Telematic Device Development Based on Framework for Embedded Systems

LEOPOLDO R. YOSHIOKA, CLAUDO L. MARTE, CAIO F. FONTANA
Polytechnic School
University of Sao Paulo (USP)
Av. Prof. Luciano Gualberto, 158, SAo Paulo, SP
BRAZIL
leopoldo.yoshioka@usp.br, claudio.marte@usp.br, caio.fernando@unifesp.br http://www.poli.usp.br

MARCIO C. OLIVEIRA, EDGAR T. YANO
Department of Computer Engineering
Aeronautical Institute of Technology (ITA)
Sao Jose dos Campos, SP
BRAZIL
mcamargooliveira@gmail.com, yano@ita.br http://www.ita.br

Abstract: - The proportion of electronic devices in automotive systems is increasing, representing something about 25-35% of the total production cost of a vehicle, depending on the model. In particular, the role of embedded systems in performance and quality of the vehicles is becoming critical. The effort required for the design and implementation of hardware and software for automotive applications is growing exponentially, while development cycles must be minimized due to the time-to-market constrains. This article presents a framework for development of automotive embedded systems, with the following objectives: 1) to propose a unified platform of hardware and software on which work will be developed, 2) to define a systematic process of development based on test management and 3) to evaluate the effectiveness of the proposed method through a case study of the development of a electronic module for telematic application.

Key-Words: -Telematic Device, Embedded System, Framework, Hardware Platform, Software Platform, Controller Area Network, CAN.

1 Introduction

An embedded system basically consists of a microprocessor system with few dedicated functions, usually with real-time computing constraints. In order to propitiate more comfort, safety and operational performance it is increasingly present in automotive applications. Currently, the electronic components represent about 35% of the production cost of a high end model [1].

The increasing use of embedded systems came from two factors. First, standardization of intra-vehicular systems architecture, particularly data communication networks known as CAN (Controller Area Network), which allowed the interconnection of various embedded modules in a true network of devices. Second, the availability of a range of high performance sensors, actuators and processors with reduced costs, allowing the onboard electronics to take on various vehicle functions, previously performed by mechanical and electrical systems [2] [3] [4].

Recent laws imposed by CONTRAN (Brazilian National Transit Council) are making RFID system (Automatic Vehicle Identification System – SINIAV) and vehicle tracking system (Integrated Monitoring and Automatic Registration of Vehicles – SIMRAV), among other items, mandatory onboard equipment for new vehicles [5][6]. The implementation of these components is occurring gradually, following a government schedule. This is an opportunity for companies to develop new embedded solutions, since there are not available in the global market yet, OEM products that meet the specific requirements defined for these Brazilian applications.

It should be noted that the development of an embedded system for automotive applications must meet the requirements of quality, reliability and robustness, while they must also be comply with the time-to-market and costs constraints imposed by market competition. Therefore, it represents a major challenge for companies that need to cope with the massive demands of resources throughout the life cycle of product development [7].

ISBN: 978-960-474-330-8

Given this scenario, the ability to carry out the development of embedded systems efficiently becomes critical to business success. Therefore, it is obvious the need to define and implement effective processes, associated with the use of modern development tools. Nevertheless, small size companies, mostly, do not have the culture or not acquired a reasonable level of maturity in terms of techniques for the development of its hardware or software products. Beyond the issues of investment resources, one of the causes for this fact is the difficulty in adjusting the traditional development methodologies to the context of these companies. Concerning to software, another issue is the fact that most of these techniques relates to development for computational platforms with less constraint processing capability, memory space, and operating system [8].

Therefore, this paper proposes a development framework based on a unified platform, combined with a systematic development process to increase the quality and reliability of products through the reuse of hardware and software components, minimizing rework costs and increasing the work efficiency of the development teams.

2. Framework for Embedded System Development

A framework can be understood as architecture developed in order to achieve maximum reuse, represented as a set of abstract and concrete classes, with great potential for specialization [8] [9].

Although this definition is essentially focused on the object-oriented software domain, their concepts can be applied to the development paradigm of automotive embedded systems, creating a scenario that embraces the four pillars involved in developing such kind of system:

- Process development;
- Hardware;
- Software;
- Integration and Testing.

