# FROM CONCEPTION TO IMPLEMENTATION: A MODEL BASED DESIGN APPROACH

**Giovanni Gaviani[1], Giacomo Gentile[1], Giovanni Stara[1], Luigi Romagnoli[1], Thomas Thomsen[2], Alberto Ferrari[3]**

[1]*Magneti Marelli Powertrain, Via Timavo, Bologna, Italy*
[2]*dSPACE, Technologiepark 25, 33100, Paderborn, Germany*
[3]*PARADES EEIG, Via San Pantaleo 66, 00185 Roma, Italy*

Abstract: From the conception of an embedded controller to its implementation, the designer must refine the solution trying to minimize the final cost while satisfying functional and non functional requirements. We present a model-based methodology framework to address this problem and a real industrial case that shows the effectiveness of the proposed approach. Top-down and bottom-up aspects are considered and solutions are found by a meet-in-the-middle approach that couples model refinement and platform modeling. In more details, given a model of the implementation platform, which describes the available services and data types, the algorithms captured by models are refined and then automatically translated to software components. These components are integrated with handwritten (e.g. legacy) software modules together with the software platform. The validation of the system is performed at different level of details of the platform. A final validation phase on the real target is performed to finally validate the functionality and to guarantee that the performance constraints are met. *Copyright © 2002 IFAC*

*Keywords*: design methodology, automotive control, embedded systems, model-based design,

## 1. INTRODUCTION

The design of power-train controllers is a most challenging problem in automotive electronics because of the complexity of the functions to be implemented, the tight interaction among mechanical-electrical components, the safety aspects and the hard cost limits that the system must satisfy.

Time-to-market requirements and the continuously changing specifications have caused the migration of most functions from hardware to a software implementation, and the need of a close interaction between subsystem makers and car makers. At the same time, a sub-system supplier must interact during the design of the controller with component suppliers, such as RTOS and micro-controller providers, in order to exploit the best available technology. This interaction during the design of the system should be supported by a common design methodology and design flow, with a shared understanding of specifications and implementation constraints. In the automotive domain, the SEA Design Chain Initiative (The SEA Consortium, 2003) has the goal to define a common framework and a seamless process-flow for the design of complex embedded real-time systems. This project will enable car makers and sub system suppliers to exchange requirements and solutions and enrich the models with non functional properties, hence strongly reduce the design time and the design cycles.

In this paper we present the model-based design methodology that has been introduced in Magneti Marelli Powertrain that is well aliened with the basic concepts of the SEA project. The design methodology is based on a meet-in-the-middle (Vincentelli and Ferrari, 1999) approach with a defined set of abstraction layers and successive refinements (Balluchi, et al., 2002)

The requirements of the highest level of abstraction, the system, are expressed in terms of functionality and performance indexes. These requirements are captured by executable models and are shared between car makers and sub system maker, drastically reducing ambiguity, i.e. possible interpretation errors. These models are then refined down to implementation.

The design of control algorithms is a fundamental part of the design flow. It starts from a functional specification and ends up with a detailed description of the algorithms. In the model-based design methodology, the part of the control algorithm that is mapped to the software partition is automatically translated from a model representation to a set of software components. The software architecture of the application will accommodate and compose together those software components such that the real-time requirements are met. In the proposed design flow, the control algorithms are captured using the MATLAB/Simulink (The Mathworks, 2002) design environment and the automatic translation of the model to C-language code is performed with the TargetLink (dSPACE, 2002) code generator, in the sequel called model compiler and described in a following sections.

To correctly accommodate automatically and manually generated software components, a software architecture composed of different layers has been totally specified and implemented. The layer closest to the hardware is the basic input output system (BIOS). The upper layer is containing the device drivers that encapsulate electrical drivers of sensors and actuators. The RTOS and communication services are common to all layers. All these layers implement the software platform (Vincentelli and Ferrari, 1999) that supports the application software. The latter has structural and semantic aspects that allow the correct integration of the software components implementing the entire set of control algorithms.

To obtain the final implementation of the power-train controller, several players belonging to different organizations within a company and/or different companies have to cooperate during the design of each component and of the entire system: car makers must provide and share controller specifications, plant models and calibration sets, silicon suppliers must provide performance models of the micro-controllers. A common design methodology and tool chain is the key of success in cooping with the design complexity and constraints.

