27th ITS World Congress, Hamburg, Germany, 11-15 October 2021

Paper ID #937

DLR V2X Software Framework: A CCU-agnostic solution for transceiving V2X messages

Maximiliano Bottazzi[1], Daniel Wesemeyer[1], Maik Bargmann[1], Sten Ruppe[1]

[1.] German Aerospace Center (DLR),
Institute of Transportation Systems,
Rutherfordstrasse 2, 12489 Berlin,
{maximiliano.bottazzi | daniel.wesemeyer | maik.bargmann | sten.ruppe}@dlr.de

Abstract

The employment of Intelligent Transportation Systems stations for sending and receiving V2X messages in research projects can become an inconvenient task, even more if hardware of different manufacturers is used. The APIs that manufacturers offer for their hardware are often difficult to handle on the user side. Furthermore, the creation and definition of experimental V2X message types or the extension of existing messages can be time-consuming, especially if a team of developers use different programming languages or frameworks. This paper presents the DLR V2X Software Framework (V2X-SF) that aims at solving the named problems by providing an easy-to-use API for V2X end users. First, the system architecture and the design philosophy of the framework are presented followed by an example of the framework's application in a specific project. In the end, a conclusion and an outlook on future works on the framework are given.

Keywords:

ASN.1, CAM, CCU, CPM, DENM, ETSI, ITS, MCM, ObjSys Code generator, OBU, ROS, RSU, SPAT, TransAID, TxRx, V2X, V2X Software Framework, V2X-SF

Introduction

Connected mobility, in the road sector called Vehicle-to-Everything (V2X), will be a vital part of future transportation systems. Among its expected merits are the reduction of traffic congestion and greenhouse emissions by providing infrastructure advice for vehicles, e.g. green wave information at traffic lights, and the enhancement of traffic safety [1]. Regarding automated driving, it can also significantly reduce the amount of data that needs to be collected by sensors of automated vehicles since there are V2X message types that indicate the presence of other vehicles or even vulnerable road users (VRU). [2]

There are two basic communication technologies for V2X, namely ITS-G5 (or WAVE in the United States) and Cellular-V2X (C-V2X). ITS-G5 uses the 802.11p Wi-Fi standard while C-V2X is a 3GPP standard. [3]

The message formats used in V2X are standardized by the European Telecommunications Standardization Institute (ETSI) and SAE International in the United States. Both organizations use the interface description language named Abstract Syntax Notation One (ASN.1) to define the message formats and contents.

To reduce the channel utilization and to ensure that all Intelligent Transportation Systems (ITS) stations understand each other messages are encoded using standard and well-known coding rules aimed to compact its final length. By default, V2X uses the unaligned packed encoding rules (UPER). The encoded messages are transmitted via GeoNetworking protocol, a network layer protocol that allows communication among individual ITS stations and broadcasts in a geographical area. [4]

ITS stations are classified depending on where they are installed as either inside a vehicle (On-Board Unit, OBU) or on the road side infrastructure (Roadside Unit, RSU). There are different manufacturers of ITS stations hardware in the market and each of them provides different application programming interfaces (API) for connecting with their devices. Many of these ITS stations are used in several national and international research projects at the DLR and other research institutes. Basic interoperability is given among these since all stations communicate with each other using the same message formats and the same network layer. But given this variety of ITS hardware and their corresponding APIs, the problem is more on the user side. V2X application developers have to take care of generating valid V2X messages and encoding them using the encoding rules required by the V2X standard and back into a decoded format to process the transmitted information. These tasks can become time-consuming, especially when collaborating in a team that use different programming languages, operating systems or frameworks. The DLR V2X Software Framework (V2X-SF) aims at reducing the effort necessary to develop a running V2X system by providing uncomplicated software tools for composing, validating and transceiving V2X messages.

