A Walk through EMS2010 Modular Software Development

Jean-Marc Dressler

A Walk through EMS2010 Modular Software Development

Jean-Marc Dressler[1]

1: Renault, 1 rue du Golf, 78288 Guyancourt France

Abstract: The development of internal combustion engines is driven by two forces: the need for increased performances and most important of all the regulation for emissions reduction. Thanks to electronic control, in recent years engines have known spectacular progression in those two areas, but also a spectacular increase in complexity, especially for software. With the current high level of complexity rigorous processes are not enough to guarantee the quality. This is why RENAULT has launched the EMS 2010 (Engine Management System) project to build the standardization of the software structure with the support of its suppliers, especially the support of SIEMENS VDO.

This article presents the main specificities of EMS 2010 software development through two parts. The first part introduces the EMS 2010 software architecture, its principles and main notions, and ends with a comparison with AUTOSAR. The second part goes through the technical particularities of the different steps of EMS 2010 software development from the specification of a module to its integration into a complete software.

Keywords: Modular architecture, MBD, MIL, SIL

1. Architecture

1.1 An applicative architecture

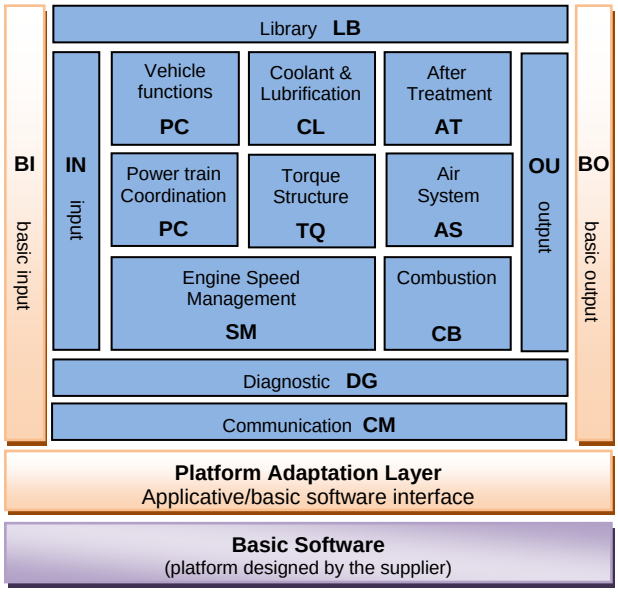

Figure 1: EMS 2010 architecture

An embedded software architecture can roughly be divided in two parts:

- The applicative software which contains the control laws and strategies which support the system functionality and performances.
- The basic software which main role is to manage the hardware resources of the ECU and to provide services (communication, memory management, scheduling) to the applicative part.

For an engine manufacturer applicative software is the strategic part to define and control, and this is why EMS 2010 focuses on the applicative software. As shown by figure 1, the EMS 2010 architecture covers the applicative part (the darker boxes) and its interfaces with the basic software (the three boxes surrounding the applicative software). The basic software itself remains specific to the supplier of the ECU.

1.2 A modular architecture

The EMS 2010 architecture has three level of composition: the higher level is the function (this is the level visible on figure 1), functions are divided in sub-functions which are themselves composed of modules. The modules are the atomic elements of the architecture and a complete software can contain about 200 of them.
A module has a well defined functionality and interface. Also the division into modules aims at reducing dependencies. The benefits of such a modular architecture are well known:

- Module level validation (reduces the complexity and cost of validation).
- Improved evolution management (impact analysis, containment, validation).

Another important principle of EMS 2010 is the reusability at the source code level. EMS 2010 distinguishes two kinds of modules:

- Specific modules which are specific to a supplier platform. Examples of specific modules are the basic input/output (functions BI and BO) modules which make the interface between the supplier platform and the applicative software.
- Reuse modules which can be integrated in any platform with an EMS 2010 compliant interface (e.g. a reuse module can be common to diesel and gasoline engines). The EMS 2010 software

design rules ensure that reuse modules are portable at the source code level.

Most modules are reuse modules, which means that they are developed once and then reused on the different engine projects. Together modularity and reusability set the basis for a simpler and better validation and therefore an improved quality of the final software.

