Decoding of data on a CAN powertrain network

T. Hermans[(1)], J. Denil[(1,3)], P. De Meulenaere[(1)] and J. Anthonis[(2)]

(1) TERA-Labs, Karel de Grote University College,
Salesianenlaan 30, 2660 Antwerp, Belgium
(2) LMS International, Interleuvenlaan 68, 3001 Leuven, Belgium
(3) LoRE Lab, University of Antwerp,
Middelheimlaan 1, 2020 Antwerpen, Belgium
Email: tim.hermans@kdg.be, joachim.denil@kdg.be,
paul.demeulenaere@kdg.be, jan.anthonis@lmsintl.com

Abstract --- *Contemporary vehicles contain increasingly more electronic modules and electronic networks. Therefore, modifying or testing these vehicles by third parties becomes virtually impossible in case the original vehicle manufacturer does not disclose its technical data. This paper proposes a reverse engineering method to discover to large extent the semantics of CAN messages in a given network. This way, the manipulation of such a CAN network becomes possible. The method is applied to a test vehicle provided by LMS International where the powertrain CAN bus of the vehicle is successfully reverse engineered.*

I. Introduction

Most of the cars that are currently produced contain one or more CAN (Controller Area Network) buses [1]. While these in-vehicle networks undoubtedly contribute to better performance in terms of functionality, safety, ecology, comfort, etc., they also make life of third parties much more difficult since the meaning of the messages on the in-vehicle networks is in most cases not disclosed.

In testing situations, the in-vehicle networks sometimes cause an undesired behaviour in the vehicle. For example when a vehicle is put on a chassis dynamometer, the vehicle may believe it is in a dangerous situation because just two of the four wheels are turning. Therefore, some control unit may decide to adjust or shut down some vehicle functions which results in different driving characteristics.

It is however likely that such adjustments by the control units are unwanted in dedicated test conditions, such as HIL (Hardware-In-the-Loop)-tests. If it would be possible to identify which CAN message contains the command for the shut down or adjustment, this behaviour could be prevented. However, to our knowledge, no method to reverse engineer the semantics of these messages is described in the literature.

In this paper, a method to discover the semantics of CAN data messages will be elaborated and applied to a vehicle.

II. Related Work

Reverse engineering of software systems has attracted a lot of research attention [2]. In the current work, an essential step in the reverse engineering method is localizing features in the CAN data frames. A similar method for locating features in source code was proposed by Eisenbarth et. al. [3]. It proposes a semi-automatic technique that uses dynamic and static analysis to locate features in different parts of the software. To locate these features, the engineer has to define a scenario that triggers these actions in the system. In the paper at hand, we apply these techniques.

To support the reverse engineering process, many different commercial and academic analysis tools exist on the market. An example of these tools is the CANeye program[1]. The user can define its own CAN database that translates the unstructured data in the CAN frames to the corresponding physical units. However, there is no description of a method by which the database can be build.

In some industrial domains, the semantics of CAN messages have been standardized. In the trucks and bus industry, the SAE J1939 [4] standard is defined which describes all data elements within a CAN-frame. This has been used as a starting point for the NMEA2000 standard [5] for marine electronics and the ISO-11783 standard [6] for agricultural machinery. For passenger cars, however, the semantics of CAN messages have not been standardized.

III. The five step method

As the name implies, the five step method consist of five steps which will lead to the discovery of the meaning of CAN data. This method can be applied to any CAN network regardless of the application.

The five steps are as follows:

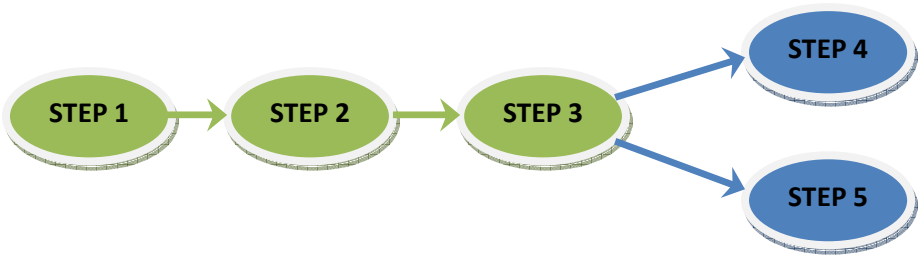

Figure 1: Flow chart of the five step method.

