Software Development of OSEK/VDX Direct Network Management

Rong-Terng Juang
Engineer, Automotive Research & Testing Center
No.6, Lugong S. 7th Rd., Lugang, Changhua County 50544, Taiwan
rtjuang@artc.org.tw

ABSTRACT
Network management (NM) is essential to the safety and reliability of in-vehicle communication networks. This paper reports on a software development of OSEK/VDX direct NM specified in ISO 17356-5. The design flow, consisting of software implementation, manually functional verification and auto-testing, is elaborated step by step. Following this design flow, the NM software development is straightforward and efficient. The major contribution of the paper is to walk designers through the NM implementation, and then this methodology can be extended to various software developments on vehicular control units.

Keywords: OSEK/VDX, Network Management, Software Development

1. INTRODUCTION
There is an increasing tendency for electronic control units (ECUs) made by different manufacturers to be networked within vehicles. To ensure the functionality of inter-networking, Open Systems and the Corresponding Interfaces for Automotive/Vehicle Distributed eXecutive (OSEK/VDX) specifies interfaces and protocols for network management (NM) [1]. It is a well-known joint project of the automotive industry, aiming at an industry standard for an open-ended architecture for distributed control units in vehicles. Besides NM, OSEK/VDX also defines a real-time operating system (OS) [2] and software interfaces and functions for communication (COM) [3].

This drive towards standardization puts pressure on software suppliers to comply with standards. However, many suppliers have a vested interest in proprietary software and are seeking ways to migrate their existing code-base to comply with these standards. Therefore, Denil et al. reported on a feasibility study to wrap a proprietary real-time operating system with an OSEK/VDX compliant interface and evaluated trade-offs when adopting an incremental migration strategy towards a standard compliant interface [4].

With regard to network management, OSEK/VDX supports both direct and indirect mechanisms. A design and simulations of the direct NM on Controller Area Network (CAN)
using CANoe tool was presented in [5]. The design of network management involves the settings of multiple variables. Therefore, the performances of a real CAN network resulting from different settings were discussed in [6].

In addition to direct NM, the implementation of indirect NM on CAN network was given in [7], where the performance was analyzed based on real applications. Besides CAN network, both direct NM and indirect NM were applied to Local Interconnect Network [8]. The advantages and disadvantages of these different implementations were analyzed. And it was suggested that direct NM is appropriate to manage the CAN network, while indirect NM is appropriate for LIN network [8].

OSEK/VDX NM is now a published standard of the International Organization for Standardization (ISO): ISO 17356-5: 2006 [9]. An implementation of the NM software should be able to run on the varieties of available control units. Therefore, this paper presents a methodology for the NM software implementation and system verification.

The remainder of this paper is organized as follows. Section II reviews the OSEK/VDX NM standard. Section III elaborates the design flow of the NM software development. The design flow covers the implementation of the NM standard, construction of a virtual network, emulation and verification of NM functionality, and auto-testing. Finally, some conclusions are drawn in Section IV.

2. REVIEW OF OSEK/VDX NETWORK MANAGEMENT

The essential task of network management is to guarantee the safety and reliability of a communication network for control units, based on network monitoring. OSEK/VDX NM offers two mechanisms: direct monitoring by dedicated NM communication using token principle; and indirect monitoring by monitored application messages. The debate on the pros and cons of these two alternatives is out of the scope of this paper. The discussions here are limited to the direct NM.

Figure 1 shows how the OSEK/VDX direct NM is embedded into an ECU system. The NM consists of algorithms defined by OSEK/VDX and protocol specific algorithms. It communicates with the application software, e.g. station management, in the application layer, and vice versa, by using an application program interface in order to enable (or disable) OSEK/VDX algorithms, which take charge of the operation and transition of NM states. At the bottom, the protocol specific algorithms are responsible for handling data transfer on the network by communicating with the data link layer.

On an ECU network, the NM is implemented in all nodes, with each node having one unique ID. Dedicated messages, Ring, Alive, and Limp Home messages, are used in network monitoring. The nodes transmit and receive the NM messages via a logical ring in which a communication sequence for synchronization is defined. To set up a logical ring, a virtual token is sequentially passed from the node with the lowest ID to the one with the highest ID, and then it is passed back to the lowest ID node. During the circulation, token-possession
grants the node permission to broadcast a Ring message on the communication bus. The reception of a Ring message is interpreted as broadcaster-specific alive signal and synchronization to initiate the transmitting of the successor’s NM message, according to the token ring protocol.