1.3 Configuration mechanisms

EMS 2010 provides two levels of configurations:

- On-board configuration: this is the usual calibration mechanism. It is mostly used to fine tune the control laws and strategies and adapt them to a particular vehicle. This mechanism provides a great flexibility as calibrations can be changed on a running ECU.
- Off-board configuration: this mechanism has been introduced to efficiently handle technical definition diversity. This configuration takes place at compile time and as such is much less flexible than calibration but it avoid wasting memory resources with unnecessary code.

With off-board configuration a module can handle different technical definitions such as different sensor or actuator technologies, diesel/gasoline engines with a unique module version. Alternative solutions would have been either to use different versions of the same module or finer grained modules but it would have introduced more complexity in the version management. The off-board configuration mechanism will be detailed later on.

1.4 Functions organisation

In this section we will have a closer look at the software structure showed by figure 1. The layout of the figure is organized so that the left to right orientation shows the sense of the main data flow between the functions, starting with the BI and ending with the BO. The modules in IN and OU functions perform applicative signal filtering and formatting. In the remaining functions we distinguish the transversal functions LB, DG and CM from the non transversal functions which contain the control laws and strategies for the engine sub-systems.

The role of transversal functions is to provide common services to other modules:

- The DG (diagnostic) function contains the diagnostic services (i.e. interaction with diagnostic tools) and the failure manager. Note that the failure manager does not handle the faults itself since dysfunctional behaviour is managed by the modules themselves, but it collects the fault reported by the modules for diagnostic and fault confirmation purposes.
- The CM (communication) function is composed of one module which handles the applicative strategies for bus/network communications. This module is also responsible for performing multiplexing / demultiplexing of network frames so that other applicative modules only see the signals and never the frames.
- The LB (library) function essentially contains one reuse module which provides commonly used operations (filtering, timers, …) through an API.

1.5 The platform adaptation layer

In order to provide an EMS 2010 compliant platform, each integrator supplier must implement and configure the EMS 2010 platform adaptation layer above his own basic software platform.

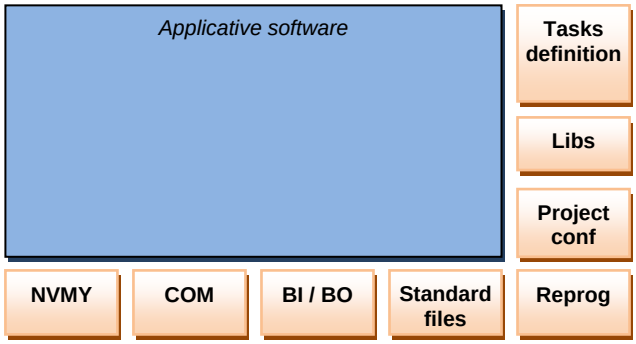

Figure 2: The platform adaptation layer

The first role of the adaptation layer is to provide an interface between the basic and the applicative software. This interface is composed of the BI and BO modules, the COM interface (mainly used by the applicative CM function) and the NVMY (the non volatile memory manager) interface.

In addition to those interfaces the adaptation layer provides the following non reuse libraries: an arithmetic library for saturated operations (additions, multiplications …), an interpolation library, a bit operation library and a memory management library. Those libraries are non reuse because they are low level and can be optimized for a supplier platform, and as they are widely used their impact on performance is important.

The adaptation layer also contains the interfaces for the following services: the NVM (Non volatile Memory) manager, the lower communication layer (only used by the applicative communication module).

Finally the adaptation layer also includes the ECU reprogramming functionality, several EMS 2010 source files mainly used for portability (type definition) and the tasks definition and the project (off-board) configuration which we will both see in more details in the section about module integration.

1.6 Comparison with AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. The EMS 2010 project and AUTOSAR phase 1 progressed separately and nearly in parallel and while both architectures are modular there are many differences between the two.

Focus

AUTOSAR aims at providing a general purpose software architecture, and while it the AUTOSAR partnership worked on standardized functional architecture, its main focus has been the definition of the RTE (Run Time Environment) and the extensive standardization of the basic software modules. The EMS 2010 project took a different approach focused on the definition of the functional architecture for engine control software and the associated modularity mechanisms. EMS 2010 kept standardization of basic software to a minimum by only specifying the interfaces necessary for the applicative software.

