Framework for designing automotive embedded systems based on reuse approach

Leopoldo R. Yoshioka, Marcio C. Oliveira, Claudio L. Marte, Caio F. Fontana, Cledson A. Sakurai, and Edgar T. Yano

***Abstract*—** The use of embedded hardware and software are growing in the automotive applications. However, the development effort required is increasing at the same time that schedule is becoming tight. In this paper we present a methodology for the development of embedded systems suited to the dynamics of the automotive sector. We propose a development framework that adds the concepts of component-base software engineering, reusable hardware platforms and pair-wise testing techniques. The effectiveness was evaluated through a case study of developing a telematic module. Development cycle of automotive embedded system can be accelerated by using framework.

***Keywords*—** Telematics Module, Embedded System, Framework, Hardware Platform, Software Platform, Pairwise Testing Technique, Controller Area Network, CAN.

I. Introduction

An embedded system basically consists of a microcontroller based system with dedicated functions, usually with real-time processing constraints. Nowadays it is increasingly present in automotive applications to provide more comfort, safety and operational performance. The electronic components represent about 35% of the production cost of high end model [1].

The increasing use of embedded systems came from two factors. Firstly, the standardization of vehicular data communication architecture as controller area network (CAN). Secondly, the availability of a large range of high performance sensors, actuators and processors with reduced costs [2], [3].

The development of an embedded system for automotive applications must meet the requirements of quality, reliability and robustness. Also, they must comply with the time-to-market and costs constraints imposed by market competition. Therefore represents a major challenge for companies who need to apply significant resources throughout the product development cycle [4].

There are several initiatives for the standardization of the automotive embedded system development process. For example, a process reference model was created jointly by several vehicle manufacturers through the Automotive **S**pecial Interest Group (SIG) [5]. Their goal is bringing the best practices defined in ISO/IEC 15502-2 standard for automotive environment. Other example is Automotive Open System Architecture (AUTOSAR), where the embedded software is separated into two distinct categories: application and infrastructure [6]. 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.

Given this scenario, the ability to carry out the development of embedded systems efficiently becomes critical to business success. 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. Another issue is the fact that most of these techniques are related to development of computational platforms with less constraint in terms of processing capability, memory space, and operating system [7].

This paper proposes a development framework for automotive embedded systems based on a unified platform of hardware and software combined with a systematic development process. The framework goal is to increase the quality and reliability of products [8].

II. Framework for Embedded System Development

A. Development Framework

A framework can be understood as architecture developed in order to achieve maximum reuse. It is 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 of automotive embedded systems, creating a scenario that embraces the four pillars involved in developing such kind of system:; (1) hardware platform; (2) software platform; (3) development process; and (4) integration and test. Within the context of this work we will adopt the following definition:

Leopoldo R. Yoshioka and Claudio L. Marte are with the University of Sao Paulo, Sao Paulo, SP, Brazil, (e-mail: leopoldo.yoshioka@usp.br).

Cledson A. Sakurai and Caio F. Fontana are with Federal University of Sao Paulo, Santos, SP, Brazil, (e-mail: caio.fernando@unifesp.br).

Marcio C. Oliveira and Edgar T. Yano are with the Aeronautical Institute of Technology, Sao Jose dos Campos, SP, Brazil, (e-mail: mcamargooliveira@gmail.com).

"*A development framework for embedded automotive systems is characterized as a well-defined development process. It is coupled with the appropriated tools for management, development and testing. It enables efficient implementation of embedded systems. Thus, it preserves the quality and reliability of the final products, ensuring time-to-market.*"

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

B. Hardware Platform Definition

In general, for the hardware design of a particular embedded system, we consider that it has to attend specific application requirements. However, if we choose the hardware architecture considering only specific requirements, strongly attached to a particular purpose, it will limit the life span to its original purpose. It reduces the opportunity of its reuse in another similar product. Thus, it is required a full development cycles for each new product, as shown in the Fig. 1, without substantial reuse of previous solutions and effort.

Now, one can consider the hardware as a *platform* for the development of different applications. We will apply the concept of *reusability* to develop a new embedded hardware. In this proposal the full development cycle is performed only in the first development cycle. In the following cycles, the architecture and design of the hardware platform is reused in different applications, with specific configuration or arrangement variations to meet distinct requirements. Therefore, a hardware platform is a family of architecture that satisfies a set of architectural requirements for a particular class of products, allowing the reuse of hardware and software.