The outline of the paper is as follows: First, an overview of the system architecture of the V2X-SF is presented. The architecture description is subdivided into a development phase in which the API for specific message types is generated and a runtime phase for the application of the previously generated APIs. In the following, a brief report of the use of the V2X-SF in a European research project of the DLR is given to emphasize certain aspects of the framework's workflow. The paper concludes with an

outlook on future works and possible extensions of the framework.

System architecture

This section describes the software architecture of the V2X-SF and the basic design philosophy of its code. The software architecture is separable into two phases: The first phase describes the generation of the message representations and transceivers for a specific V2X message type (development phase) and the second phase describes the interaction between different application modules (runtime phase).

Concept and design

The V2X-SF is first and foremost meant to be used in research projects for integrating existing V2X messages and for developing and testing of expanded or new V2X messages. It is not meant to be used in final products. The V2X-SF is written in Java, its modules and dependencies are built and managed using Apache Maven.

From an operational viewpoint, the very final output of the framework are programmatically generated applications, so called transceivers (TxRx), a user-friendly API and tools for interacting with them (the so-called TxRx clients). Basically, TxRxs shall hide a part of the complexity of composing, sending and receiving (transceiving) V2X messages, so the user does not need to be an expert in that area. Other secondary features of TxRxs are, among others, logging of incoming and outgoing messages, forwarding of data to a backend administration system and creation of random messages for testing. Several TxRxs can run and interact simultaneously on the same device since they follow the design and functional philosophy of microservices. This means, every module operates independently of the others and executes as specific task, in this case each TxRx is in charge of an individual V2X message type. [5]

Development phase

The V2X-SF development phase describes the steps necessary to generate the software required for transceiving V2X messages: TxRxs (and their Java and ROS clients) and the message representations. For all descriptions in this section see Figure 1. The starting point of the development phase for each V2X message type is the ASN.1 definition, which describes its data structures and value types and ranges. These definitions are usually provided by institutions (e.g. C2C-Consortium, ETSI), project partners or are self-created. They are used in two different processes:

In the first process, Java source code is autogenerated out of the ASN.1 definition using an UPER compiler. UPER compilers are expensive pieces of software and the DLR uses the proprietary *ASN1C ASN.1 Compiler* provided by Objective Systems, Inc.[1]

During the second process, the ASN.1 definition is slightly adapted by adding some metainformation and is processed using the V2X-SF code generator (WinD), a development tool of the V2X-SF. WinD generates message representations and translators for the different programming languages (Java

[1] https://www.obj-sys.com/

classes, C++ structs and ROS[2] messages), software modules for the TxRxs and ready-to-run ROS nodes (ROS TxRx Client).

In both processes, JAR files are automatically built and stored in a Maven repository. Finally, the source code that was autogenerated with the UPER compiler, the message representations and translators are all together combined into a universal interface, transceiver application or, shortly, TxRx.