The Alive message identifies a new node present on the network and puts the network into a transient state. After the new node learns the presence of all the other nodes, the network changes into a stable state where each node is fully aware of the statuses of the other nodes on the network. If abnormal, the faulty node broadcasts a Limp Home message cyclically. Other nodes that have received a Limp Home message update their configurations to identify the malfunction node being absent from the bus. The faulty node can not enter a reset state and perform an NM initialization until being able to receive any NM message from other nodes correctly. Accordingly, the status of the network is recorded and evaluated uniformly at all control units.

For supervision of the NM communication and the transition of the NM states, some alarm services are defined in the NM based on timers and counters, such as $T_{\text{Typ}}$, $T_{\text{Max}}$, $T_{\text{Error}}$, $T_{\text{waitBusSleep}}$, $T_{\text{Tx}}$, $\text{NM}_{\text{rxcount}}$, and $\text{NM}_{\text{txcount}}$. For more details, please refer to the ISO standard.

![Diagram of network management environment](image)

**Figure 1. Direct network management environment**

3. DESIGN FLOW OF NM SOFTWARE DEVELOPMENT
The design flow starts from the implementation of the NM standard. Then, this implementation is used to construct a virtual network with multiple control units on a computer. Lastly, the NM software is ported to a microcontroller, which joints the simulated network through a real communication bus. Consequently, the NM functionality can be verified efficiently.

3.1 Implementation of NM Standard
At the beginning of development, the NM source code must be implemented. Because each
control unit is responsible for certain ECU-specific tasks, the NM functionality is incorporated into a control unit in a time-division multiplexing manner. That is, the NM software routine is executed periodically to reduce the inference with the primary tasks. Figure 2 depicts the architecture of the NM implementation. The NM software routine is repeatedly triggered at the end of a static time period. In addition to the NM messages received from other nodes, some predefined events are fed into the NM software routine. The events include (i) NMOn, indicating NM is switched on; (ii) NMOff, indicating NM is switched off; (iii) Sleep, meaning sleep mode is desired; and (iv) Awake, meaning sleep mode is not desired. These routine inputs are implemented in an event-driven manner. The received NM message and the raised event are put in an arrival message queue and an event queue, respectively.

The NM software routine is divided into three steps. When executed, the first step is to check if any event is raised and then perform the corresponding process. Secondly, the NM routine checks if any message arrives or any alarm occurs. The message produced in the first two steps is put in a departure message queue. If available, lastly, a NM message is transmitted.

![Figure 2. Architecture of the NM implementation](image)

Here, the NM software is implemented in console mode using C/C++ programming language. The scenario is described in a text file, which specifies the number of nodes on the network and the timestamps of event occurrences associated with each node. These nodes connect to each other through a simulated communication bus. That is, the NM protocol data unit (NMPDU) is exchanged via this bus. Basically, it contains source ID, destination ID, and the NM status. For debugging, the data traffic on the bus is logged. The NM functionality is verified by analyzing these bus communications. Ideally, this NM software will be adopted in the next step without further debugging.

3.2 Construction of Virtual Network of Control Units

The verification above is based on the predefined NM behavior description, which inevitably lacks inclusions of all possible combinations of the NM behaviors. Therefore, the second step in the design flow is to develop a simulated network in a visually appealing manner. Here, the
windows program for network simulations is built using GCC compiler with wxWidgets library.

Figure 3 shows a screenshot of the developed network simulator. This simulator consists of four virtual control units connecting to a simulated CAN bus and a panel for the display of NM messages on the bus. The NMPDU to be transmitted is translated into CAN message following the CAN 2.0A specification. Reversely, any arrival CAN message is reconstructed as a NMPDU before transferring to the upper layer.

In the virtual network, each control unit is equipped with one toggle button and three regular buttons, which are used as triggers for NMO/nMO, LimpHome, Sleep, and Awake, respectively. Pressing the LimpHome button indicates the failure of participation in the logical ring. Based on these buttons, developers can exercise NM behaviors of every variety. Furthermore, there are also panels available for visualizing the ECU parameters and status, including node ID, successor ID, current NM status, and timers.

This purely virtual network is valuable because permitting developers to analyze the bus communications and NM behaviors at a very early point in the development process without the presence of real control units. Accordingly, the NM functionality can be verified thoroughly and efficiently.

3.3 Emulation and Verification

In this step, the NM software is ported to a micro-processor on a real hardware, which is then incorporated into the virtual network through a real CAN bus.

Figure 4 illustrates the integration of a real hardware with the virtual network. The virtual ECU network is simulated on a PC. The PC accesses a real CAN bus through the Kvaser Leaf.
SemiPro HS [10], which provides USB interface for CAN. The Microchip’s Explorer 16 Demo Board [11] is taken as a real control unit. The NM software is ported to the PIC32MX795F512L micro-processor on the demo board. This demo board accesses the CAN bus via a Microchip’s CAN transceiver [12]. The CAN protocol here follows CAN 2.0A and the data rate is set as 250Kbps.