Step 1: Research
To start with, a general literature study of the specific network is held. It should be discovered which control units are present on the network and which kind of data is expected. One should also have general knowledge on the operation of specific subsystems (e.g. ABS, airbags,...). The most important providers of this information are OEM's, importers and automotive literature.

Step 2: The interface to the CAN bus
The actual connection between the CAN bus and the laptop/pc is made. A CAN controller and CAN-analysis software is chosen. There are several commercial programs on the market (CANalyzer, CANoe, CANape, canAnalyzer, CANeye,...), but for this project a homemade analysis program was developed.

Step 3: Linking the identifiers to the ECU's
By linking the CAN-identifiers (ID's) to the in-vehicle control units, the identifiers are linked to a specific subsystem. In practise, the relation between ID and control unit is made by disconnecting the control units one by one.

[1] *http://katho.caneye.be/*

Step 4: Finding essential data (top-down)
Data that simply must be present from a functional point of view, is identified with a software program that is able to compare two logging sequences. The program shows the difference in value for a certain data byte. Due to this, relations between functionality and data can be revealed.

Step 5: Finding the meaning of unknown data (bottom-up)
The remaining, unidentified data bytes can be identified by logging and analysing the data. A very important aspect is that these data bytes are already linked to a specific subsystem from step 3. This way, the possibilities from a functional point of view are narrowed. Relations are discovered through analysing multiple log files with a spreadsheet program.

As shown in figure 1, steps 4 and 5 can be accomplished in parallel.

Step 3: Linking the ID's to the ECU's	
Precondition:	The SW is running properly. The physical location of the controllers is known.
Problem:	The origin of the ID's has not been found.
Difficulties:	none
Solution:	Linking of the ID's to the controllers by disconnecting the controllers one by one.
Post conditions:	All identifiers are linked to a certain ECU.

Table 1: Template for the documentation of step 3.

The execution of the five step method is documented according to a set of templates, one of which is shown in table 1. *Pre-* and *post conditions* are defined to point out the input respectively the deliverables of the current step. The *problem* is described as well as possible *difficulties* that this step could bring along. The core of the step is the fourth topic where a detailed description of the *solution* is covered.

IV. Proof of concept

The five step method is applied to a Volkswagen Golf IV at the premises of the engineering innovation company: LMS International.

As described in step 1, there is a detailed investigation preceding the actual reverse engineering of the CAN network. In this research, the most important contributor was the Belgian importer of VW/Audi. They provided diagrams of the network that showed how many ECU's are present, where they are located and what the basic functionality is.

In step 2, some different homemade software programs were used to analyse the CAN bus: they visualise a list of CAN-identifiers that are observed during the test conditions. This list can also be logged into spreadsheets and set out in graphs. The other way

around, the programs also allow sending[2] data into the network. This way, one can verify whether it has the expected response. (step 2)

Then the ID's are linked to a subsystem by disconnecting all seven controllers from the powertrain network one by one. This caused a disappearance of several identifiers per controller. Obviously, the identifiers that are still present on the bus could not be sent by the disconnected ECU. By correlation, the disappeared ID's are linked to the disconnected controller. (step 3)

Locating required data (step 4) and interpreting unknown data (step 5) is accomplished by analysis software in combination with the insights developed in the first step of the method. Logging data and analysing the data in spreadsheets culminated in the appearance of certain behaviour of the data bytes. After multiple measurements and data logging, this behaviour could be linked with a characteristic of the subsystem, such that the data byte obtains a meaning.