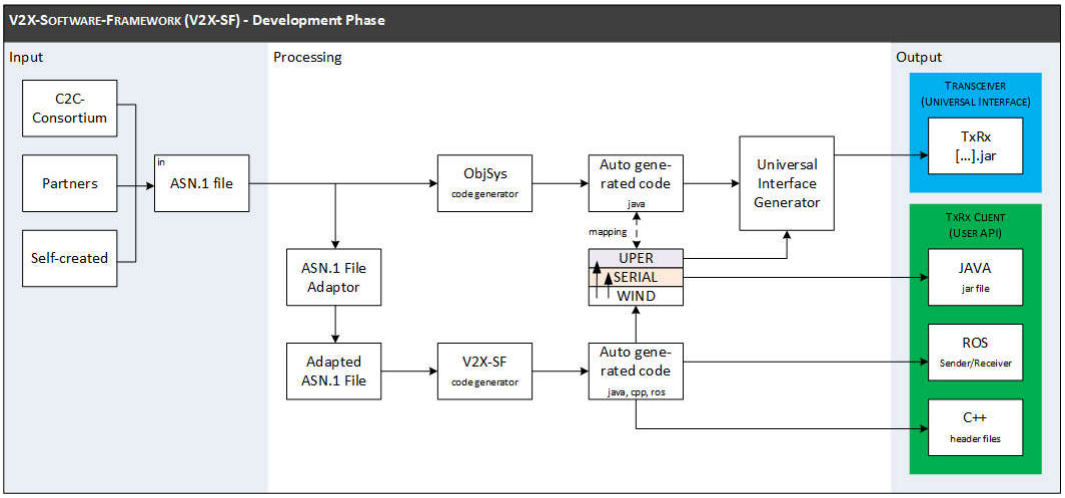

Figure 1: Schematic of the development phase of the V2X Software Framework

Runtime phase

The runtime phase describes the execution and interaction of the transceiver application and other two parties. The concept of this phase is depicted in Figure 2.

In the following, the process of generating and sending a V2X message is described as a use case. Three actors are involved in the runtime phase: a user (or end) application, a TxRx and a communication control unit (CCU). The TxRx acts as a middleware, translator or interface between them and connects both ends via UDP. On the CCU or wireless side, UPER messages are used as data exchange format, and on the API or user application side SERIAL messages. A SERIAL message is a V2X-SF internal message type composed of a 16-byte header and a payload containing the V2X message data, which is encoded with a simple set of encoding rules. This set of encoding rules is based on the standard method for serializing data in C-like systems, it resembles the byte representation obtained through serializing structs, in this case the representation of the message at issue.

