypi2 (Middle Left), NEO-6M GPS Module (Middle Right), SKPANGS PICAN board (Down Left), WIFI Module (Down Right).

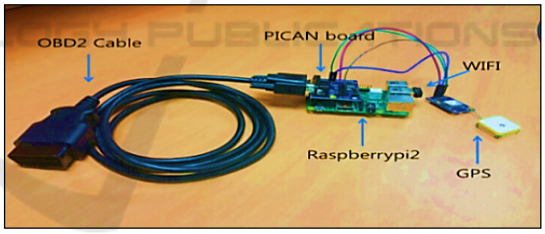

Figure 5: Combined Hardware Module.

First of all, a car was needed that would support an on-board diagnostics (OBD) interface. An OBD interface is normally used to check the car status which is data usually called car area network (CAN) data from the electronic control unit (ECU) network in the car. We could get four types of event data, namely, RPM, brake on/off status, the degree of the steering wheel, and the degree of the accelerator. To get data from OBD interface, we needed a special cable called an OBD cable. In addition, we used a GPS sensor to get the speed, longitude, latitude, and altitude of car. When a ‘Driving’ event happens while the car is being driven, all eight events (No. 2~ No. 9 in Table 1) can be received from the OBD interface and GPS sensor. All data is collected and

processed by an application running on a Raspberrypi2. The Raspberrypi2 functions as the central processing unit in the hardware composition. Also, we used a PICAN board (PICAN CAN-Bus) to receive data from the OBD interface. A PICAN board is specialized developed add-on hardware for the Raspberrypi2. Figure 5 shows the completed hardware configuration for the project. Table 6 summarizes the arranged hardware and its purpose.

Table 6: Hardware list.

No.	Hardware	Purpose
1	Car	Generate car events
2	Raspberry pi2	Run capturing application and send formatted data to EPCIS server using network
3	OBD2 Cable	Get car status data from OBD interface of car
4	GPS sensor module	Get GPS data
5	PICAN board	Get car status data through OBD cable
6	WIFI module	Support wireless communication of Rasberrypi2
7	WiFi Access point	Communication from the car to the external Oliot-EPCIS server. (Here, just mobile hotspot is used.)

Figure 6 shows the hardware installed in the car. Power for the hardware environment is supplied by the car.

Figure 6: Combined hardware (TOP) , Regular Car - KIA Sorrento (DOWN LEFT), Installed Hardware in the car (DOWN RIGHT).

Figure 7 shows the overall data stream with interfaces from the car to the Rasberrypi2.

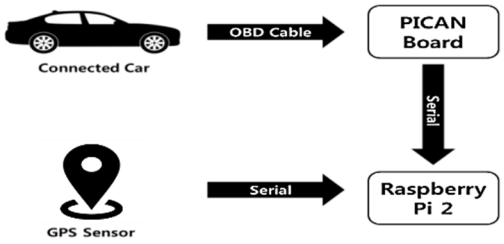

Figure 7: Event data flow.

Finally, collected and processed data from the Raspberry pi2 is sent to the Oliot-EPCIS server presented in section 2.2. In this case, wireless communication is used. A WiFi module and a WiFi access point are necessary for that.

3.3 Building Oliot-EPCIS

The Oliot-EPCIS stores and shares the data. It supports web service interfaces and RESTapi for the capture and query services, so we can store the sensed data in the EPCIS and share it using the interfaces.