The paper is organized as follow: the first part describes the methodology framework defined by Magneti Marelli Powertrain. The second part describes the TargetLink environment by dSPACE. The third part is dedicated to the description of a real design case. The fourth one indicates future work and conclusions.

## 2. DESIGN METHODOLOGY

In the proposed approach, five main levels of abstraction are identified: system level, function level, operation level, architecture level, and component level (Balluchi, et al., 2002).

***System***: car manufactures define the specifications of power-train control systems in terms of desired performances of the vehicle in response to driver's commands. Additional requested specifications, defined by governments or car manufacturers associations, are concerned with fuel consumption, noise and tail pipe emissions. At the system level, the given specifications are analyzed and expressed in an analytical formalism. Specifications have to be clearly stated and negotiated between customer and supplier to make sure that they are realizable within the budget and time allowed for completing the design.

***Functions***: the design of the functionality to be realized by the control system to meet the system specifications described above is very complex. A good quality of the design is obtained by decomposing the system into interacting sub-systems, referred to as functions. The decomposition allows designers to address the complexity if it leads to a design process that can be carried out as independently as possible for each component. The structure of the functions is the model of the platform at this level of abstraction. System specifications are spread out among the functional components so that the composition of the behaviors of the components is guaranteed to meet the requested objectives and constraints. The output of the functional level design is a desired behavior for each function.

***Operation***: at the operation level, the desired behaviors have to be obtained, satisfying also some local objectives and constraints. Solutions are expressed in terms of basic building blocks, called operations. In a first design attempt, for each function, control strategies achieving the given specifications are devised and captured with executive models. The control strategies operate on variables that are measured on the physical domain and produce values of variables that act on the physical domain. Then, each control strategy is refined by introducing chains of elementary operations, so that the set of all solutions can be integrated in a unique operations network.

***Architecture***: the design step at the architectural level produces a mapping between the behavior that the system must realize (operations) and the platform representing the chosen system architecture, i.e. an interconnection of mechanical and electrical components (e.g., sensors, actuators, microprocessors and ASICs). The set of components either are available in a library of existing parts or must be designed ex novo.

This architecture and component-selection task is the subject of intense research by the system design community.

At the end of the integration process, the calibration phase, i.e. the tuning of control and model parameters, is carried out in collaboration with the car maker, feeding back the control designers fundamental information of the actual behavior of the system.

The correctness of the final implementation is strictly connected to the correctness of the control algorithms (operations), hence the models from which the implementation has been derived. In the past and since the first design phase, prototypes of target ECU were employed for long time to validate the algorithms, frequently described only on documents and C language, resulting in a very high prototyping effort. Moreover, the design cycles were carried out mainly at the software level, generating problems in tracking back the final solutions at the control level. In the model-based design methodology, validation can start as soon as the designer conceives the controllers with the use of complex models of the plant, describing the engine, driveline and the driver, or in a simpler way set of recorded input/output traces. If a control algorithm is subject to changes, the model is firstly modified and then the new software is generated, resulting in a natural synchronization between the model representation of the control algorithm and its implementation. This drastically reduces the usage of prototypes of target ECUs, ending in a strong reduction of design time and cost. Nowadays, this goal has been only partially achieved because of the difficulties of modeling power-train physical processes and other electro-mechanical components (sensors and actuators). Nevertheless, the creation and use of these plant models play a strategic role in the automotive field.