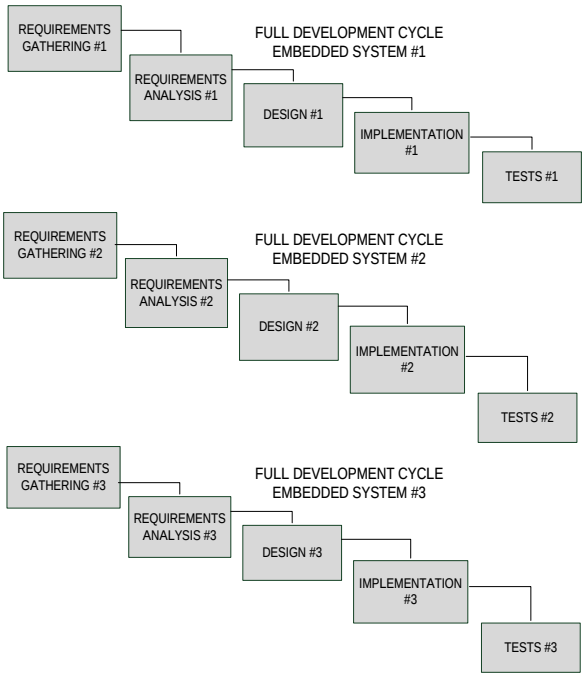

Fig. 1 traditional hardware development process – repetition of traditional waterfall model for each new embedded system development

Within this idea, the platform must have an architecture that meets the needs of a wide range of class of product 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. Fig. 2 illustrates the design and development process of hardware within the context of the proposed reusable platform framework.

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 compensated by development efficiency gain.

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.

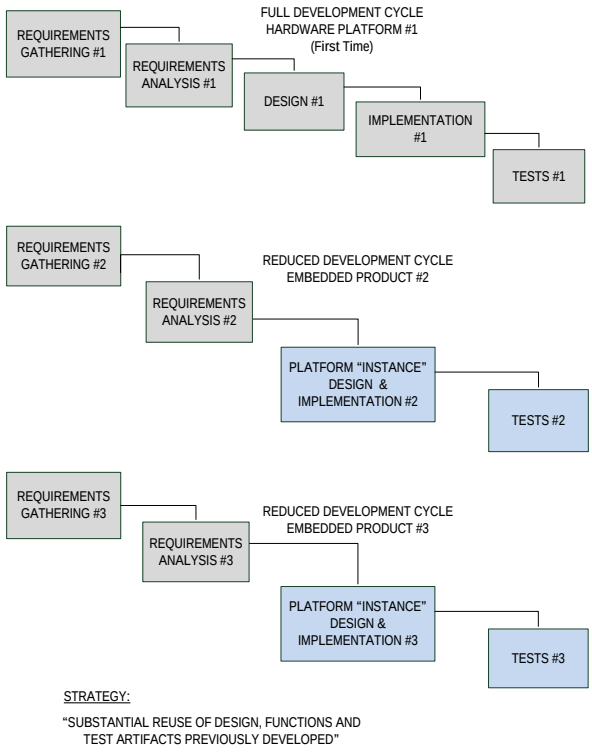

Fig. 2 proposed hardware development process – the development cycle is reduced by introducing platform "instance" concept in the design and implementation phase

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, and power consumption.

The choice of a specific hardware component must take into account factors such as availability of development tools, expertise of technical team (minimizing any training needs or renewal of the team), component life cycle and technical support. 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 satisfy cost requirements. For example, the adoption of a processor with multi task features becomes a key factor for the embedded systems based on complex algorithms [9].

C. Software Platform Definition

The software platform should follow the concept of reusability and scalability. A very common scenario is the practice to implement the embedded system software from the scratch for each new project. This is due to the influence of its own hardware, which constantly changes for each new product development cycle [4].

The software architecture should be based on componentization of modules along all software layers, using the concepts of Component Based Software Engineering (CBSE) [10]. 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.

A software component is part of a system with non-trivial, relatively independent and replaceable characteristics, with the goal of satisfying a clear function within the context of a well-defined architecture. Each component is fully encapsulated, with its inner logic *isolated* from the other components of the embedded software. All communication between components is performed according to defined and standardized interfaces.

The integration of the components depends on two important concepts: component model and component framework [11].

The concept of component model comprises a set of standards and well-defined conventions for a component, defining basically what the component is and how it interacts with other components. In order to components may interact with each other, they must be adherent to the same Component Model.

The component framework is a specific technical solution that allows the components, adherent to a particular component model, work together. A component framework can be compared to a mini operating system, because it manages resources used by components, and provides mechanisms to communicate exactly how an operating system do with the process [12].