[2] ROS: Robot Operating System, see www.ros.org

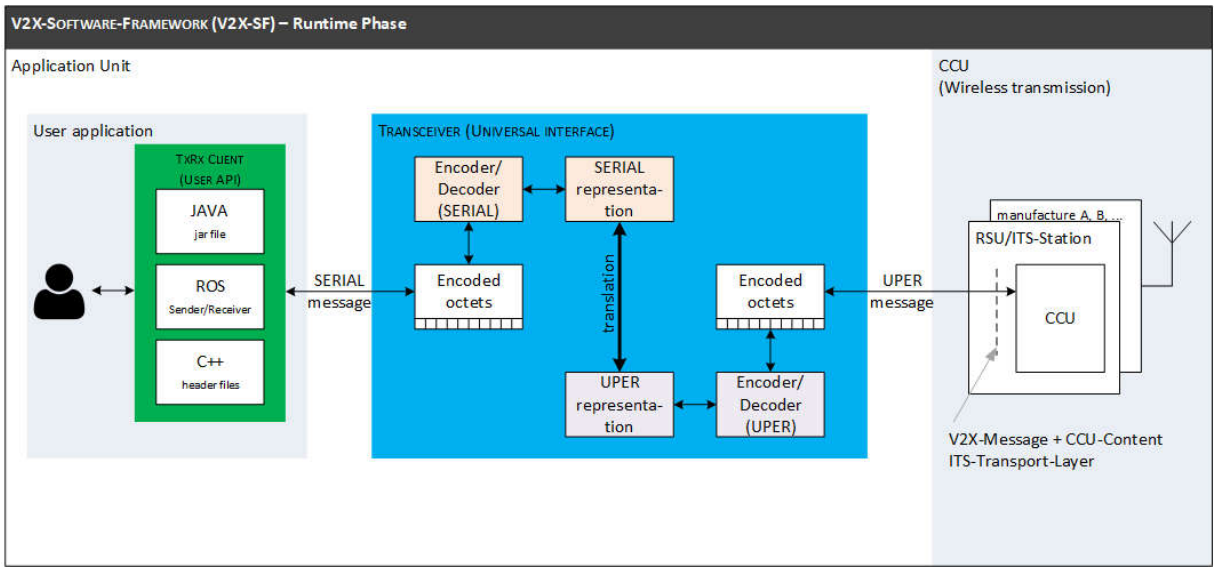

Figure 2: Schematic of the runtime phase of the V2X Software Framework

Depending on which language or platform the user application is implemented, developers can make use of different tools offered by the V2X-SF for creating SERIAL messages and communicating with a TxRx. Java-based applications can make use of the *WinDConnector* library and by using factory methods, empty and valid V2X messages can be composed with ease. This might sound trivial, but even empty messages have specific fields, for example message id and the protocol version, that need to be filled with the correct values and when users are not aware of that, they will fail to create valid messages. Furthermore, getter and setter methods can be used to access and modify message fields according to the use case or scenario one wants to demonstrate. ROS-based applications willing to transceive V2X messages make use of two autogenerated ROS nodes, one for incoming (indication) and another for outgoing (request) messages.

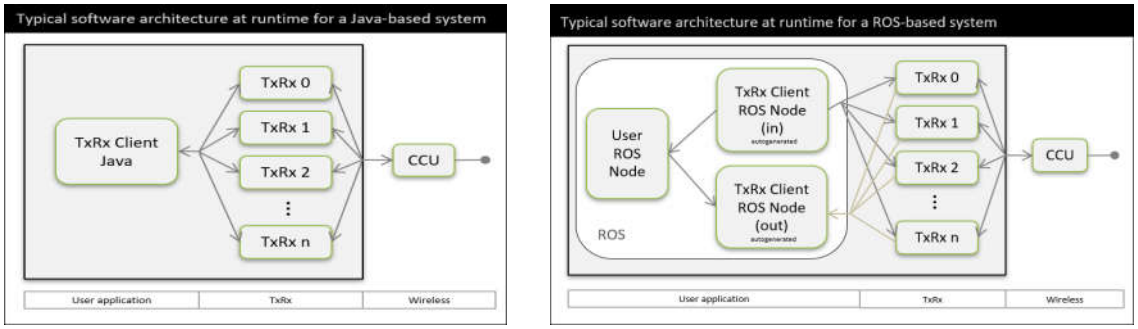

Figure 3: Typical software architecture at runtime (with Java and ROS clients)

Internally, TxRxs translate SERIAL messages into UPER messages and vice versa. This step is one of the benefits of the V2X-SF because the translation of the message bytes between the UPER format and a more accessible and easier format like SERIAL requires knowledge of the encoding rules which not all users or developers necessarily have. As mentioned before, the package format used for exchanging UPER messages with different CCUs may vary depending on the vendor and are normally composed

of a header and a payload. The header contains information required by the GeoNetworking protocol, such as the BTP port or dissemination type and the payload the V2X message encoded in UPER format. The user selects the desired CCU in the configuration file of the TxRx.
When a message shall be sent, the TxRx receives an indication SERIAL message generated by a user application, translates it into an UPER message and forwards it to the CCU through its interface. On the CCU, the message is transmitted wirelessly, either to the receiving station or as a broadcast in a geographic area.

Example application

The V2X software framework is being used in various projects at the DLR. In the following section, its application and extension in the EU research project TransAID [3] (Transition Areas for Infrastructure-Assisted Driving) are presented.

Project aims and implications for the framework

In the early stages of automated driving, the SAE[4] levels of road vehicles and their abilities to communicate via V2X will be very heterogeneous. Furthermore, there will certainly be areas (e.g. urban areas, accident scenes) where highly automated driving cannot be allowed and where automated vehicles have to change their levels of automation. These areas are herein referred to as "transition areas". TransAID aims at developing traffic management procedures and protocols to assist automated vehicles in transitioning from high levels of automation to lower levels or even manual driving. This assistance should increase traffic safety and efficiency, for example by distributing the transitions over a road section instead of triggering all transitions at one specific point.
The project included several simulation studies to assess the impact of these procedures on a dense traffic flow. Additional, real-world studies were performed that included automated driving and infrastructure advice via V2X communication. During these studies, the V2X framework was used extensively for implementing the communication between the automated vehicle (AV) and the road side unit (RSU).
The use cases of the TransAID field tests required several V2X message types to be used, the most basic of which were CAMs[5] [6], MAPs[6] and SPATs[7] [7]. Those message types were already integrated into the framework and could already be used in applications. In order to generate infrastructure advise, new message types had to be included into the framework. The requirement to demonstrate vehicle transitions at accident scenes or construction sites required the use of DENMs[8] to