Therefore, within the context of this work we will adopt the following definition:

"A Development Framework for Embedded Automotive Systems is characterized as a well-defined development process, coupled with the appropriated tools for management, development and testing, enabling the implementation, reuse and customization of embedded systems efficiently with low cost, thus preserving the quality and reliability of the final products, ensuring reduced time-to-market."

The increasing adoption of embedded software has led many automakers to consider to deal with subjects that earlier where exclusive to software developers, such as managing multiple vendors systems, an item that is identified as a potential source of problems, and having to consider the use of international software development standards the same way as happens in aerospace industry.

Such initiatives can be seen, for example, in Automotive Spice – a process model reference created jointly by several vehicle manufacturers through the SIG (Automotive Special Interest Group Automotive), with the goal of bringing the best practices defined in ISO/IEC 15502-2 standard for automotive environment. And, also in AUTOSAR [10] which is a methodology where the embedded software is separated into two distinct categories: application and infrastructure. Here, the software components are tailored from the beginning to be interconnected, by means of well-defined ports, independently of the CPU, hardware or type of application.

2.1 Unified Hardware and Software Platform

In the following it is presented the background for hardware and software development platform.

2.1.1 Hardware platform definition

When we are designing the hardware for a particular embedded system, we should not consider that the hardware will only have to attend the requirements of a specific application. We need to expand the scope, and see it as a "Platform" for the development of different future applications.

In order to apply the concept of reusability in the context of the development of a new embedded system, it is essential to have a particular concern regarding that the hardware architecture will be adopted for a particular class of embedded products.

Fig. 1 illustrates a conventional hardware development process, based on the waterfall model, where the development cycles are recurring for each new product being created.

If we choose the hardware architecture considering the specific requirements strongly attached to a particular purpose, it could result in a good final product, and even achieves the purpose for which it was designed with some success. However, it will have a limited life span to its original purpose, reducing the chance of its use in other similar product, thus, requiring a new development cycles for each new product. As showing in the Fig. 1, each design and development cycle of a new product is repeated throughout the life cycle, without substantial reuse of previous solutions and effort.

The proposal to minimize this problem is the incorporation of a hardware platform, where the full development cycle is performed only in the first development. In the following cycles, the design of the hardware platform is reused in different applications,

ISBN: 978-960-474-330-8

with specific configuration or arrangement variations to meet distinct requirements.

A hardware platform is, therefore, a family of architecture that satisfies a set of architectural requirements, imposed to allow the reuse of hardware and software.

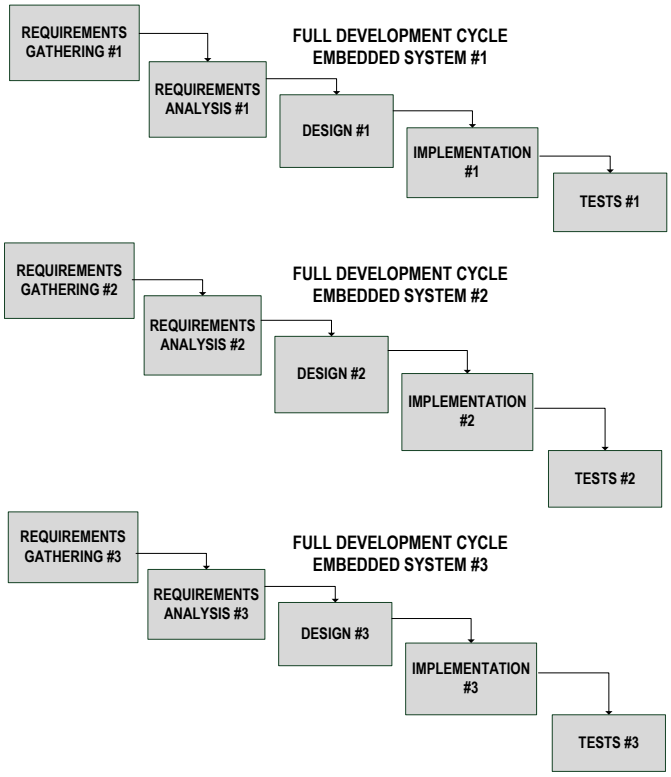

Fig. 1. Hardware development process – repetition of traditional waterfall model for each new embedded system development.

Within this idea, the assembly comprising the platform architecture must cover all possible requirements of the class of products considered. Thus, given a hardware platform, each new product will define an "instance", a sort of "sub-platform" that consists of a subset of the technological possibilities available on the hardware platform, and where all the requirements of the embedded module are being fully complied.