## 3. MODEL REFINEMENT AND SOFTWARE PLATFORM

To support the methodology, the tool chain must handle the refinement of components from one level of abstraction to another one. Ideally, it should be possible to support, in the same design framework, the refinement of system requirements to control algorithms, and then to software or hardware components. If we consider data refinements, system specification and control algorithms might be provided with floating point notation of quantities, e.g. the quantity of fuel injected in the cylinder, while at the software level this data might be refined to 16 bits fixed point representation.
In the design process, different actors (working even in different companies) will interpret the data in different ways: as floating point or fixed point notation. For example, car maker will provide requirements in floating point notation and will perform calibrations and measurements in a coherent manner, but at the implementation level the power-train controller executes operation only in fixed point data, hence software engineers have to manage software in fixed point notation.
Not a single tool today in the market easily supports refinement in a general meaning. The more advance research project on this topic is the Metropolis project at the University of Berkeley, see (Burch, et al., 2002) and (The Metropolis Project).
In our approach, the algorithm specifications are captured in Simulink and are refined with different models, expressing different level of details, consistently linked by a configuration management system. The verification of the refinement is obtained only via functional simulation.
The translation of a Simulink/Stateflow model to C code might require some design steps, some of them not always applicable. If the target architecture does not support native floating point operations, the Simulink/Stateflow specification must be correctly and efficiently translated to the fixed point notation of the target micro-controller. Since, even the state-of-the-art transformation algorithms cannot formally proves the equivalence of the two representations, the float-to-fixed point transformation requires to be verified by simulation. The last refinement produces a model ready to be automatically translated to C code. In particular, it requires the definition of all the target dependencies, such as: target software architecture hooks, target instruction set architecture (ISA), RAM and ROM mapping.
At each refinement step, information about the target platform has been considered, from the supported data type to the semantic and syntax of the provided services (such as input/output). During the conception of the solution (function and operation level), the designer is free to use the full expressiveness of the capturing environment (Simulink), while during the refinement, the models must satisfy a set of design rules, checked by a model style checker. The design rules have been necessarily introduced to limit the execution model (not formal) to a known model of computation. This guarantees the existence of the transformation to C code and increases its correctness by construction, hence simplifying the validation phase. In particular, a single control algorithm is typically refined to an extended finite state machine. While a set of different algorithms is composed with the globally asynchronous locally synchronous (GALS) model of computation as defined in (Balarin, 1997). This model of computation with an additional scheduling of the components completely defines the functionality of the control application.

As already mentioned in the introduction, the software platform must accommodate the automatically generated software components as well as the set of hand written ones. The software platform is mainly written manually and is composed of RTOS, communication services, BIOS and device drivers (see Figure 1). It is important to clarify that the software platform defines:

- a language (or set of ) to program the platform (C and ASM)
- a rigid set of structural rules: naming convention, hierarchy and visibility rules
- a clear execution and communication semantic: process scheduling , system events and process communication
- a well defined set of services, e.g. OSEK, OSEK Com, etc

The model compiler generates the C code according to the characteristic of the select software platform.

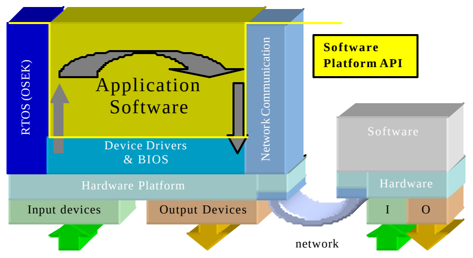

Figure 1 ECU Architecture

The entire software platform has been specified across all the products segments and cover different product generations.

## 4. TARGETLINK: AUTOMATIC TRANSLATION FROM MODELS TO C CODE

The translation of Simulink/Stateflow models to C code is performed by TargetLink from dSPACE (Hanselmann, et al, 1999). This model compiler creates the software modules (component level) from refined algorithms, as described in the previous sections. Code efficiency is one of the primary requirements for a model compiler. Another important necessity is process efficiency and flexibility. These three features of TargetLink, as well as several others, make it a tool which is highly integrateable within the environment of model-based design methodology.

### *4.1. User-Guided Model Refinement*

As outlined in the previous section, the center focus of working with a model compiler is model refinement. The Simulink and Stateflow models represent the complete functional specification of an algorithm. This specification has to be prepared for code generation, which basically means that data for implementation has to be added to each block and each subsystem of the model specification.

If micro-controllers with fixed-point arithmetic are being used, then variable scaling is the inevitable first step of the refinement process. Further data for variable definitions and declarations have to be specified, such as names, storage classes, scope and other attributes. Function and task partitioning takes place, and some model optimizations can be carried out to support efficient code generation. The designer has different options for entering the code generation related data. One of these is based on a data dictionary: a common storage location for data objects that are being referenced in models and are used for code generation.