To store event data in the EPCIS server, the schema of the database (mongoDB) should be defined first. The basic schema for a basic event, called an 'EPCIS event', was already pre-defined, but if new extended data for a specific domain (such as a connected car) should be stored, another extended schema is necessary. The schema was saved in the XML .XSD file. For example, the schema for 'Driving' event is like below Figure 8. All eight data and data types of 'Driving' event are specified.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="Cargs1_driving"
    targetNamespace="http://tempuri.org/Cargs1_driving.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/Cargs1_driving.xsd"
    xmlns:mstns="http://tempuri.org/Cargs1_driving.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name ="VinNumber">
    <xs:simpleType>
      <xs:restriction base="xs:string"></xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name ="PositionSensorLat">
    <xs:simpleType>
      <xs:restriction base="xs:double"></xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name ="PositionSensorLng">
    <xs:simpleType>
      <xs:restriction base="xs:double"></xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name ="PositionSensorAlt">
    <xs:simpleType>
      <xs:restriction base="xs:double"></xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name ="SpeedSensor">
    <xs:simpleType>
      <xs:restriction base="xs:double"></xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name ="RPMSensor">
    <xs:simpleType>
      <xs:restriction base="xs:double"></xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name ="BreakSensor">
    <xs:simpleType>
      <xs:restriction base="xs:boolean"></xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name ="AccelPedal">
    <xs:simpleType>
      <xs:restriction base="xs:double"></xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name ="SteeringWheel">
    <xs:simpleType>
      <xs:restriction base="xs:double"></xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
<!-- Cargs1_driving.xsd -->
```

Figure 8: XML schema (.XSD) for car 'Driving' event data.

In case of ‘Selling’ and ‘Scrapping’ events, we do not need an explicit XML schema because the default schema of an EPCIS event already includes the necessary schema.

3.4 Event Capturing Application

Modelling events, building the EPCIS server, and defining the schema were necessary, but the most important aspect of this work was to develop an event capturing application to operate the hardware and control the process of collecting and sharing the event data from a car. The event capturing application initializes the hardware module and collects the data (RPM, break, the degree of accelerator, and the degree of steering wheel), GPS data (latitude, longitude, altitude, and speed). After that, it sends the data to the EPCIS server after making an EPCIS XML formatted document. In addition, it periodically (once per second by default) generates virtual events, such as ‘Selling’, ‘Repairing’, and ‘Scrapping’, and also send this data to the EPCIS server.

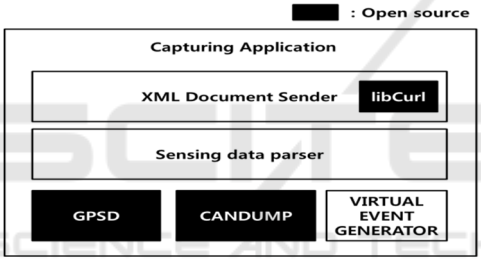

Figure 9: Structure of capturing application.

Figure 9 shows the simple structure of the event capturing application. To obtain GPS data, a modified GPSD (GPS daemon) is used, while to obtain CAN data, a modified CANDUMP utility is used. Both are open-source, and they can be obtained easily from the website.

A ‘Sensing data parser’ is applied to the GPS and CAN sensor data. An XML document sender sends XML documents to the Oliot-EPCIS using the HTTP Post protocol. The ‘libCurl’ is a popular open-source library (Libcurl library, 2015) that provides the HTTP protocol used to send XML documents to EPCIS.

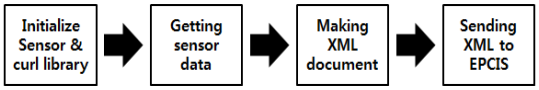

Figure 10: Sequence of capturing application.

Figure 10 shows the processing sequence of the capturing application.

The XML document is sent to the EPCIS server using the HTTP/Post protocol by the event capturing application.

It is important to note that the XML string must be adapted to the XSD schema, which we defined in section 3.3.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE EPCISDocument>
<epcis:EPCISDocument xmlns:epcis="urn:epcglobal:epcis:xsd:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     creationDate="2015-01-03T11:30:47.0Z" schemaVersion="1.1" xmlns:car="DargsI_driving.xsd">
  <EPCISBody>
    <EventList>
      <ObjectEvent>
        <eventTime>2015-11-20T13:05:46.000+09:00</eventTime>
        <eventTimeZoneOffset>+9:00</eventTimeZoneOffset>
        <epcList>
          <epc>urn:epc:id:sgtin:4012345.077889.27</epc>
        </epcList>
        <action>OBSERVE</action>
        <bizStep>urn:epcglobal:cbv:bizstep:driving</bizStep>
        <car:PositionSensorLat>37.4836 </car:PositionSensorLat>
        <car:PositionSensorLng>127.044 </car:PositionSensorLng>
        <car:PositionSensorAlt>45.3 </car:PositionSensorAlt>
        <car:SpeedSensor>2</car:SpeedSensor>
        <car:VinNumber>KMHJF32J7TU123456</car:VinNumber>
        <car:RPMSensor>785</car:RPMSensor>
        <car:BreakSensor>on</car:BreakSensor>
        <car:AccelPedal>0</car:AccelPedal>
        <car:SteeringWheel>30</car:SteeringWheel>
      </ObjectEvent>
    </EventList>
  </EPCISBody>
</epcis:EPCISDocument>
```

Figure 11: XML string to send car ‘Driving’ event data.

Figure 11 shows an example XML document for a ‘Driving’ event. It means that the car is being driven, and the car speed is 2 Km/hour, the car break is on, the degree of accelerator is 0, and the degree of the steering wheel is 30 to the left direction. The current car position is latitude : 37.4836, longitude: 127.044, and altitude : 45.3 at the moment.

4 DEMONSTRATION SERVICES

In the prior sections, the design of the platform to collect connected car event data and share it by Oliot-EPCIS was explained. In this section, we describe the implementation of demo applications based on the platform for verification.