Fig. 3 illustrates the concept of Component Framework, where the software components behave as if they were hardware components being *embedded* in a printed circuit board [13].

The proposed framework considers that the software will be

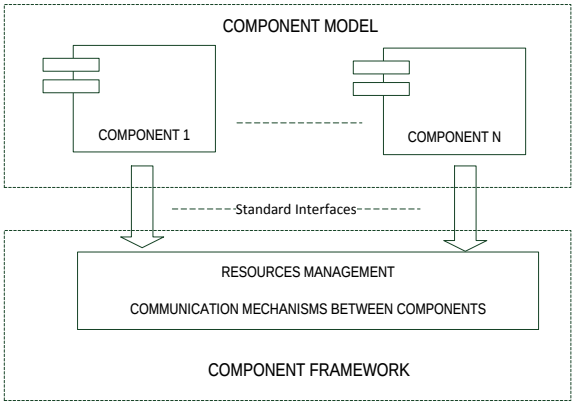

Fig. 3 component framework concept

organized into three distinct layers: infrastructure layer, service layer; and application layer.

First, the infrastructure layer is 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

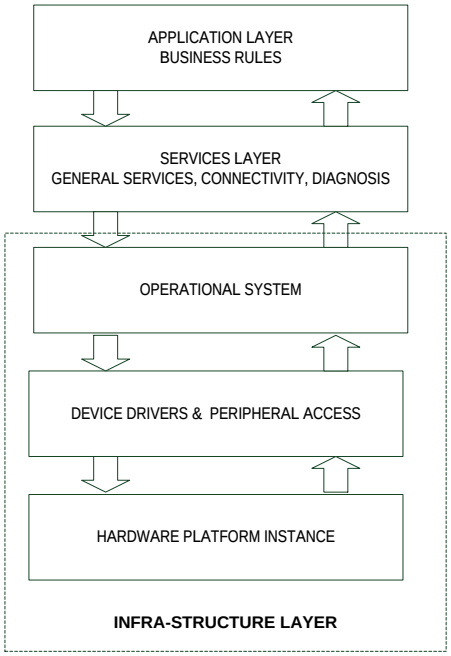

Fig. 4 proposed software architectural model

layer. It is based on real-time operating system and device drivers addressed in the infrastructure layer. These services include general connectivity elements as serial port, network access, timers and threads.

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.

Fig. 4 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

aspects of the module´s operation and integration with the vehicle. Some items include interfaces with other modules, communication protocols, performance, reliability, and security requirements. The final result of this phase is *Software Specification Document* with detailed technical requirements.

2) *Architectural Design* – This phase defines the software assembly activities from the final software components, either from the encoding of new components to be