At first glance someone can suppose an increased unit cost of a basic product, which eventually will be using a hardware platform more complex than it needs to have. However, this cost will be diluted by efficiency gain and economies of scale when considered throughout the lifecycle of each new product.

Following this strategy, the proposed framework considers that the full development cycles will only occur at the first time when the hardware platform is designed. For each new embedded system, will exists a reduced cycle where the requirements are analyzed, and it will be generated an instance of the hardware platform in accordance with the needs of the new product.

The platform must be designed in such a way that it covers the key features and future needs envisioned by the company from the point of view of technological resources for embedded modules, such as processing speed, memory (RAM and FLASH), interfaces, power consumption etc.

The choice of a specific hardware component must take into account factors such as availability of development tools, ease of use, expertise of technical team (minimizing any training needs or renewal of the team), component life cycle and technical support provided by the manufacturer.

By defining a platform, you should consider that it must have scalability and needs to be equipped with current technologies available on the market, thus ensuring at least a three year life cycle, without the need of significant changes in the project in the short and medium term.

Fig. 2 in the following illustrates the design and development process of hardware within the context of the proposed reusable platform framework.

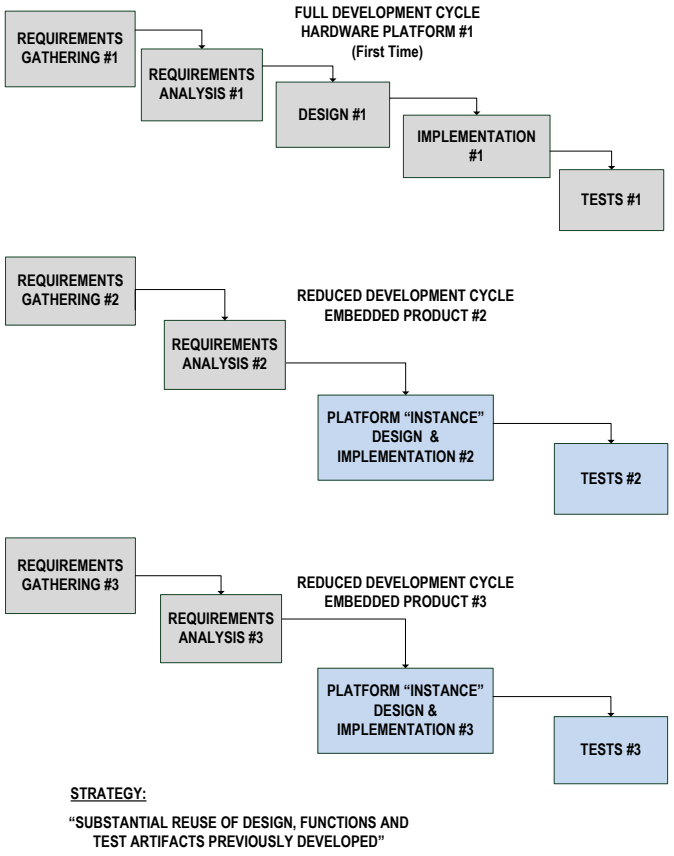

Fig. 2. Proposed hardware development process – reducing the development cycle by introducing platform "instance" concept.

Critical components such as microcontrollers and memories must be chosen preferably within device families that have a broad spectrum of features and

capabilities, while preserving the "pin-to-pin" compatibility so that they can be eventually exchanged for each platform instance, in order to adjust cost requirements. For example, the adoption of a processor with multi task features becomes a key factor for the segment of automotive embedded systems that are based on complex algorithms.

Moreover, a very important factor which should be taken into account for an automotive embedded system is the fact that the devices chosen should have capability to operate in low power consumption mode (standby).

2.1.2 Software platform definition

The software platform should also follow the concept of reusability and scalability, allowing substantial code reuse and adoption of the best practices of software engineering and object oriented model. This results in advantages such as:

- **Increased reliability** – reused software, already tried and tested, tends to be more reliable than new software, once that design and implementation flaws have been cleared;
- **Compliance with standards** – some patterns can be implemented as a set of standardized reusable components;
- **Improve Team Efficiency** – rather than developing the same work again and again, the team experts can focus their efforts on the specific application, thereby reducing development time and costs, and enabling the use of algorithms that have been tested and consolidated by the development team.