Data dictionaries typically store data objects such as the definition of global variables, OS messages, function interfaces and macros. In traditional manual programming, such data is directly coded in C. This is why data dictionaries are seldom used in manual programming. However, together with a model-based design methodology, data dictionaries can indeed be used and are very beneficial. Their major advantages are:

- A common project data source for large ECU projects.
- Support for multi-model projects, allowing projects to be spread among different models, with their shared data objects stored in one location.
- Protection of intellectual property by a systematic separation of the model-based algorithm specification from the data dictionary-based implementation specification.
- Variant handling by switching data dictionaries or branches of one data dictionary in the background.

A dialog-guided model refinement process, utilizing the user interfaces as described above, relieves the implementation specialist from a lot of tedious detailed work. He still specifies the implementation on a bit-accurate level, but does not write C code any longer. This reduces implementation errors and significantly increases software quality.

### *4.2. Configurability of the Code Generation Process*

When code should be implemented on a production ECU, many details about the code become quite important. Model refinements are not just limited to data typing and fixed-point scaling. Properties that directly impact programming language aspects of the generated code are equally important. Naming conventions have to be followed, variables have to be properly declared and put into the right memory sections, there need to be efficient ways to link to external code, and the code output format should comply to company-specific standards.

Generated code for real production projects will always have to interface external code, specifically to components of the lower software layers or to the proven legacy code of the application layer. TargetLink has a wide variety of specification means on the block diagram level to easily interface with non-generated code. In particular, these are:

- inclusion of existing header files
- use of external global variables
- use of externally defined macros
- call to imported functions
- call to access functions or macros
- definition of Custom Code blocks which contain hand-written C code

Related to this is TargetLink's full support of the OSEK/VDX operating system, see (OSEK/VDX, 2001) and (Thomsen, 2002). TargetLink provides an extended library of special OSEK blocks which make the operation of system objects, such as tasks, alarms or critical sections, available at the block diagram level.

The code can be generated in a format that exactly matches company-specific C code templates. Code output formatting is possible through XML and a XSLT style sheet. This allows the user to define the format of file and function headers, the format of code comments and the inclusion of specific header files. Furthermore, TargetLink-generated code complies with the MISRA C standard, see (MISRA, 1998) and (Thomsen, 2002).

The link between the code and the calibration tools is parameter description files which are standardized by the ASAM-MCD 2MC standard (formerly called ASAP2) see (ASAM-MCD 2MC, 2000). Modern code generators can all generate this format. Should there be calibration systems in use which apply a proprietary standard, then a TargetLink-generated ASAP2 file can be post-processed within the MATLAB environment to any other format.

### *4.3. Simulation-Based Testing and Calibration*

Code generators and simulation environments complement each other in an almost symbiotic relationship. An integrated environment, such as Simulink together with TargetLink, allows a variety of significant development tasks to be completed. Simulation results are used for:

- automatic fixed-point scaling
- code testing and verification
- benchmarking
- model-based automatic calibration

Although code generators work virtually flawlessly in comparison to manual programming, the generated code still needs to be tested. The strength of an integrated environment is that code tests are performed in the same environment that was used to specify the underlying simulation model. Functional identity is achieved when simulation results match. The validity of tests is documented by C1 or C2 code coverage. TargetLink provides the environment for a 3-step verification process, which shows that the model and the generated software components have identical functions.

The first step of this verification process is called model-in-the-loop simulation (MIL). It captures the specified behavior of the model by recording block output and block state data to an internal data server. The minimum and maximum values are used for the automatic scaling of fixed-point data types. The

traces from MIL simulation are the basis for the subsequent steps.
Software-in-the-loop simulation (SIL) is the next step where the code is generated and placed into the same simulation environment. TargetLink does this automatically in the background. The user still sees the block diagram in the model. However, the corresponding code is executed during simulation. The simulation plots should be highly identical when compared with the results of MIL simulation. If they are not, then the model was probably insufficiently scaled, or the generated code is incorrect.