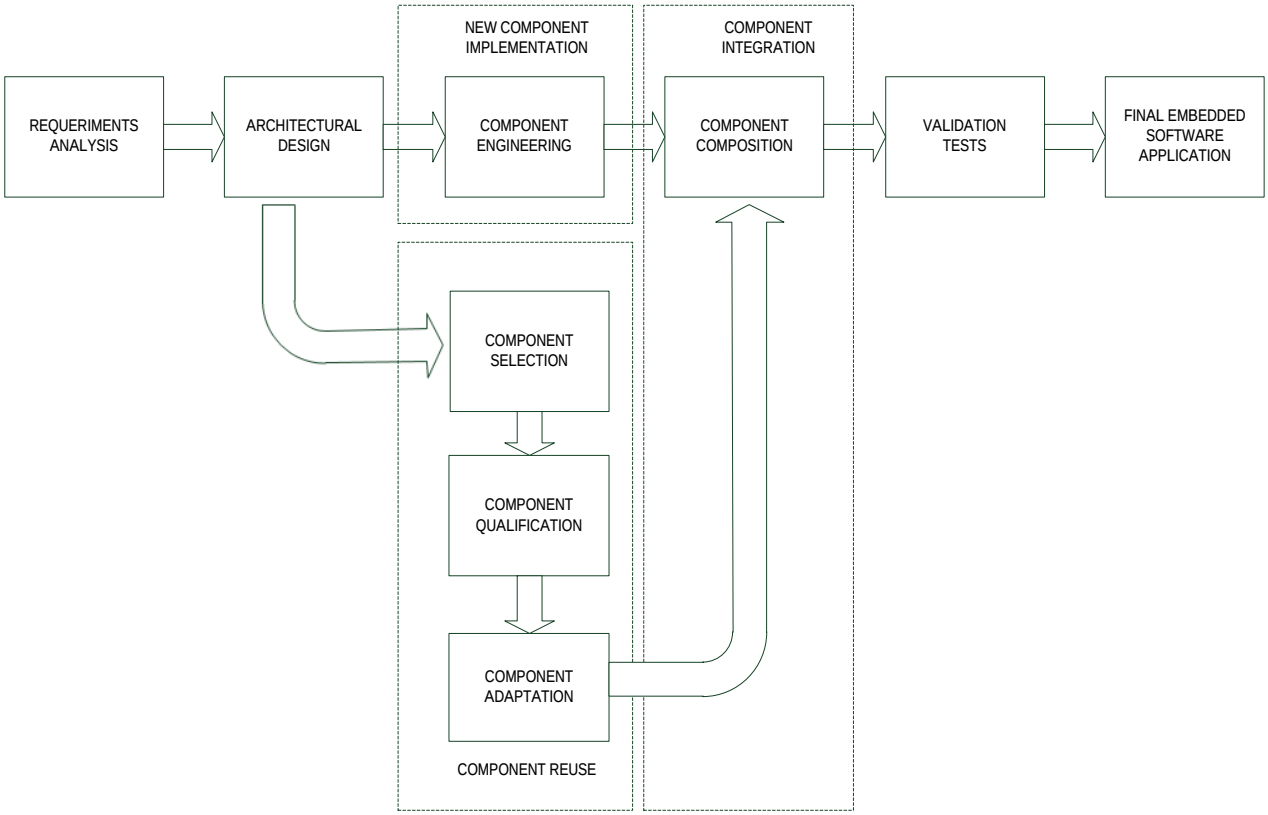

Fig. 5 life cycle of the component-based development process model

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. 5 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 we describe each development process phases:

1) *Requirement analysis* - In this initial stage all the features of the product should be discussed by the design and development team. The requirements are listed, detailed and classified according to their nature. At this point, the information is a high level abstraction from the embedded module point of view. However, it already contains a certain level of technical detail regarding to the

developed, or any ready components that can be qualified, adapted and reused. The output of this phase is the *Architectural Design Document*, which consists of UML diagrams containing static and dynamic representations of embedded software.

3) Implementation or Reuse - depending on the design and the final software components, the development cycle can follow the path of engineering components or component reuse. In some situations there may be a mixed case. In the first case the components will be implemented from scratch. In the second case there will be a reuse of previously developed components after processes including selection, qualification, and if necessary, adaptation.

4) *Component Selection* – can be considered as a process that formalizes the search for components, both in the market or within the organization itself. It includes

assessing a particular component, previously developed and tested, to verify if it is suitable for use in the new system [14]. The goal is to assemble a list of components *candidates* for use in the design. Thus, it creates subsidies for the next step of *qualification*. In some cases the selection process cannot classify any component, ready-made, as being proper for use. If this occurs, it means that the component shall be implemented from scratch. We propose in our framework the use of Weighted Scoring Method (WSM) [15] that is widely used in ranking and decision making, as a tool to systematize the component selection process. This method classifies several options available depending on pre-established criteria, which defines a scale that represents the relative preference of each attribute [16].

5) *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 evaluates aspects as functionality, usability, and reliability. Moreover, items such as adherence of the component to the component framework in use are also evaluated.

6) *Component Adaptation* - it evaluates the coupling between various components of the new software which will be integrated. The goal of this activity is to ensure that conflicts between components will be avoid or minimized. Thus, ensuring that they work with the same component framework. One can remove this way any undesirable features of a particular component, making it compatible with the framework and component model adopted.

7) *Component Engineering* – it consists in the implementation of new component, which is developed for the first time to use in an embedded application. The design and codification should focus on future reuse.

8) *Component Composition* - this activity consists in the integrating of 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 available interfaces. There are three different types of composition: *hierarchical* (a component directly calls the services of another component; *sequential* (component services are executed in sequence); or *additive* (two or more interfaces components are composed to create a new component).

9) *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.

III. CASE STUDY – DEVELOPMENT OF A TELEMATICS MODULE

In this article we present a case study, where we describe the development process of telematic control unit (TCU) compatible with the Brazilian National Transit Council (CONTRAN) specifications [18].

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. 6 illustrates the TCU module functional block diagram and the Fig. 7 shows the picture of TCU considered in the present case study [18].