First, the web services for searching and managing car data are explained. Secondly, we present an Android app on a mobile device which obtains car data in real-time.

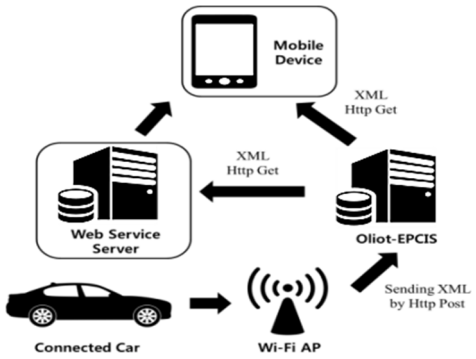

Figure 12: Overall structure of services.

Figure 12 shows the overall structure of the services. The services were implemented on a mobile device and a Web service server.

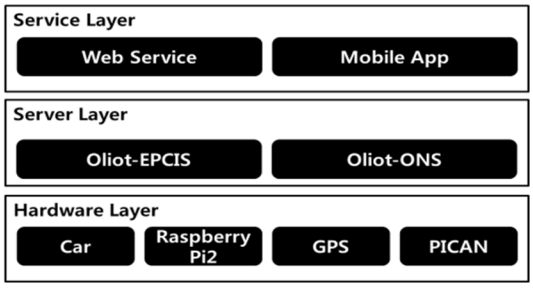

Figure 13: Overall Service.

Figure 13 shows the demo application implemented as a service layer, and others were already mentioned above. A Web service and mobile app query was sent to EPCIS using an Http Request and an XML document was received as a response as shown in Figure 14.

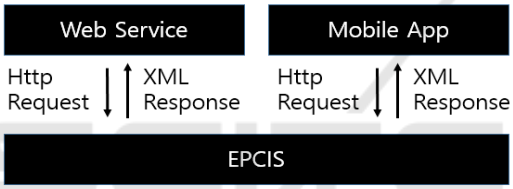

Figure 14: Service operation.

4.1 Car Data Management System (CDMS) Web Services

Figure 15: Main intro of CDMS Web service.

CDMS (Figure 15) provides services to a user to search and manage various car events (Table 1). CDMS provides two searching services: the first is searching 'Driving' history (Table 3); the second is searching the history of a car (Table 2, 4, 5).

The 'Driving' history search service, provides the driving history more specifically by simple web service. The user inputs the related car ID (SGTIN+VIN number) and period into CDMS, and then a related query string is made. Using an Http request, the query is executed, and its response is an XML document as mentioned in section 3. The XML document includes various data types: GPS data, break sensor, and other data (Figure 8). The XML document is parsed, and the result is displayed on a webpage like Figure 16.

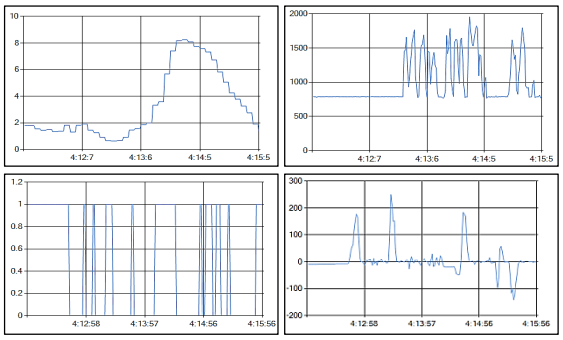

Figure 16: Speed variation (Up Left, Y-axis : speed (Km/hour), X-axis : time), RPM variation (Up Right, Y-axis : RPM, X-axis : time), Break sensor (Down Left, Y-axis : break status - on : 1 otherwise 0, X-axis : time), Steering wheel variation (Down Right, Y-axis : degree of steering wheel, X-axis : time).

RPM, speed, break, and angle of steering wheel data are shown as line graphs. Also, using the Google Maps service, the car's location is displayed like Figure 17.

Figure 17: Car location results (Google Maps).

The Web service is implemented using Active Server Page Extension (ASPX).

Second, the car history search service provides the specific history of the car after the user inputs the related car ID. The user can find out when the car was manufactured, which parts have been replaced, and who has sold the car.

4.2 Car Data Monitoring in Real-time (CDMR) Mobile Application Service

CDMR is a mobile application (Android) that

provides car data in real-time. CDMR queries the Oliot-EPCIS server with a specific car ID every second, and the latest car data is obtained and displayed via a mobile application. The displayed data is related to RPM, speed, location, break sensor, and so on (Table 1). Thus, a user can monitor car data and location in real-time. Also, a user can monitor different cars by entering another car ID. Figure 18 shows the display of real-time car status on a mobile phone.