Example of step 4: The engine speed

In this (and any other modern) vehicle, there is sensor data that needs to be present on the CAN bus to have a normal functioning vehicle: e.g. the engine speed. The engine speed is transmitted from the motor management ECU to at least one other control unit. An obvious control unit is the dashboard controller which needs the engine speed data to display on the tachometer. In this example, the vehicle has a six-speed automatic gearbox thus also an automatic gearbox ECU. Evidently, this ECU also requires the engine speed data to select the correct gear. The next paragraphs will clarify the method that was used to find the engine speed data byte(s).

The engine speed was discovered in 3 phases: phase one were real-time tests with the vehicle parked, phase two were road tests and phase three were control transmissions. In the beginning of phase one, the contact was switched on but the engine was not running. Although the engine was off, the throttle was pushed, causing several data bytes being altered. Also when the engine would be running, these data bytes would change as they are related to the throttle. This way the set of possible data bytes that could contain engine speed data is reduced. Then, with the vehicle still parked, the engine was started. When the throttle was pushed again the engine speed increased. Every data byte that increased proportional to the throttle but did not increase during the previous test, was considered a possible engine speed data byte.

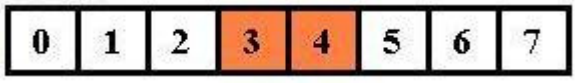

Figure 2: Representation of the eight data bytes of identifier 0x280 that contains the engine speed.

In phase two, the vehicle was tested on the road. While driving, several data bytes were logged into a spreadsheet and analysed afterwards. Key data bytes to log are the selected gear and the throttle sensor which were already identified at this point. The logging of these two data bytes and the suspected engine speed byte gave the possibility to exclude all the incorrect engine speed candidates. In this example the engine speed was located at ID 0x280. Data bytes 3 and 4 represent the engine speed in little endian form (see figure 2).

[2] *Warning: sending data on the network might influence the networks' behaviour in an unexpected way. Caution and well considered actions are advised.*

After the engine speed was identified, the verification phase began. Through transmission of data in ID 0x280 data byte three and four, the previous findings were inspected. The transmission resulted in the indication of the corresponding value by the tachometer. As can be understood, not all vehicle data can be easily verified in this way.

Figure 3 shows a graph containing some measured vehicle data from the powertrain CAN bus. Besides the engine speed, also the vehicle speed, delivered torque and selected gear were found in step 4.

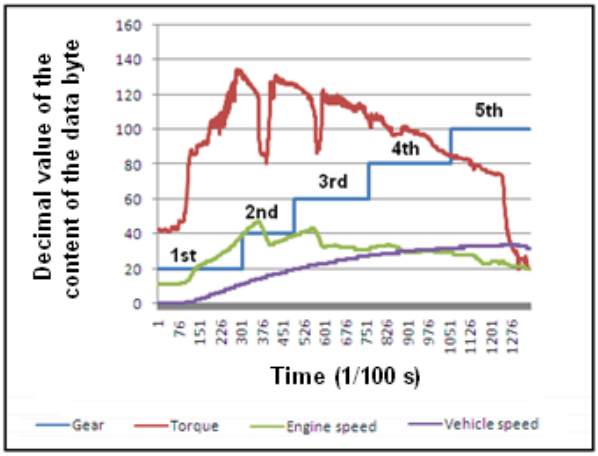

Figure 3: Data that is set out in a graph to be analysed: the engine torque was found.

Example of step 5: The fuel consumption

In modern vehicles the fuel consumption is shown on the on-board display. However, this data was not located in step 4 due to a non-straightforward representation of the fuel consumption. Instead, it was discovered in step 5 where an unknown data byte, data byte three of identifier 0x480, was inspected.

At first, the data byte appeared to be a normal 8-bit counter. Though, after a short real-time inspection it was found that this data byte depended on some inputs. Some conclusions were made after logging and analysing the files:

- The counter runs very slowly when the engine is idling.
- The higher the engine speed, the faster the counter runs.
- The higher the engine load, the faster the counter runs.
- Suddenly releasing the throttle pedal when the engine speed is high, resulted in a temporarily stop of the counter.