Any NM message on the simulated CAN bus of the virtual network is sent to the real CAN bus. Reversely, any arrival message at the Kvaser Leaf SemiPro HS is transferred to the simulated CAN bus. Therefore, the real hardware and the virtual control units are logically connected to the same CAN bus, resulting in five control units on the network.

The same to the virtual control unit, four switches on the demo board are used as the triggers for NMO/sub/NMOff, LimpHome, Sleep, and Awake, respectively. Besides, an alpha-numeric 16 x 2 LCD display is available on the demo board to visualize the real-time NM status. Accordingly, developers can test the integrated network by manipulating the triggers on both the real hardware and the virtual control units. Therefore, the NM software can be validated thoroughly and efficiently.

![Figure 4. Coupling of real ECU and virtual network](image)

3.4 Auto-testing

Step by step, the developed NM software is ready to be ported to any new hardware. Although the porting can be verified manually using the tool shown in Fig. 4, automatic test is desirable. To this end, an auto-testing tool is developed. The framework of this tool is identical to the one shown in Fig. 4, where a PC connects with a control unit under test through a real CAN bus. However, the virtual ECU network on the PC is not visualized and only parameter settings are available here. Figure 5 shows a screenshot of the auto-testing software on the PC, which includes four control units. Like the first step, each control unit behaves according to the description in a texted scenario file. But, the difference is that the first virtual control unit shares the same ID with the control unit under test, meaning these two control units are logically the same node on the network. In fact, the presence of the real control unit is irrelevant to the operations of these four virtual control units. The NM messages produced by
the real control unit are not transferred to this virtual network. Reversely, besides the first virtual control unit, the NM messages sent from the rest control units are replicated and broadcasted on the real CAN bus. Consequently, from the view of the real control unit, there are three other nodes on the network. If functioning correctly, the control unit under test acts as an image of the first virtual control unit. Therefore, its functionality can be validated by comparing the NM messages in time series with the first virtual control unit.

### Figure 5. Screenshot of auto-testing tool

Note that the virtual ECU performs according to the script, which specifies the timing of event occurrences. To cover the combination of NM behaviors as extensive as possible, the scenario files should be elaborately designed. In addition, since the real control unit is supposed to be the image of one of the virtual control units, the event triggering should be also implemented on it. Here, the event triggering is achieved by sending dedicated commands to the control unit under test. Figure 6 reveals the command format derived from the NMPDU, where the most significant bit is used to distinguish between the regular NMPUD and event triggering commands. Similar to the previous design, four switches, SW1, SW2, SW3, and SW4, are used for the triggering of NMOn/NMOff, LimpHome, Sleep, and Awake, respectively.

<table>
<thead>
<tr>
<th>Virtual ECU</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseID: 0x400</td>
<td>T_Typ: 260</td>
</tr>
<tr>
<td>ECU 1: 0x01</td>
<td>T_Max: 100</td>
</tr>
<tr>
<td>ECU 2: 0x02</td>
<td>T_Error: 1000</td>
</tr>
<tr>
<td>ECU 3: 0x03</td>
<td>T_WES: 1500</td>
</tr>
<tr>
<td>ECU 4: 0x04</td>
<td>T_Txt: 25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real ECU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: 0x01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>File: scenario.txt</td>
</tr>
<tr>
<td>Repetition: 10</td>
</tr>
</tbody>
</table>
Lastly, the ITRI WAVE/DSRC Communication Unit (IWCU) [13] is adopted as a target device for the software porting. DSRC radio is primarily used for the purpose of automotive safety applications [14]. Therefore, the IWCU is a good example for the NM application, because it is important to ensure that the safety-related controller is present and functioning normally, which is the fundamental objectives of network management. The IWCU is equipped with Freescale’s MPC8377 processor, running at 667MHz, and supports CAN bus 2.0 interface. By using those tools developed above, hardware-in-loop test is achievable and the porting of NM software is accomplished smoothly and rapidly.

4. CONCLUSIONS
This paper has presented the software development of OSEK/VDX direct NM specified in ISO 17356-5. The elaboration of the design flow has been carried out step by step. Firstly, the visualized virtual network supports versatile simulations of the NM functionality, enabling the testing of the NM software without the presence of hardware. Secondly, the coupling of a real hardware with the simulated network makes it possible to validate the NM software efficiently. Thirdly, the auto-testing facilitates the examination of NM software on various types of hardware. Based on these tools, software porting to any new hardware can be achieved smoothly and rapidly. Moreover, this methodology can be applied to various software developments on vehicular control units.

5. Acknowledgment
This work was supported by Department of Industrial Technology, Ministry of Economic Affairs, Taiwan, under Project 102-EC-17-A-03-01-0809.

6. REFERENCES