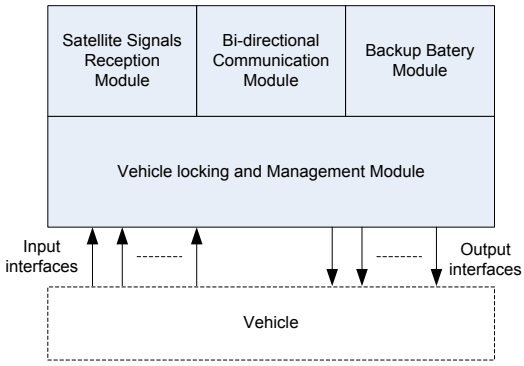

Fig. 6 functional block diagram of the TCU module

Fig. 7 picture of Telematic Control Unit considered in the case study (source: author)

A. Functional description of the TCU:

In the following we describe the main functional elements of the TCU module.

1) *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.

2) *Satellite signals reception module* – it consists of GPS

antenna and a receiver.

3) *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 vehicle monitoring center (VMC). Also, it receives commands from VMC.

4) *Vehicle blocking and management module* - it is responsible for the integration of all other modules. It receives information from GPS receiver, captures events information from the vehicle´s interfaces, receives data from the bi-directional communication module, and also manages the equipment features.

5) *Inputs Interfaces* – they read the vehicle sensors states including ignition, panic buttons, doors status, brakes, and pedal.

5) *Output interfaces* – they allow the activation of external devices like the vehicle blocking system or alarm.

6) *Backup Battery Module* – it ensures the equipment power supply in case of main power failure (vehicle battery). It is capable to operate for two hours, enabling the TCU to keep the communication with the VMC. Thus enabling alarm messages sending and the reception of remote blocking commands.

B. Architectural Design and Component Representation of the TCU

The Fig. 8 shows the representation in UML 2.0 component set for TCU in *architectural design phase*. The components are represented by rectangles and the interfaces between them through connections with *plug* symbols. For this case study we were not used *of the shelf* components. However, we can identify several components that are potential candidates for a future process of selection and reuse. One example is the component responsible for interfacing with the vehicle. This component can be reused in other embedded module, even if it has a purpose other than TCU.