A very common scenario, especially in small size companies, is the practice to always implement the embedded system software from the scratch for each new project. This is partly due to the influence of its own hardware, which constantly changes for each new product development cycle [7].

The software architecture shold be based on componentization of modules along all software layers, using the concepts of CBSE (Component Based Software Engineering) [11]. The basic philosophy consists in the implementation of the software systems from pre-existing components instead of creating them from the scratch, i.e., the focus is on reusability.

This architecture promotes several advantages, among which we highlight:

- enables better control of activities like design and coding – because, the scope of requirements is concentrated in small packages grouped in each component;
- allows the outsourcing of software implementation to various suppliers – since the componentization becomes possible to work with isolated artifacts, inter-connectable via a standard interface, which will be integrated to the final embedded software application.

The proposed framework considers that the software will be organized into three distinct layers:

- Application Layer;
- Services Layer, and
- Infrastructure Layer.

First, the infrastructure layer will be responsible for interacting with the hardware, creating an abstraction layer between the hardware and the upper layers, decoupling them from direct contact with the hardware platform in use. It is included in this group all device drivers required for interaction with the hardware platform, including devices such as CPU, memory, IO, modems, GPS modules, network devices etc.

Second, the services layer aggregates the components responsible for the provision of services to the application layer, using processes of the real-time operating system and device drivers addressed in the infrastructure layer. These services include general connectivity elements as serial, network, GPRS, GPS, timers and threads. Also, in this layer will be found the components responsible by the device diagnosis, and by communication services between components.

Third, the application layer is responsible for the business rules, comprising all the algorithms and logic used in the operation of the embedded system according to its application. This layer will be the user of the services provided by the services layer.

The operating system for the proposed software platform is based on Linux, operating in conjunction with real-time microkernel RTLinux [12].

An important feature of RTLinux is that it works on Linux as a system module, operating in HAL mode (Hardware Abstraction Level).

Fig. 5 shows the proposed three-layer architectural model. On the lower level there is the hardware, which in this case is the "instance" of the platform to be used. On the next level there are the software components that build up the infrastructure layer, immediately after the service layer, and at the highest level, the software components that characterize the application layer, including the algorithms and business rules of the embedded module.

Fig. 6 illustrates the component-based development process model.

All phases receive input artifacts and generate output artifacts. For each type of input / output there is a standard set of documents and artifacts to be generated, based on the tools available in the market.

In the following items we describe the development process phases:

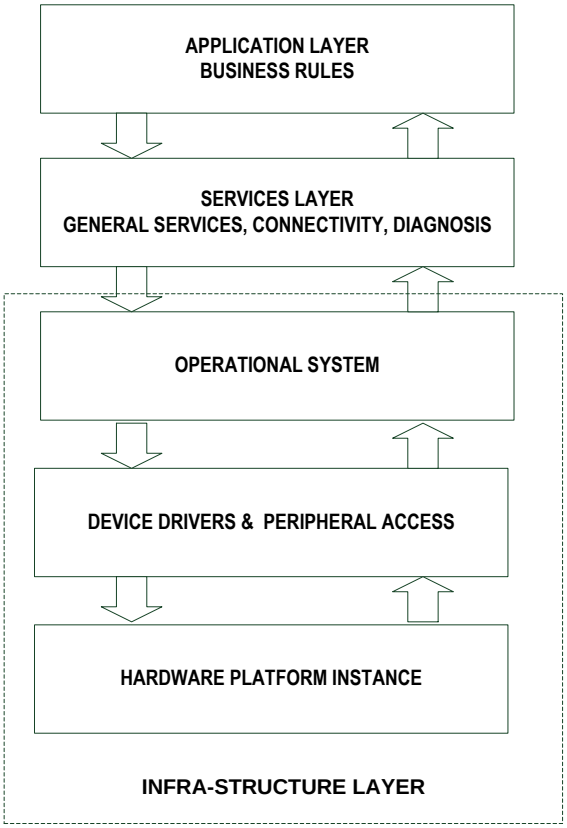

Fig. 5. Proposed software architectural model.

a) Analysis

In this initial stage all the features of the product should be discussed by the project and development team, and the requirements listed, detailed and classified according to their nature.