Modularity mechanisms

At the centre of the AUTOSAR architecture lies the RTE. The RTE is responsible for inter module communications, and to that purpose provides sender/receiver and client/server mechanisms. Connections between modules are made through the RTE configuration by associating the ports of the modules. Compared to the AUTOSAR mechanism EMS 2010 is much simpler because of its static architecture. Modules communicate through data signals (implemented as global variables) and events (implemented as function calls). As EMS 2010 scheduling policy is almost completely cooperative there is no need to protect the global variables. Also there no configuration is needed to "connect" the modules because this connection is done through the signal and event names which are unique. This static architecture has the advantage of reduced resource consumption compared to AUTOSAR more generic mechanisms. Finally the off-board configuration mechanism is completely specific to EMS 2010.

Fault Management

The purpose of the AUTOSAR DEM (Diagnostic Event Manager) is very close to the EMS 2010 failure manager. But contrary to AUTOSAR, the EMS 2010 architecture does not provide a generic fault handling mechanism like the FIM (Function Inhibition Manager) as it is the responsibility of each module to handle the faults. To manage coherency of fault management at the system level, EMS 2010 relies on tools to analyze the degraded modes activated by a fault confirmation.

2. EMS 2010 software development

In this part we will walk through the different steps of the development of EMS 2010 reuse modules.

2.1 Specification

The MBD approach

EMS 2010 takes full advantage of the MBD (Model Based Design) approach. Indeed each module specification is composed of several Simulink® models (Simulink® is a modeling environment developed by The MathWorks). Each model is executable and so can be used for simulation or rapid prototyping. This means that the specification is very detailed and can be validated, sometimes directly on the vehicle. Each module is actually validated in a MIL (Model In the Loop) environment; the test cases used for this validation will be reused for the validation of the final module source code. Each model also has an associated textual documentation to facilitate its comprehension and a data dictionary which gives more details on the characteristic of the signals definition (units, resolution …).

Specification structure

The specification of a module is composed of the following items:

- Specification elements: the elements are equivalent to sub-modules, also each element correspond to a Simulink® model and its interface. One of the elements plays the specific role of module scheduler, this means that it defines the execution sequence of the different elements entry points.
- The MID (Module Interface Description): this is a textual document which describes the interface of the module (data and event inputs/outputs) and its off-board configuration. The interface of the module is actually a subset of its elements interface, defining such an interface avoids to expose unnecessary elements to other modules.

Off-board configuration

The off-board configuration mechanism takes place at the module level. Specification elements are not themselves configurable but simply are present or not in a given configuration. A configuration is defined by a set of configuration constants which can take a limited number of values. For example the variable *engine_type* can take two values: *diesel* and *gasoline*. Figure 3 shows an example of module with two configurations: gasoline and diesel. Note that

since the module interface is a subset of its element interface, the module interface depends on the configuration (actually this is only true for data inputs/outputs)

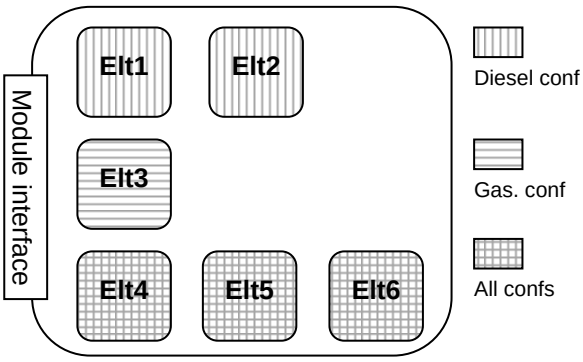

Figure 3: Module off-board configuration

2.2 Module coding and validation

By coding we mean the transformation of the specification in optimized C source code. As explained earlier the main characteristic of a reuse module is its portability: a reuse module can be integrated to any platform which implements the EMS 2010 adaptation layer. To achieve this portability RENAULT in close partnership with SIEMENS VDO has elaborated a set of coding rules. The coding rules:

- Set constraints on the implementation (the code must be C ANSI, only EMS 2010 libraries can be used …).
- Define the structure of the module source code.
- Define the different software mechanisms (memory allocation, off-board configuration …).

Code structure

One of the main principles of the coding rules is the 1 to 1 relationship with specification, and this is reflected in the code structure.