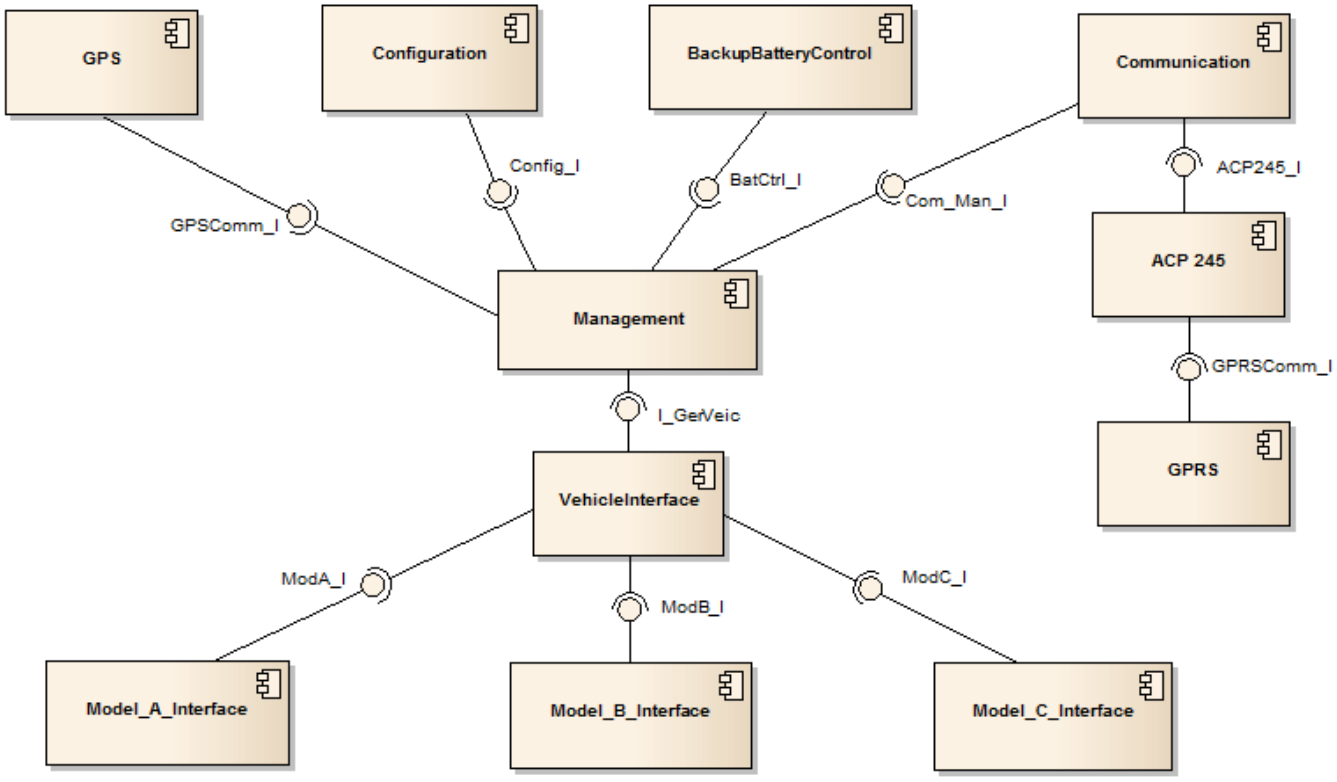

Fig. 8 UML 2.0 representation of the TCU module

In the case of necessity to exchange communication protocol the ACP245 component, shown in Fig. 8, can be replaced by another component available on the market. This increases the possibility of embedded device applications. Similarly, the GPRS component (responsible for communication via a mobile network) could be replaced by satellite communication component. This allows the module to operate in the areas where there is no mobile signal.

C. Operational Scenarios for Validating the TCU

In the following we present the variables involved in the TCU operational scenario for validating the TCU. They are described possible values that these variables can take, properly classified in the applicability of the module,

environmental changes, and configuration changes.

a) Application variability:

1) *Vehicle models* (where the TCU 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: VMC 1, VMC 2, VMC 3, and VMC 4.

b) Environmental variability:

4) *GPS signal status*: (i) permanent sight of satellites; (ii) intermittent sight of satellites; (iii) without sight.

5) *GPRS data channel status*: (i) constant signal; (ii) intermittent signal; (iii) no signal.

c) Vehicle variability:

6) *Vehicle speed:* (i) V = 0 km / h; (ii) V > 0 km / h.

7) *Vehicle ignition status*: (i) On; (ii) Off.

d) Configuration variability:

8) *VMC localization function*: (i) On; (ii) Off.

9) *Localization function* (for local fleet management system): (i) On; (ii) Off.

10) *VMC tracking service status*: (i) activated (service was contracted); (ii) not activated (service did not contracted).

The Table 1 summarizes the TCU parameters and respective range of values for operational context considered.

TABLE I PARAMETERS AND VALUES INVOLVED IN THE TEST OF TCU

Parameter	Range of values
1) Vehicle models	1 to 3
2) Mobile Service Operators	1 to 4
3) VMC Service Providers	1 to 4
4) GPS signal status	1 to 3
5) GPRS link status	1 to 3
6) Vehicle speed	1 to 2
7) Vehicle ignition status	1 to 2
8) VMC localization function	1 to 2
9) Localization function	1 to 2
10) VMC tracking sevice status	1 to 2

From the Table 2 we can calculate the total combination possible for the test cases. Applying the rule of the product, we can see that the number of test cases (NTC) is given by

$$NTC = 3 \text{ x } 4 \text{ x } 4 \text{ x } 3 \text{ x } 3 \text{ x } 2 \text{ x } 2 \text{ x } 2 \text{ x } 2 \text{ x } 2 = 13284. \quad (1)$$

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

It should be noted that the number of the possible combinations of the parameters, considering their range of values, results in a huge number of test cases. This could derail the development due to the large effort need to cover all possible test case combinations.

Faced with this situation, where there are a huge number of variants to be tested, it is necessary to select a subset of combinations. This enables the test execution in accordance with the available resources [14].

The framework proposed in this paper used the *pairwise* testing technique, which is a combinatorial method of software testing [19]. For each pair of input parameters of a system, tests are made for all possible discrete combinations of those parameters. Using carefully chosen test vectors, this can be done much faster than an exhaustive search of all combinations of all parameters, by *parallelizing* the tests of parameter pairs. This technique reduces significantly the number of test cases that must be created and run.

There are two techniques for the application of pairwise testing: orthogonal array and all-pairs algorithm. In this paper we consider only the orthogonal array technique [20], [21].

D. Application of the Orthogonal Array Technique

An orthogonal array is a two-dimensional matrix (elements consisting of 1 to n_1, 1 to n_2 1, ..., 1 to n_m in each column). It has the following properties:

1) Choosing any two columns of the matrix, in each pair of columns will appear all combinations of pairs.

2) 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. Let us consider the matrix M:

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

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

We use the following notation to represent the orthogonal matrix M:

$$M = L_4(2^3). \quad (3)$$

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" is 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". Let us consider the following notation:

$$M' = L_{18}(2^1 3^7). \quad (4)$$

The expression (4) represents an orthogonal matrix which

there is one column (variable) with a maximum value 2, and 7 columns (variables) with maximum value 3.

