Interoperability issues on the design of safe in-vehicle embedded systems

Françoise Simonot-Lion

Interoperability issues on the design of safe in-vehicle embedded systems

Françoise Simonot-Lion, Nancy Université, Nancy (France)
LORIA UMR 7503 - Centre de Recherche INRIA Nancy-Grand Est
2 rue du Jardin Botanique - 54506 - Villers les Nancy - France
simonot@loria.fr

***Abstract*—The design of in-vehicle embedded systems follows a complex multi-partner development process. Carmakers specify the whole system and have to integrate several parts of the system provided by different suppliers. Specification as well as integration are concerned with properties requirements (safety, performance, cost, etc.) and validation issues. On another hand, the economical aspects lead suppliers to reuse previously developped components. At least, the portability of components is a necessary means that enable the flexibility of the development. For short, the problem when developping an automotive embedded system is the interoperability between components. To tackle this problem, two complementary solutions have been proposed by the automotive industry. The first one is the definition of a reference model for embedded systems that identifies component types and the formal rules of their interactions together. The other solution is a modeling language that can be shared by the different actors. In this paper, we show how automotive industry has contributed to these two aspects.**

I. Context

The part of embedded electronic systems in vehicles is nowadays growing due to economical and technological reasons. The main features of these systems are their distributed nature and the fact that they have to provide a level of quality of service fixed by the market, the cost requirements and the safety requirements. This last feature will be critical in the next decade because of the emerging standards that are likely to influence the certification process for in-vehicle embedded systems in terms of dependability guarantee. Furthermore, their development process is shared between different partners (mainly carmakers and suppliers). So, their design and their production have to be based on a suitable methodology including modeling, validation, optimization and test. One way to support a cooperative development process while ensuring the required properties of an embedded system is to share the same modeling language between the different actors. Other important problems are the portability of components from a microcontroller (termed Electronic Control Unit, ECU) to another one in a distributed architecture and the reuse of components from one to another platform. So the above mentioned language has firstly to take into account the final product, for short the operational architecture of an embedded system. Therefore the interoperability issue in the development of in-vehicle embedded systems is met through two main concepts:

- a reference model of embedded architecture
- and a language, based on this model, for the representation of an architecture, according to this reference model, at each step of the development.

In the following sections, these two complementary techniques will be presented.

II. Reference model of an embedded architecture - The AUTOSAR standard

As usually, a reference model is achieved by defining a layered software architecture focused on a middleware concept (see OSI model for protocol architecture definition). A similar approach, termed “Integrated Modular Avionics” was proposed in order to master the development of avionics software [22]. It provides a common API (Application Program Interface) to access the hardware and network resources, allows the Application developers to focus on the Application layer disregarding the deployment architecture. The proposal is based on the standardization of modules named LRMs (Line Replaceable Modules) housed in a cabinet and communicating through back-plane data buses. The aim is to connect several cabinets and existing dedicated avionics units on global multiplexed serial data buses to form an integrated system for performing all avionics functions. An important objective of this approach is the integration of maintenance support facilities.

Another study aiming a closed objective was driven in the TTA project that focused on time-triggered architecture (TTA). The proposed approach was designed for a wide range of fault-tolerant distributed real-time systems (automotive, aerospace, and railway industries). The proposal intended to ease the composability and a transparent implementation of fault-tolerance [29]. Within the Time-Triggered Architecture all activities (tasks, messages) in the system are controlled by the progression of a globally synchronised time-base. A main drawback of the proposal, in the present context where the CAN network is still predominant in automotive embedded systems, is that it imposes the use of a TDMA-based network (TTP/C or static policy of the FlexRay protocol). Therefore certain other studies have been done in order to define a less constrainted reference model for an embedded architecture that furthermore take into account the actual development process of software components that integrates a large amount of partners.

The first proposal comes from the European project EAST-EEA [3] that identified the main function of an in-vehicle

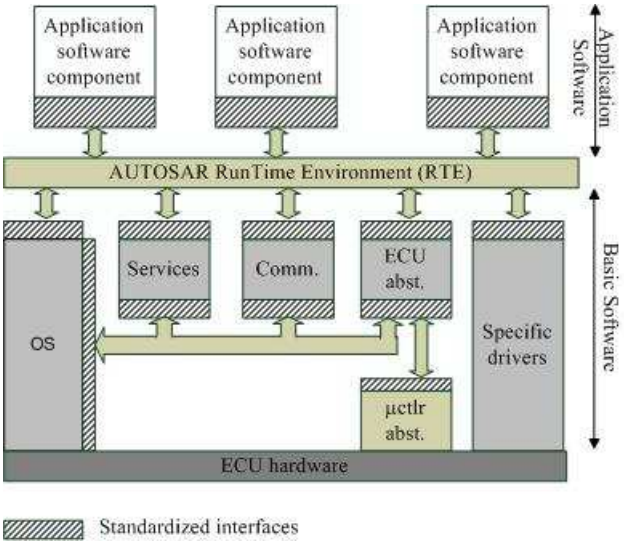

Figure 1. Autosar reference architecture

embedded middleware. The present business model in automotive industry is such that middleware itself is realized in each ECU by the integration of several components. So the only knowledge of an embedded middleware in terms of services and interface (API) provided to the application layer is not sufficient; the architecture of the middleware itself has to be carefully mastered in terms of components and interaction between these components.