These findings strongly point in the direction that the data byte is a digital representation of the fuel consumption. To verify this assumption, a random value was transmitted on the CAN bus which resulted indeed in a changing value on the on-board display. This validated the earlier assumption.

V. Results

A total of 148 data bytes which originate from the CAN powertrain network were visualised during the project. 66% from these 148 bytes were inspected which was considered to be sufficient to prove validity of the method.

A lot of data was discovered: 74% of the inspected data bytes were successfully identified, 22% had a constant value during the complete test cycle and just 4% was not identified due to non-repetitive behaviour (figure 4).

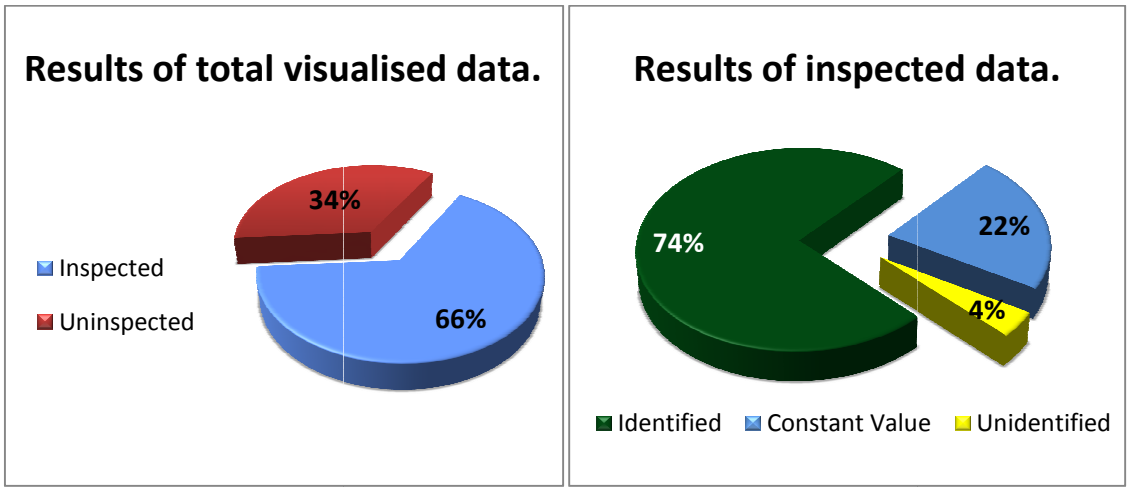

Figure 4: Graph of the results of the total visualised (left) and inspected data (right).

VI. Conclusion

A five-step method is proposed for the reverse engineering of a CAN network. The method is successfully tested on a powertrain CAN network of a passenger vehicle. Out of the inspected data, the semantics of 74% could be defined. This resulted in a good insight in the given CAN network and paves the path to integration of existing CAN networks in testing environments such as HIL-testing.

It must be questioned whether or not it would be possible to make an automotive standard which standardises the CAN identifiers and corresponding data, as is already the case for heavy duty vehicles. This way the reverse engineering of the network would be much easier or even almost unnecessary.

References

[1] International Organization for Standardization: ISO 11898-1:2003 Road Vehicles – Controller Area Network (CAN) – Part 1: Data link layer and physical signaling Std., 2003

[2] H. Müller, J. Jahnke, D. Smith and M. Storey. Reverse Engineering: a roadmap, Proceedings of the conference on the future of software engineering, 2000

[3] T. Eisenbarth, R. Koschke and D. Simon. Locating features in source code, IEEE Transactions on Software Engineering, 29(3):210-224, 2003

[4] Society of Automotive Engineers: SAE J1939-71: Vehicle Application Layer Std., 2008

[5] L.A. Luft, L. Anderson and F.Cassidy, NMEA2000 A digital Interface for the 21st century, Institute of Navigations 2002 National Technical Meeting, 2002

[6] M. Stone, K. McKee, C. Formwalt, R. Benneweis. ISO 11783: An Electronic Communications Protocol for Agricultural Equipment. Agricultural Equipment Technology Conference, 1999.