The input of this phase will be composed by technical scope commercially aligned with the requesting client, in this case an auto maker. At this point, the information will still have a high level of abstraction from the viewpoint of the embedded module itself, but it already contains a certain level of technical detail regarding aspects of the module´s operation and integration with the vehicle, such as the interfaces with the outside world, communication protocols to be used, performance, reliability and security requirements.

The final result of this phase will be the generation of a specification document with detailed technical requirements, which will be used as a guideline for the next phase, the architectural design.

b) Architectural Design

Architectural design will details the componentization of software within the context of layers defined in the framework. It will be defined the components and their respective interfaces. In this phase it is performed the final software assembly activities from the software components, either from the coding of new components that will be developed, or eventually already ready components that can be qualified, adapted and reused.

The output of this phase will be the Architectural Design Document, which will consist of UML 2.0 diagrams containing static and dynamic representations of the embedded software.

c) Component Qualification

It consists in evaluating the applicability of a component to the final system where it will be used. This activity is applicable when it is being considered the use of some ready component (reuse) in a given system. It will be evaluated aspects such as functionality, usability, reliability in order to approve certain component for use. Moreover, essential items such as the adherence of the component to the Component Framework in use are also evaluated.

d) Component Adaptation

It consists in evaluating the coupling between various components eventually developed for use in different contexts, within the context of the new software which will be integrated (reuse). The goal of this activity is to ensure that conflicts between components will be worked and minimized, ensuring among other things that all they work with the same Component Framework. The idea is to eliminate any undesirable characteristics of a particular component, making it compatible with the framework and component model adopted.

e) Component Engineering

This activity consists in the implementation of new components, which are being first time developed for use in an embedded application, being designed and coded with focus on future reuse, upon which they may eventually go through the steps of Qualification and Adaptation.

f) Component Composition

This activity consists in integrating the various components to create the final application. In this step the components are interconnected through the component framework providing services to each other via the available interfaces. Typically, the composition may be of three different types: i) Hierarchical (a component directly calls the services of another component) ii) Sequential (component services are executed in sequence), or iii) Additive (two or more interfaces components are composed to create a new component).

g) Validation Tests

This activity consists in the validation test of the individual software components or final software. It should be noted that the validation tests brings forth a wide range of variations of test cases due to various combinations of applicability of the module and the environment in which it will operate.