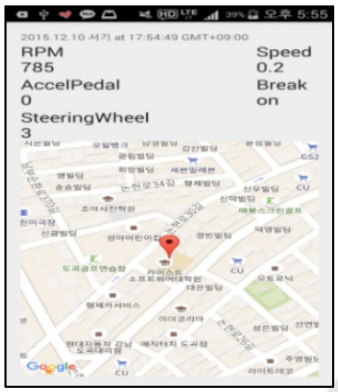

Figure 18: Real-time car position and status in the mobile.

4.3 Building Oliot -ONS

After designing these services, we constructed Oliot-ONS for the end-user to find a list of services, such as CDMS and CDMR of specific car, regardless of time or place. Consumers or other service providers can obtain the service list using only the GS1 identification key of a car. Service lists normally include the address of the EPCIS server, the Web service server, and so on. Figure 19 shows the overall demonstration service architecture for connected cars including Oliot-ONS.

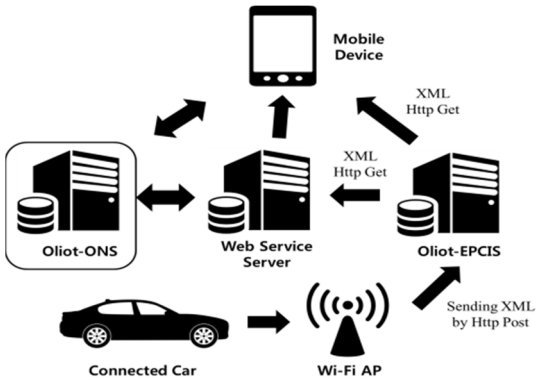

Figure 19: Services with Oliot-ONS.

If someone wants to develop a service based on car event data, they can access the event data shared by the Oliot-EPCIS Server and make and deploy the service, or a previously developed service based on data shared by EPCIS can be used. Oliot-ONS supports Web-based ONS record management. Using ONS API, Web-enabled devices can register, delete, and edit information of main databases. The EPCIS server and a separate service server would be registered in the ONS server in this case. In this paper, this function was not implemented, but it could be easily implemented and used.

5 CONCLUSIONS

In this paper, we proposed a new IoT platform for connected cars based on GS1. We modelled car events and produced a hardware sensing module. In addition, we developed a capturing application to capture and send event data to the Oliot-EPCIS. Finally, the car event data is shared and used with various end-users and service providers. For verification of the feasibility of the system, we implemented CDMS and CDMR services for demonstration.

Any car manufacturers can use this concept and build their platforms to process and share their data through this system, which is based on a globally standardized GS1 mechanism. Through this, all data from connected cars can be shared easily, and service providers can design and deploy IoT services easily. This work represents an innovative new means of processing big data from connected cars. We can easily eliminate the silos and share the data. Finally, this will be a major breakthrough in big data processing for connected cars.

REFERENCES

Allied Market Research, 2014. *"Connected Car Market - Opportunities and Forecasts, 2013 – 2020".*

Byun, J., & Kim, D., 2015. Oliot EPCIS: New EPC information service and challenges towards the Internet of Things. In RFID (RFID), *2015 IEEE International Conference on (pp. 70-77).*

Engadget, 2015. “Keep tabs on your Volkswagen with the Apple Watch".

Global Standards 1, Available from : http://www.gs1.org.

GS1, 2014 *Core Business Vocabulary (CBV) version 1.1.*

GS1, 2015. “EPCIS for Rail Vehicle Visibility Application Standard Release 1.0.1, Ratified" available from : http://www.gs1.org/epcis/rail/latest.

GS1., How GS1 standards work from http://www.gs1.org/how-gs1-standards-work.

Investor's Business Daily, 2015. *"Amazon: Cloud Will Make Internet of Things BetterWeb Service Guides IoT Data New 'AWS IoT' can power apps for online devices".*

Libcurl library, 2015. Available from http://curl.haxx.se/libcurl/

Oliot, 2015. Open Language for Internet of Things. Available from : http://oliot.org.

PICAN CAN-Bus Board for Raspberry Pi. Available from http://skpang.co.uk/catalog/pican-canbus-board-for-raspberry-pi-p-1196.html.

GS1, 2007. *EPC Information Services (EPCIS) Version 1.1 Specification.*

Mealling, M., 2004. *EPCglobal Object Name Service (ONS) 1.0. EPCglobal Working Draft.*

Traub, K., Allgair, G., Barthel, H., Burstein, L., Garrett, J., Hogan, B., ... & Stewart, R. (2005). *The EPCglobal architecture framework. EPCglobal Ratified specification.*