IN VEHICLE NETWORKING & DIAGNOSTICS WITH MODEL BASED DEVELOPMENT APPROACH

AUTHOR & SPEAKER: PARAS BALDOTA.
COMPANY: HARMAN INTERNATIONAL INDIA.
CONTACT: PH. 09325111094.

EMAIL: (PARAS_BALDOTA@HARMAN.COM / PARASBALDOTA@YAHOO.CO.IN)

DOMAIN:
MODEL BASED DESIGN AND AUTOMATIC CODE GENERATION.
TERMINOLOGIES & ABBREVIATION

- IVN : IN VEHICLE NETWORKING.
- FMS : FAULT MANAGEMENT SYSTEM.
- CAN : CONTROLLER AREA NETWORK.
- TLC : TARGET LANGUAGE COMPILER
- RTWEC : REAL TIME WORKSHOP EMBEDDED CODER.
- MBSD : MODEL BASED SOFTWARE DEVELOPMENT.

by PARAS BALDOTA.
OBJECTIVE

- MBSD approach widely used for development; simulation and testing of automotive control strategies but not much often used for IVN and Diagnostics.

- Current tool support from Mathworks is limited only for testing.

- First time introducing the Model Based approach for development cycle of IVN application interface and diagnostics layer to seamlessly integrate with control strategies modules.

by PARAS BALDOTA.
INTRODUCTION

- MBSD approach has the advantage of simulation, flexibility in testing and verification and supports automatic code generation.

- Extending exactly the same advantage to IVN and diagnostics brings most of the system under single roof of model.
MODEL BASED DEVELOPMENT PROCESS

FIGURE 2

by PARAS BALDOTA.
Probable reasons may be:

- These layers being mostly developed independent of application layer,

- feasibility limitation of driver simulation and

- There are relatively less changes in these layers in product lifecycle as diagnostics is protocol specific and IVN is controller specific to some extent.
But when it comes to testing, IVN and diagnostics layers not only play important role as input/output to main application but also used as flow between different logic modules, very similar to importance of veins in our body.

These layers being hand coded, provides limited scope to use the advantage of MBSD approach stated above.

by PARAS BALDOTA.
To bring the network and diagnostic interface under the modeling environment, there are two approaches:

- The complete stack of OSI 7 layers except physical layer can be perceived using MBSD approach.

- Or in most of the cases, datalink layer to session layer is provided from third party vendor such as Vector, in this case, developing the application interface along with diagnostics and fault management system using MBSD.

This paper will describe the methodology for approach 2.

by PARAS BALDOTA.
MBSD Approach for IVN and diagnostics interface. (Red outlined boxes are provided from 3rd party vendor such as Vector)

by PARAS BALDOTA.
Now to interface IVN & Diagnostics with application, we will categorize the further process into 2 steps:

- Diagnostic palettes for FMS (Fault Management system).
- CAN message transmit-receive palette
A fault management system is a unit which identifies a specific component or system fault, leading to an observable failure effect, manages the handling of fault & acts as an interface to the external world for fault diagnosis. In brief, FMS will cover below features:

- Detecting the Fault
- Analyzing the fault
- Systematic Recording of the Fault
- Indication of the Fault to the user
- Diagnosis of the Fault
- FMS would need various APIs to support the successful running of diagnosis.
- These APIs can be converted in form of model blocks and interweaved in control strategies.
- These blocks will play important role during simulation and code generation.
- These blocks will be used for decision making during simulation by enabling or disabling the sub-system if fault is detected.
- The same flow can be converted into code by correctly replacing the block with underlying function call. (Using S-function and TLC options in case Mathworks tools.)

by PARAS BALDOTA.
Few generic functions which can be interfaced using model blocks:

1. **Enable/disable Fault Manager:**
   This feature is required during certain critical functionality such as programming, EEPROM writing etc.

2. **Know the Fault Manager status:**
   This block would give output as the current status of fault manager if enabled or disabled.