ISBN: 978-960-474-330-8

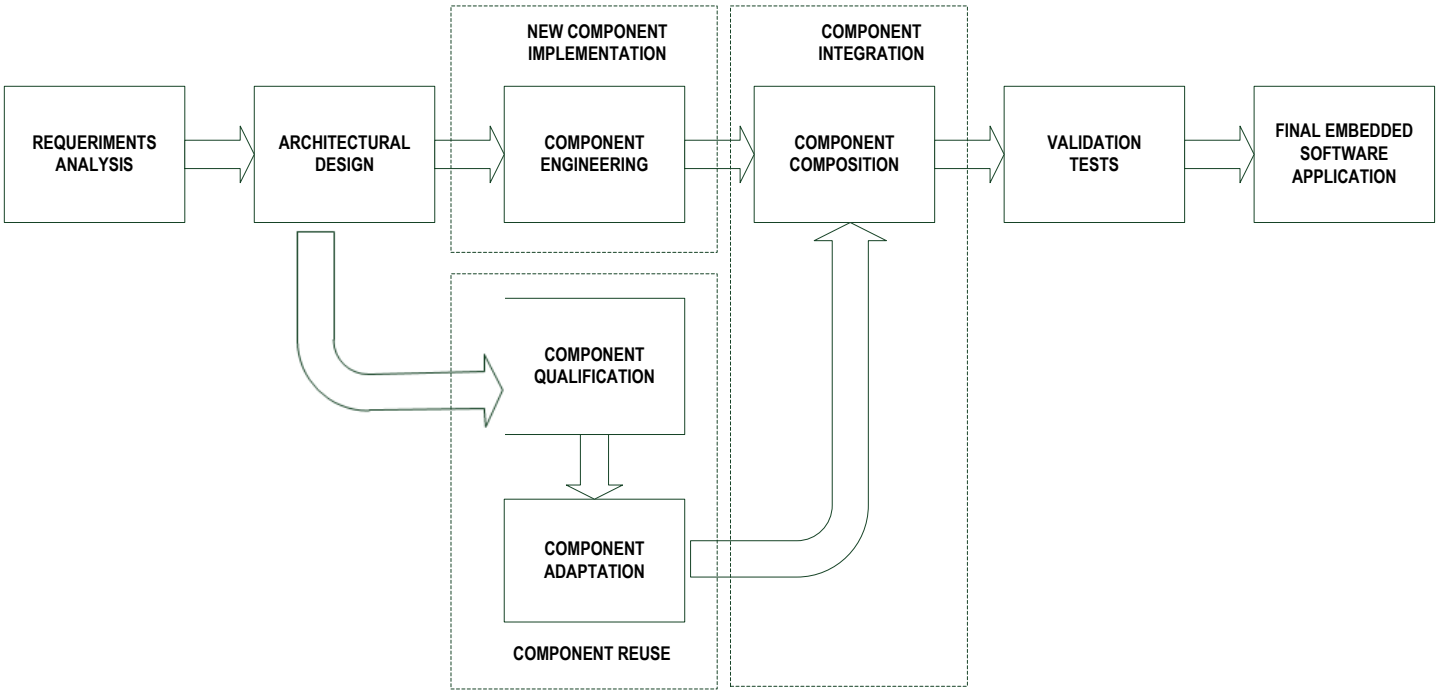

Fig. 6. Life cycle of the component-based development process model.

3. Case Study – Development of the Telematic Module

In this article, it will be taken as an example the development of a telematics control unit (TCU) adherent to SIMRAV specification [5] [13].

The TCU is an electronic device capable of performing the functions of vehicle tracking and blocking. The blocking function, which prevents vehicle operation can be enabled / disabled remotely or locally (by the own device under special circumstances, or by the service operator). The tracking function sends out data regarding the positioning coordinates and security-related events to the Monitoring Service Provider (TIV). Fig. 8 and Fig. 9 illustrate TCU module used in this Case Study.

3.1 Functional description of the TCU:

It is described in the following the main functional elements that composes the TCU module.

- **Satellite Signals Reception Module**: it consists of an antenna and a receiver. It has the function to capture signals from a constellation of global positioning satellites (GPS) and determine the position of the vehicle.
- **Bi-directional Communication Module**: it consists of an antenna and a communication unit. Its function is to send the localization data and events from the vehicle to a TIV, and to receive commands from it, beyond data related to the equipment configuration.
- **Vehicle locking and Management Module**: it is responsible for the integration of all other functional modules. It receives information from satellite signals receiver, captures events information from the vehicle´s interfaces, receives data from the bi-directional communication module, and also manages the equipment features, and the backup battery module.
- **Inputs Interfaces**: they read the sensors states installed on the vehicle (ignition, panic buttons, doors status, brakes pedal, etc.). There are six digital inputs available.
- **Output interfaces**: they allow the activation of external devices like the vehicle locking system or a siren. There are four digital outputs.
- **Backup Battery Module**: it ensures the functioning of the equipment in case of main power failure (vehicle battery). It is capable to operate instead the main energy source for at least two hours, enabling the device to keep the communication with the TIV, thus enabling alarm messages sending and the reception of remote blocking commands.

ISBN: 978-960-474-330-8

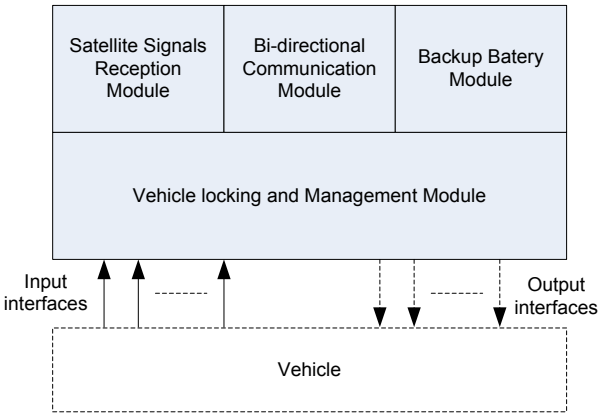

Fig. 8. Functional block diagram of the TCU module.

Fig. 9. Telematic module used in the Case Study.

4.2 Application Variability:

In this context, we will consider the following situation regarding the TCU applicability:

1. Models of vehicles where the module will be applied: Model A, Model B and Model C.
2. Data communication service providers: Operator 1, Operator 2, Operator 3 and Operator 4.
3. Monitoring service providers: TIV 1, TIV 2, TIV 3 and TIV 4.