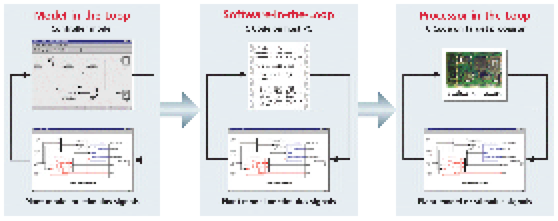

Figure 2: MIL, SIL and PIL simulation modes: a three-step process to verify generated code

Code that runs correctly on the PC can still cause trouble on the target processor. Therefore, the final checks need to be done with processor-in-the-loop simulation (PIL). An off-the-shelf evaluation board is connected to the host PC; the generated code is compiled with the target compiler and downloaded to the board. TargetLink manages communication between the host PC and the processor board. All of these activities are automated without user interaction. The simulation results are plotted in the same plot windows that were used by the other simulation modes. If plots from the PIL simulation deviate from those in the SIL simulation, then the most likely cause is either a problem with the target compiler or a problem with the processor.
If the plots match each other and the C2 code coverage was about 100%, then the functions of the generated software component is with a high level of certainty equivalent with those in the specification model. This 3-step simulation approach is easy, intuitive and quick – and, as a result, it is a safe testing method.
PIL simulation can also be used to profile the generated code and to further refine the implementation. During simulation, TargetLink automatically measures the execution time and stack consumption of the generated C functions directly on the target processor. Furthermore, code summaries list the RAM and ROM usage detailed for each function. These features allow the user to quickly try out implementation options, immediately measure the impact of the change on the generated code, and make logical implementation decisions for the most efficient implementation of a software component.
Methods for automatic, model-based calibration are becoming increasingly popular. This trend is caused by the continuously improving accuracy of simulation models for automotive subsystems (e.g., engine models or vehicle dynamic models) and constant growth of computing power. Code generators like TargetLink are prepared to meet this challenge. The simulation modes (MIL, SIL and PIL) are API supported, meaning that tool automation can be done with MATLAB's M-script language. TargetLink's SIL and PIL simulation modes allow the user to change data logging options for global variables without recompiling the code. Direct memory access for parameter tuning is possible, even while the simulation is running. This allows users to create optimization strategies for automatic model calibration and to set up an autonomously working calibration system.

## 5. GASOLINE DIRECT INJECTION CASE STUDY

The model-based methodology has been applied to the gasoline direct injection (GDI) engine control. The most innovative concept of a GDI engine that requires new control algorithm is the ability to inject the gasoline directly in the combustion chamber trough an injector. This capability removes the restriction of introducing fuel into the combustion chamber only when induction valves are open, and as a result a GDI engine has better performance and fuel economy and less pollution than traditional gasoline one. The complexity of a GDI engine resides to the need of a more precise control on the fuel-air mixture and combustion. In particular, the system differs from traditional one for the presence of a high-pressure fuel pump (to inject fuel directly into the cylinder), injectors that support a high pressure flux of gasoline and generate adapted spray pattern (Pontoppidan and Gaviani, 1997) an intake port that generates the desired vortex in the combustion chamber, a more complex treatment of exhaust gas. The engine runs with two different type and independent combustion modes: homogeneous and stratified. The former being the traditional combustion mode, the latter presenting a non homogeneous air to fuel ratio (A/R) in the combustion chamber.

The high complexity and the presence of innovative control algorithms make this system a perfect case study. Moreover, the strong dependency between the design of the combustion chamber and the design of the combustion control algorithm requires a deep analysis (prior implementation) and a strong interaction, with exchange of models, between car makers and sub system suppliers. The most important component of a GDI engine management system are: the air control by electronic throttle (DBW), variable valve timing or exhaust gas recirculation (EGR), self diagnosis for sensors and actuators and for emission regulations (EOBD/OBDII), safety control, exhaust emission control with Lambda sensor and linear lambda sensor to control the A/R, NoX sensor for NoX trap control, high pressure injector control.
Starting from the car maker requirements, the system has been decomposed and refined into 98 operations, 75 of them have been completely modeled and automatically translated to C code via TargetLink, resulting in 94% of the total application code. The application component accounts for more than 2/3 of the total lines of code, while the remaining part is