*by PARAS BALDOTA.*
3. **Report/clear the fault:**

While reporting the fault, first all the faults also referred as DTCs (Diagnostic trouble codes) can be imported via ODX format (Open Diagnostics Exchange) or through excel. Then specific fault can be reported or cleared by using index mechanism. The block API can also provide the facility to add new DTC in DTC table.

4. **Query fault status:**

Again fault querying can be achieved through indexing and by maintaining separate fault status table whose indices are one to one mapped fault indices.

by PARAS BALDOTA.
5. **Reset fault:**
   Fault status can be reset by using its index. May be used for resetting the faults during operation or ignition cycles.

6. **Clear Active / Inactive DTCs:**
   Such requests generally received through tool. The block will output the status if cleared or not.
These blocks can be easily integrated within the application control strategy which not only simplifies the coding task but also represent better behavioral model for simulation.

Here utmost care needs to be taken for modeling of various physical faults whose simulation behavior and code generation behavior are different due to limitations in simulation. In such cases, this can be very well achieved by **configurable subsystems** in which Simulink hook can be used to switch between subsystems to be used during simulation and code generation. In simple cases, s-function and TLC can handle such scenarios.

*by PARAS BALDOTA.*
MBSD approach can also be used for fault maturity due to its inherent support to **Stateflow** machines. This includes managing DTC lifecycle from fault detection to changing its state to pending and then confirmation of fault based on categorization of faults.

This fault maturity process in conjunction with diagnostics interface blocks explained above helps in handling increasing complexities of the dynamic system.

*by PARAS BALDOTA.*
Current the similar palette is present in existing Vector toolbox, but it is used only during testing and supports only Vector formats.

Now the approach is to develop the tool independent and more robust architecture with support to code generation with the help of s-functions, TLC (Target Language Compiler) files and RTWEC.
Application receives various information from other control units via CAN. By importing the CAN database file into modeling environment, the CAN messages can be associated with specific channel. These messages are then encoded or decoded using transmit-receive palette with application specific message and signal names such as VehicleMode, EngineSpeed.

Further scaled engineering units with necessary resolution for particular signal can be added. The palette will also provide the bit packing and unpacking rules for the associated signals.

by PARAS BALDOTA.
The palette would need below attributes for CAN message which can be inherited from CAN database file:

1. Name of the CAN message.
2. Identifier type, standard or extended.
3. CAN Identifier, preferably hex number.
4. Length of CAN message.
5. Next and most important, the signals contained in message with below attributes:
   - Signal name as per application usage.
   - Signal datatype.
CAN MESSAGE TRANSMIT-RECEIVE PALETTE : APPROACH CONTD...

- Start bit of the signal with respect to entire CAN message.
- Length in bits, number of bits signal occupies in message
- Byte order weather little endian or big endian
- Resolution of the signal while sending or receiving over CAN
- Offset, required in case negative values of signal or if minimum value of signal if other than 0.

by PARAS BALDOTA.
In code generation of these transmit/receive blocks, these will be replaced with transmit/receive API function calls provided from third party stack such as Vector. These function calls would need the various inputs which will be inherited from attributes added above.

Before passing the information about message from the palette to API function calls, preprocessing may be required. For e.g. resolution and offset settings must be operated on received data for all the signals in message before passing it to function calls. This can be achieved by using various features of code generation tools. If we use RTWEC for code generation then using S-function and TLC files, necessary preprocessing can be achieved.

by PARAS BALDOTA.
• Thus we have seen here, how model base development approach helps to focus on analyzing our own signals rather than defining them, validating the logic with actual field data reducing the risk of introducing the error in later stages. Moreover, adding diagnostic interface under the umbrella of model based environment increases the reusability and scalability of components, further reducing cost and time to market.

by PARAS BALDOTA.
REFERENCES

- [http://www.mathworks.in/help/simulink/slref/sfunction.html](http://www.mathworks.in/help/simulink/slref/sfunction.html)

Thank you!

by PARAS BALDOTA.