In the following we present the steps defined by orthogonal array techniques:

1) Identify the variables (functions). In the case of the TCU, we have 10 variables.

2) 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.

3) 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):

$$M = L_x(4^2 3^3 2^5). \quad (5)$$

4) Because mathematically does not exist orthogonal matrices for all cases, we need to use Taguchi's orthogonal matrix selection table [20], 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:

$$M = L_{32}(4^{10}). \quad (6)$$

As shown in Table 2, note that the selected matrix has 32 rows and 10 columns, allowing variation of values of "1 to 4" in 10 variables, which loosely covers the TCU module test cases example.

Applying the values obtained in the matrix we selected 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 notice that original test case was 13,284. It represents a substantial reduction in labor, time and resources for system validation.

IV. 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.

ACKNOWLEDGMENT

The authors would like to thank the company COMPSIS Computadores e Sistemas Ind. Com. Ltda, for the opportunity to conduct this research.

TABLE II ORTHOGONAL ARRAY APPLIED TO THE TCU CASE STUDY

TEST CASE	VAR1	VAR2	VAR3	VAR4	VAR5	VAR6	VAR7	VAR8	VAR9	VAR10
1	1	1	1	1	1	1	1	1	1	1
2	1	1	2	2	2	2	2	2	2	2
3	1	1	3	3	3	3	3	3	3	3
4	1	1	4	4	4	4	4	4	4	4
5	1	2	1	1	2	2	3	3	4	4
6	1	2	2	2	1	1	4	4	3	3
7	1	2	3	3	4	4	1	1	2	2
8	1	2	4	4	3	3	2	2	1	1
9	1	3	1	2	3	4	1	2	3	4
10	1	3	2	1	4	3	2	1	4	3
11	1	3	3	4	1	2	3	4	1	2
12	1	3	4	3	2	1	4	3	2	1
13	1	4	1	2	4	3	3	4	2	1
14	1	4	2	1	3	4	4	3	1	2
15	1	4	3	4	2	1	1	2	4	3
16	1	4	4	3	1	2	2	1	3	4
17	2	1	1	4	1	4	2	3	2	3
18	2	1	2	3	2	3	1	4	1	4
19	2	1	3	2	3	2	4	1	4	1
20	2	1	4	1	4	1	3	2	3	2
21	2	2	1	4	2	3	4	1	3	2
22	2	2	1	4	2	3	4	1	3	2
23	2	2	3	2	4	1	2	3	1	4
24	2	2	4	1	3	2	1	4	2	3
25	2	3	1	3	3	1	2	4	3	2
26	2	3	2	4	4	2	1	3	3	1
27	2	3	3	1	1	3	4	2	2	4
28	2	3	4	2	2	4	3	1	1	3
29	2	4	1	3	4	2	4	2	1	3
30	2	4	2	4	3	1	3	1	2	4
31	2	4	3	1	2	4	2	4	3	1
32	2	4	4	2	1	3	1	3	4	2

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, pp. 4623-4625.

[2] K.H. Johansson, M. Törngren, M., L.Nielsen, Vehicle applications of controller area network. *Handbook of Networked and Embedded Control System,*. 2005, pp. 741–766,.

[3] Y.J. Choi's et al, "A study of HMI on in-vehicle Telematic System", in *Proc. 5th WSEAs International Conference on Applied Informatics and Communications*, 2005, pp.281–283.

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

[5] *Automotive SPICE Process Reference Model*. SPICE User Group, 2007, pp.1-47.

[6] *AUTomotive Open System Architecture* – AUTOSAR. Available: http://www.autosar.org/.

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

[8] T. Novosel, L. Jelenkovic, "Framework for embedded systems development", in *Proc. 34 th International Convention, MIPRO 2011*, 2011, pp. 825-828.

[9] P. Dostálek, J. Dolinay and V. Vašek, "Embedded System for audio source localization based on beamforming", in *International Journal of Circuits, Systems and Signal Processing*, Issue 6, vol. 6, 2012, pp. 367-375.

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

[11] C. Szyperski, Component Software Beyond Object-Oriented Programming, *Addison-Wesley Componet Software*, 2002.