related to the software platform. As expected the system design cycle has been reduced compared to the traditional approach. However, the time of the first design cycle was comparable (or even longer) of the traditional one. This was mainly due to the complexity to harnesses the design process and builds the modeling library. The subsequent design cycles have been drastically faster and the final number of design cycle has been reduced.
The introduction of a well defined software platform has been instrumental to manage a variety of engine configurations and a not fixed hardware platform. The number of cylinder has been set from 2 to 6 and the list of sensor and actuators has been adapted to the different engine configurations. Moreover, the set of custom ICs has been replaced without adapting any control models. This *flexibility* has been obtained by the adopted methodology that encapsulates these variants with the minimum amount of software differentiation. In particular, all the hardware and engine configuration variants have been captured in the lower level of the layered software architecture, respectively BIOS and device drivers, while the software application has been composed with the automatically generated or hand written software components. This flexibility introduced by the software layering has also encapsulated the evolution of the ECU from the first hardware prototype (A) to the start of production.

## 6. CONCLUSION

The methodology described in this paper has shown is these years of use in the GDI product development its validity and the maturity level of the tools. The application to a real product has shown the improvement of the time-to-market and the capability to cope with the complexity of modern power-train controllers. The TargetLink model compiler has been instrumental in implementing our model-based design methodology.
Nevertheless, some improvement must be done to better cover some important design aspects, such as requirements tracking at the model level, unified framework for refinement, model protection, etc.

In the future we expect:

- to have more data related to the process to qualify the real advantages of the approach;
- to start a formalization of architectural aspects, such as the description of the software platform with architectural description language and UML;
- to improve the integration in the design chain.

We plan to extend the use of the model-based design methodology to other power-train application and to exploit the new coming features of the new releases of TargetLink. The application of the model-based design methodology is expected to drastically decrease the time-to-market of new power-train controllers. In conclusion, the definition of a common design methodology and tool chain is the key of success in cooping with the complexity and constraints of the design of a modern engine management.

## 7. ACKNOWLEDGMENT

We would like to thank Alberto Sangiovanni-Vincentelli and A. Balluchi from PARADES for the main conception and contribution to the methodology. Cesare Pancotti, Stefano Monti, Giovanni Reggiani, Walter Nesci and Paolo Marceca from Magneti Marelli Powertrain for their contribution to the GDI project and Software Architecture.

## REFERENCES

ASAM-MCD 2MC (2000), Version 1.4
Balluchi A, et al. (1999).
Functional and Architectural Specification for Power--train Control System Design.
In *Proc. 2nd IFAC Conference on Mechatronic Systems*.Berkeley, CA, USA
Balarin F. (1997)
The POLIS Approach
In *Hardware-Software Co-Design of Embedded Systems*. Kluwer Academic Publishers
Burch J. R, et al (2002)
Modeling Techniques in Design-by-Refinement Methodologies
In *Integrated Design and Process Technology*
dSPACE (2002)
TargetLink
http://www.dspace.de
Hanselmann, H.Kiffmeier, U.Köster, L.Meyer (1999)
Automatic Generation of Production Quality Code for ECUs, In *SAE Technical Paper* 99P-12
The Mathworks (2002)
MATLAB/Simulink, http://www.mathworks.com
The Metropolis Project
http://www.gigascale.org/metropolis
MISRA (1998)
Guidelines for the Use of the C Language
OSEK/VDX Operating System (2001), Version 2.2
Pontoppidan, M., Gaviani, G. (1997)
Direct Fuel Injection, a Study of Injector Requirements for Different Mixture Preparation Concepts, In *SAE Paper*, No: 970628
Thomsen T, Stracke R, Köster L.(2001)
Connecting Simulink to OSEK: Automatic Code Generation for Real-Time Operating Systems with TargetLink
*SAE Technical Paper* 01PC-117
Thomsen T (2002)
Integration of International Standards for Production Code Generation
In *SAE Technical Paper* 2003-01-0855
Vincentelli A. Sangiovanni, A. Ferrari (1999)
System Design: Traditional concepts and new paradigms. In *Proceedings of ICCD*
The SEA Consortium (2003): J.Y. Brunel, P. Buratti, W. Damm, et al. Seamless Design Flow for Automotive Electronic Systems with Architecture Design Exploration Emphasis. Proposal for an Integrated Project in the 6th Framework Programme of the European Commission.