[3] See https://www.transaid.eu/

[4] Society of Automotive Engineers, see https://www.sae.org/

[5] Cooperative Awareness Message

[6] Map Data

[7] Signal Phase and Timing

[8] Decentralized Environmental Message

warn approaching vehicles of the presence of a blockage. [8] Also, a message was needed to coordinate the driving maneuvers of automated vehicles, for example by giving them lane advise. This was realized through MCMs[9]. [9] Lastly, it was essential to also detect non-connected or parking vehicles. To achieve this, CPMs[10] generated by camera-based object detection were used. [10] The inclusion of CPMs also required the extension of the framework's interfaces, adding an API for the camera's operating system ROS.

Conclusion and outlook

This paper presented the V2X Software Framework, a universal API for flexible and easy implementation of V2X applications. It was reported how the framework could successfully be used in a research project in order to create even very complex use cases with relatively little effort. If an ASN.1 file of a specific V2X message is available, the respective transceivers and user APIs can be auto-generated and included in applications.

The V2X-SF is under active development since several improvements can still be made. For once, the usage of the UPER compiler of ObjSys is problematic since it is very expensive and its functionalities are not modifiable. At the moment, the DLR is evaluating whether to search for an open source alternative or to implement an own compiler. Also, in some use cases a bottleneck on the User interface side can occur since the ROS representation of the V2X messages, depending on their ASN.1 definition, can quickly become quite large. Since ROS is not designed to exchange and process vast amounts of data, the performance could deteriorate rapidly. To date, no considerable issues on that matter were reported.

References

[1] T. Löffler, H.-W. Schaal: Car2X – von der Forschung zur Serienentwicklung. In: Elektronik automotive 12.2012.

[2] L. Hobert, A. Festag, I. Llatser, L. Altomare, F. Visintainer and A. Kovacs, "Enhancements of V2X communication in support of cooperative autonomous driving," in IEEE Communications Magazine, vol. 53, no. 12, pp. 64-70, Dec. 2015, doi: 10.1109/MCOM.2015.7355568.

[3] Z. MacHardy, A. Khan, K. Obana and S. Iwashina, "V2X Access Technologies: Regulation, Research, and Remaining Challenges," in IEEE Communications Surveys & Tutorials, vol. 20, no. 3, pp. 1858-1877, third quarter 2018, doi: 10.1109/COMST.2018.2808444.

[4] ETSI (2017): Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1: Media-Independent Functionality; EN 302 636-4-1, v1.3.1, August

[9] Maneuver Coordination Message

[10] Collective Perception Message

2017.

[5] S. Newman (2015): Building Microservices, 2015, O'Reilly Media.

[6] ETSI (2019a): ITS; Veh Comms; Basic Set of Apps; Part 2: Spec. of Coop. Awareness Basic Service, EN 302 637-2, v1.4.1, 2019.

[7] SAE (2016): Dedicated Short-Range Communications (DSRC) Message Set Dictionary, SAE J2735, 2016.

[8] ETSI (2014): ITS; Veh Comms; Basic Set of Apps; Part 3: Spec. of Decentralized Environ. Notif. Basic Service, EN 302 637-3, v1.2.2, 2014.

[9] ETSI (2020): ITS; Veh Comms; Informative Report for the Maneuver Coordination Service, TR 103 578, v0.0.5, 2020.

ETSI (2019b): ITS; Veh Comm; Basic Set of Apps; Analysis of the Collective Perception Service (CPS), TR. 103 562, v0.0.15, 2019.