The file structure strictly enforces a file structure of the module which closely matches the specification. Each specification element is implemented by a set of 5 source files named after the element (one C file containing the program, another C file containing the definition for observable variables and calibrations, and three header files for public and private declarations). In addition to files related to elements, a module contains several other files among which the most important are:

- The export file: this header files contains the declaration of all the items which are exported by the module to other modules. This mainly includes the variables produced by the module, and the module entry points/functions.
- The import file: this file simply includes all the export files of the modules which produces items which are consumed by the target module. Actually this file implements statically the dependencies between the modules.
- The configuration file: this file implements the off-board configuration logic of the module.

Again the 1 to 1 relationship with the specification is verified as the 3 files described above directly derive from the MID.

This relationship is also enforced at a lower level, with naming rules for main functions and all observable variables. It is interesting to note that all observable variables and calibration have the same name in the specification and in the code and that this name is also used by the calibration and diagnostic tools.

Off-board configuration

In the section about specification we have seen the off-board configuration mechanisms simply consist in defining a sub-set of “active” specification elements for each configuration. This mechanism is reproduced at the code level by using conditional compiling. Each source file of an element begins with the inclusion of the configuration file immediately followed by an #ifdef ELEMENT_NAME… #endif block which encompass all the rest of the file. If an element is not present in a given configuration, the configuration file will not define ELEMENT_NAME and thus after the pre processing the element files will be empty and the element will not be present in the final binary avoiding unnecessary waste of resources. Thanks to this mechanism, there is a single set of sources for all configuration of a module.

MIL/SIL validation

One part of the module validation is done in a SIL (Software In the Loop) environment. Thanks to a dedicated tool and to the 1 to 1 relationship between code and specification, the integration of the module source code in the SIL environment and the comparison between MILS and SIL results is greatly facilitated. The test cases and environment used for the SIL are exactly the same as in the MIL. The module source code is compiled as a Simulink® S-function to allow its integration in the SIL Simulink® model. All observable variables can be compared which allows a finer comparison than if only inputs and outputs where compared. It is up to the human tester to decide if the differences between MIL and SIL results are acceptable or not.

2.3. Integration

Once the reuse modules have been coded and validated they are put in an electronic shelf. Each engine software project picks the reuse modules its need for its engine technical definition.

Before the integration the integrator supplier (i.e. the supplier which provides a complete ECU integrating the reuse modules) must have prepared its EMS 2010 platform adaptation layer (implementation of BI/BO modules, non reuse libraries …). As not all modules are available at the start of the project, the integration is done incrementally. For each integration step RENAULT provides to the supplier:

- The code and MID of the reuse modules to be integrated.
- The specification of non reuse modules to be developed or updated.
- The tasks definition specification which defines for each task (periodic tasks like 10ms or 20ms, and events like the engine top dead center event), all the modules entry points to be called and their execution order. Note that this tasks definition is build directly from the specification.
- A stubbing specification for the modules which will be integrated in the future.
- The project off-board configuration specification, i.e. the value of each configuration constant.

EMS 2010 methodology defines the following process for the integration of the reuse modules in the supplier platform:

- The first step is to configure the platform adaptation layer in order to receive the new or updated modules. This step includes the following operations:
  - Update of the project configuration
  - Configuration of the NVMY with the NVM blocks from the new/updated modules.
  - Update the tasks definition.
- The second steps consist in compiling and linking the software with the new modules.
- The remaining steps essentially consist of static and dynamic checks at module and application levels to verify that:
  - The integration has been done correctly.
  - Hardware resources consumption (CPU and memory) is still within the project targets.
  - Timing are compatible with the real time execution of the software (actually the coding rules define the maximum execution time for a module entry point)

The final software consist of a binary and its associated A2L file (the A2L file contains all the information necessary to diagnostic and calibration tools).

3. Conclusion

The first goal of the EMS 2010 project is to improve the quality of the engines software through modularity and code reuse. To support this modular architecture approach many new mechanisms but also processes and tools have been developed.

As EMS 2010 has progressed separately from AUTOSAR, its architecture and mechanisms are quite different. Today RENAULT and SIEMENS VDO bring their experience acquired on EMS 2010 through their participation to AUTOSAR phase 2.

Modularity and code reuse have also opened the way to a business model were the developer of the applicative software and the provider of the electronic system can be different suppliers. At this time EMS 2010 software is running on ECUs from three different suppliers and each supplier integrates several modules developed by another supplier.

8. Glossary

MBD: Model Based Design
MIL: Model In the Loop
SIL: Software In the Loop
MID: Module Interface Description
ECU: Electronic Control Unit