The following environmental variations would be present in a typical scenario of operation:

1. Status of the GPS signal:
   - Module with permanent sight of satellites
   - Module with intermittent sight
   - Module without sight
2. Status of GPRS data channel:
   - GPRS constant signal
   - GPRS intermittent signal
   - No GPRS signal
3. Vehicle speed
   - V = 0 km / h
   - V > 0 km / h
4. Vehicle ignition status:
   - On / Off

The following configuration changes would be present in a typical scenario of operation:

5. Localization function for TIV:
   - On / Off
6. Localization function for local fleet management system:
   - On / Off
7. Status of tracking service with a TIV:
   - Activated (service contracted)
   - Not activated (service not contracted)

The Table 1 represents the total number of variables involved, and the range of possible values for each variable.

Table 1. Paramenters and values involved in the TCU's tests

Paramenter	Range of values
Vehicle models	1 to 3
Mobile Service Operators	1 to 4
TIV's	1 to 4
GPS signal situation	1 to 3
GPRS link situation	1 to 3
Vehicle speed	1 to 2
Vehicle ignition status	1 to 2
TIV's localization function	1 to 2
Local localization function	1 to 2
TIV's association	1 to 2

It should be noted that the number of the combinations of the functions and situations to be tested results in a huge number of test cases, which could derail the development due to the high number of hours and personnel necessary to cover all combinations possibilities.

In the previous example, if we consider total variation for each item From the Table 2, if we consider the number of variations for each item, applying the rule of the product, we will note that the number of test cases (NCt) is given by:

$$NCt = 3 \text{ x } 4 \text{ x } 4 \text{ x } 3 \text{ x } 3 \text{ x } 2 \text{ x } 2 \text{ x } 2 \text{ x } 2 \text{ x } 2$$

$$NCt = 13284$$

ISBN: 978-960-474-330-8

As seen, we have to perform 13,284 test cases in order to cover 100% of the test possible combinations.

Faced with a situation like this, where there are a huge number of variants to be tested, it is necessary to select a subset of combinations which facilitates the testing in accordance with the available resources [14].

Here, we will use pairwise testing technique [15], which resides in the idea not to try to test all test case variables (functions) and all possible values they can take, but to test all pairs of variables (functions).

The proper functioning of the pairwise technique lies in the fact that most of the defects is a "single defect mode", i.e,. a function under test simply does not work and any test on it will find the defect, or defects are "double mode defects ", ie it is a given pair function / module with another function / module that causes the defect even though all the other pairings work well [15]. This technique significantly reduces the number of test cases that must be created and run.

There are two techniques for the application of pairwise testing:

- Orthogonal matrices
- AllPairs Algorithm

Here we will use the orthogonal matrix technique [16].

3.3 Application of the Orthogonal matrix technique

An orthogonal matrix is a two-dimensional matrix (elements 1 to n1, 1 to n2 1, ..., 1 to nm in each column) with the following properties:

- Choosing any two columns of the matrix, in each pair of columns all combinations of pairs will appear.
- If there are n repetitions (n = 1 ... N) of a pair, in the pair of columns, these pairs will appear repeated, in the equal number, in all pairs of columns.

In the following it is presented an example that demonstrates these properties:

$$M = \begin{Bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{Bmatrix} \qquad (1)$$

It should be noticed that in each pair of columns appear the following pairs: {1,1}, {1,2}, {2,1} and {2,2}.

The orthogonal matrix notation is:

$$L_4(2^3) \qquad (2)$$

Where the number "4" represents the number of rows of the matrix, the number "2" is the maximum variation in the possible values for each variable, and the number "3" the number of columns of the matrix, which symbolizes the number of variables under test.

In practical cases, the matrix will not have in each column (each variable) the same maximum number of range values. In this case, the matrix is called "Mixed Orthogonal Matrix" and the notation will be in the following format:

$$L_{18}(2^1 3^7) \qquad (3)$$

The above example is an orthogonal matrix which there is one column (variable) with a maximum value 2, and 7 columns (variables) with maximum value 3.