[12] F. Bachman et al, *Volume II:* Technical Concepts of Component-Based Software Engineering. Technical report, *Carnegie Mellon Software Engineering Institute (SEI)*, 2000.
[13] I. Crnkovic, M. Larsson, Building Reliable Component-Based Software Systems. *Artech House Publishers*, 2002.
[14] M. S. Phadke, Quality Engineering Using Robust Design.: *Prentice-Hall, New York, NY*, 2008.
[15] J. Kontio, "A Case study in applying a Systematic Method for COTS Selection", in *Proc. 18th International Conference on Software Engineering (ICSE-18)*, 1996.
[16] A. Jadhav, "Analytic Hierarchy Process (AHP), Weighted Scoring Method (WSM), and Hybrid Knowledge Based System (HKBS) for Software Selection: A Comparative Study", in *Proc. 2nd Conference on Emerging Trends in Engineering and Technology (ICETET)*, 2009, pp. 991-997.
[17] BRASIL, CONTRAN, *Resolution No. 245, July 2007 – Deployment of National Automatic Vehicle Monitoring System (SIMRAV)*, 2007.
[18] 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", in SAE Technical Paper, 2010.
[19] K.C. Tai, Y. Lei, "A test generation strategy for pairwise testing". In *IEEE Transaction on Software Engineering*, vol. 28, 2002, pp.109-111.
[20] Taguchi Orthogonal Array Selector, Available: http://www.freequality.org/documents/tools/Tagarray_files/tamatrix.htm
[21] L. Lazica, N. Mastorakis, "Orthogonal Array application for optimal combination of software defect detection techniques choices" in WSEAS TRANSACTIONS on COMPUTERS, Issue 8, Volume 7, August 2008, pages 1319-1336.

Leopoldo Rideki Yoshioka born in São Paulo, Brazil in 1961. He received electronic engineer degree from Aeronautical Institute of Technology (ITA), Brazil, on 1984. Master and PhD degree from Tokyo Institute of Technology (Tokyo Tech), Japan, on 1988 and 1991.

He is currently a Professor of the Department of Electronic Systems Engineering at the University of São Paulo (USP), Brazil. His current research interests include embedded systems applied to the Intelligent Transportation Systems (ITS) and Autonomous Vehicles. He is a member of ITS Committee at the National Association of Public Transport (ANTP).

Marcio Camargo Oliveira born in São José dos Campos, Brazil, in 1975. He received the bachelor of Computer Science degree from Paraiba Valley University on 2002, with best student award.

He is currently, Master course student in Electronic and Computer Engineering on Aeronautical Institute of Technology (ITA), Brazil, and works as a Senior Systems Developer at Compsis Computadores e Sistemas Ltda, Brazil. His current research interests include embedded systems engineering, automotive technology and software engineering.

Claudio Luiz Marte born in São Paulo, Brazil, in 1963. In 1985 he completed his Degree at the Federal University of São Carlos [UFSC] and in 1988 completed Electrical Engineering (Electronic) at the Polytechnic School of the University of Sao Paulo [USP]. In 1994 he presented his Master of Science (MSc) and in 2000 he defended his Doctorate in Engineering (DE) thesis in Electrical Engineering (Digital Systems) at EPUSP.

He is currently a Professor of the Department of Transport Engineering (PTR) of EPUSP. His current research interests are: Moving Objects applied in ITS - Intelligent Transport Systems, Electronic Fee Collection (EFC), Advanced Public Transportation Services (APTS) and Advanced Traffic Management Services (ATMS). He is a member of ITS Brazil and ITS Committee of the National Association of Public Transport (ANTP).

Cledson Akio Sakurai born in São Paulo, Brazil, in 1972. He received the engineer degree from Faculdade de Engenharia Industrial on 1995, Master and PhD degree from Escola Politécnica of Universidade de SãoPaulo on 2004 and 2010.

He is currently, professor on Universidade Federal de São Paulo in Electrical Engineering. His current research interests include smart city, smart grid and telecommunications. He is a member of ASSESSPRO-SP (Software Association of São Paulo) and member on technical council of technological park in Santos.

Caio Fernando Fontana born in Botucatu, Brazil. He received the business administration degree from Faculdade de Administração de Empresas de Araçatuba on 1988, Master and PhD degree from Escola Politécnica of Universidade de SãoPaulo on 2004 and 2009.

He is currently, on Universidade Federal de São Paulo in Business Administration and Logistic. His current research interests include smart city, logistic and transport. He is a revisor of FAPESP (Funding Agency of São Paulo).

Edgar Toshiro Yano born in São Paulo, Brazil, in 1959. He received the engineer degree from ITA (Instituto Tecnológico de Aeronáutica) in 1981, Master degree from INPE (Instituto de Pesquisas Espaciais) in 1987 and PhD degree from ITA in 1998.

He is currently Associated Professor at ITA -Computer Science Division. His current research interests include Cybersecurity, Software Safety and C2 (Command and Control) Systems.