The application of the technique of orthogonal matrix to apply pairwise testing should follow the following steps:

- First, identify the variables (functions). In the case of the TCU, we will have 10 variables.
- Determine the maximum value of the range of values for each variable. In the case of the TCU module, we have two variables with maximum 4, 3 variables with maximum value 3, and 5 variables with maximum 2.
- Determine the orthogonal matrix which represents the situation under examination. In the TCU module example, we would have the following perfectly orthogonal matrix (which covers exactly the number of variables involved and the range of values of each variable):

$$L_x(4^2 3^3 2^5) \qquad (4)$$

- Mathematically doesn't exist orthogonal matrices for all cases, so we need to use Taguchi's orthogonal matrix selection table [16], in order to locate an Orthogonal Matrix closest to the case under analysis. For this example, the Taguchi's best suited matrix would be the following:

$$L_{32}(4^{10}) \qquad (5)$$

Applying the values obtained in the orthogonal matrix, we have 32 lines that correspond to 32 test cases, and in the columns, 10 variations (VAR1, VAR2, ... VAR10), representing the 10 variables involved.

The technique results in a list of 32 test cases. It should be noticed that original test case list have 13,284 items. It represents a substantial reduction in labor, time and resources for system validation.

4. Conclusion

The proposed framework allows embedded systems developers to move from their traditional process to an approach based on software and hardware reuse. The componentization of software associated to the concept

of hardware platform, enables the development of automotive embedded devices with cost effective, quality assured, and the timing needed to meet the market opportunities. The use of dedicated test techniques such as pairwise testing enables validation activities, keeping the formalism necessary to ensure the product quality, while minimizing costs involved in this critical step.

Acknowledgements: The authors thank the COMPSIS Computadores e Sistemas Ind. Com. Ltda, for the opportunity to conduct this research.

References:

[1] J. Yu, B. M. Wilamowski, Recent Advances in In-vehicle embedded systems, in IECON 2011 – *37th Annual Conference on IEEE Industrial Electronics Society*, 2011.

[2] B.C. Pinheiro, *Real time control for automatic parking sytem*, Master Thesis, UFSC, 2009.

[3] K.H. Johansson, M. Törngren, M., L.Nielsen, Vehicle applications of controller area network. In: *Handbook of Networked and Embedded Control* Systems. p. 741–766, 2005.

[4] Y.J. Choi's et al, A study of HMI on in-vehicle elematic System. Proceeding of the 5th WESEAs International Conference on Applied Informatics and Communications, 2005, pp.281-283.

[5] BRASIL, CONTRAN, Resolution No. 212, november 2006 – *Deployment of National Automatic Vehicle Identification System (SINIAV)*, 2006.

[6] BRASIL, CONTRAN, Resolution No. 245, July 2007 – *Deployment of National Automatic Vehicle Monitoring System (SIMRAV)*, 2007.

[7] A.C. Guerra, J.N. Moreno, Best practices for software development in small businesses. *IADIS Conferences Ibero-American*, 2008.

[8] J.C.B. Mattos, L.S. Rosa, M.L. Pilla, M. L.. *Challenges to Design Embedded Systems*, Ed. da Universidade Federal de Pelotas, 2009.

[9] T. Novosel, L.Jelenkovic, Framework for embedded systems development. In MIPRO, *2011 Proceedings of 34 th International Convention*. 2011, pp. 825-828.

[10] AUTomotive Open System Architecture – AUTOSAR: URL:http://www.autosar.org/

[11] M. Mattsson, *Evolution and Compostion of Object Oriented Frameworks*, University of Karlskrona/Ronneby, Department of Software Engineering and Computer Sciente, Karlskrona, Sweden, 2000.

[12] RTLinux Open Source – URL: http://www.osadl.org/Realtime-Linux.projects-realtime-linux.0.html.

[13] L.R. Yoshioka, M.C. Oliveira, M., Micoski, R. D. Costa, Considerations on the Design and Implementation of ACP245 Protocol in the Telematic Control Unit, *SAE Technical Paper 2010-36-0339*.

[14] M. S. Phadke, *Quality Engineering Using Robust Design*. New York, NY: Prentice-Hall, 2008.

[15] K.C. Tai, Y. Lei, A test generation strategy for pairwise testing. *IEEE Transaction on Software Engineering*, vol. 28, 2002, pp. 109-111.

[16] Taguchi Orthogonal Array Selector, http://www.freequality.org/documents/tools/Tagarray_files/tamatrix.htm , accessed in 21/04/2013.