This is the first objective of the international program AUTOSAR [13], [15] that gathers all the stakeholders of the automotive industry in the world. AUTOSAR specifies the software architecture embedded on an ECU. This architecture is composed of three main parts: the application layer, the basic software (middleware software components) and the Run Time Environment (RTE) that provides compatible software interfaces at application level. By using a well suited abstraction, it will be possible to separate software from hardware in a complex distributed architecture, allowing so the mastering of the complexity, the portability of application software components, the flexibility for product modification, upgrade and update, the scalability of solutions within and across product lines, the improvment of the quality and reliability of embedded systems. Such an architecture is schematically illustrated in the figure 1. An application software component is certified AUTOSAR compliant if its code calls only entry points defined by the RTE.

Furthermore, a basic software component, used at the middleware layer, has to be of one type specified by AUTOSAR; it is therefore certified AUTOSAR compliant if it provides the list of services and an interface defined formally in the specification of its type.

One of the main objectives of the AUTOSAR middleware is to hide the characteristic of the hardware platform as well as the distribution of application software components. So, the communication services inter or intra ECUs is of major importance and it is carrefully described (see figure 2). The role of these services is crucial for the behavioral and temporal properties of an embedded and distributed application. So, their design and configuration have to be precisely mastered and the verification of timing properties is an important activity. The problem is complex because the objects that are handled by services at one level are not the same that objects handled by services at another level and at each level; nevertheless each object is strongly dependant of one or several objects handled by services belonging to neighbouring levels. The AUTOSAR standard, as provided by the public version available in 2008, identifies three main objects regarding the communication: signal exchanged between software components at application level, I-PDU (Interaction Layer Protocol Data Unit) that consists of one or more signals and the L-PDU (Data Link Layer Protocol Data Unit) that will be effectively transmitted on the network.

More precisely AUTOSAR defines:

- *signals* at application level that are specified by a length and a type; a signal is exchanged between application software components through ports diregarding the distribution of this component; the application has just to precise a parameter that will impact the behaviour of its transmission: the value "*triggered*" of this parameter indicates that each time this signal is emitted by the application, it has to be transmitted on the network (as we will see later, this means that the sending of the frame containing this signal is directly done after the emission of the signal by the application component); on the contrary, the value "*pending*" for a signal indicates that its effective emission on the network depends only on the emission rule of the frame that contains it. Furthermore, when specifying a signal, the designer has to indicate if its a *data* or an *event*. In the former case, the emission of the signal by a component will copy it in a buffer erasing the last value of the same signal previously sent. The latter case specifies that signal are queued and therefore, it ensures that for each emission occurrence of this signal, a value will be sent. The handling of buffer or queue is done by the RTE.
- I-PDU are built by the AUTOSAR COM basic software component; each I-PDU gathers several signals (the length of an I-PDU has to be less or equal to 8 bytes); the service of AUTOSAR COM ensures the communication through a local transmission of the exchanged signal, when both components are on the same ECU or by building suited objects and requiring appropriate services of the lower layer, when the components are distant; this rule enables the portability of components and hide their distribution. In this latter case, a first problem is to go from signals to I-PDU and from I-PDU to signals [24]. This problem is ensured, thanks to an off-line configuration. Each I-PDU is characterized by a behavioural parameter, termed "transmission mode" whose value can be:
  - "direct" indicates that the sending of the I-PDU is done as soon as a "triggered" signal contained in this I-PDU is sent at application layer;
  - "periodic" means that the sending of the I-PDU is

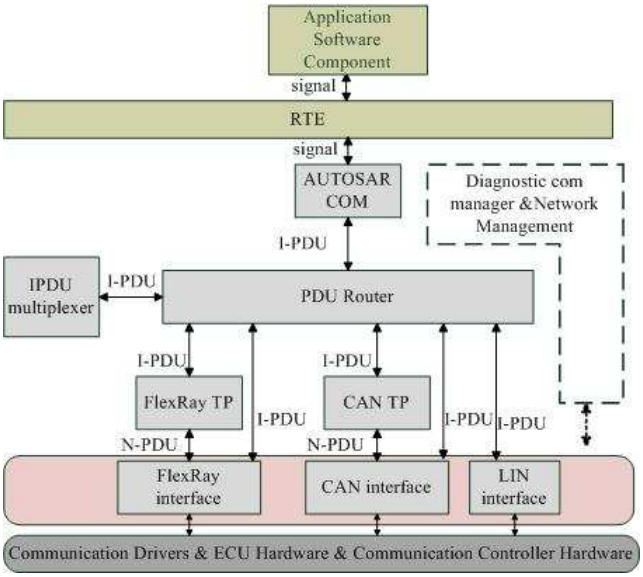

Figure 2. Communication software components and architecture

done only periodically; it imposes that the I-PDU does not contain "triggered" signals;
  - "mixte" specifies that the I-PDU is relevant of "direct" and "periodic" behaviour; this means that it takes into account the rules imposed by the "triggered" signals contained in the I-PDU and furthermore send periodically the I-PDU if it contains at least one "pending" signal
  - "none" characterizes I-PDU whose emission rules depend only of the underlying network protocol (for example, FlexRay or LIN)
- an N-PDU can be built by the basic components CAN TP or FlexRay TP (Transport Protocol); it represents the data payload of a frame that will be transmitted on the network; note that this step is not mandatory and a shortcut can directly transmit; the transport layer basic components are, obviously, network dependant; when they are used, these components build the data (N-PDU) by:
  - splitting the I-PDU for obtaining *n* N-PDUs that are, each, compliant to the frame length
  - or gathering several I-PDUs in one N-PDU.

AUTOSAR just provided a reference architecture at a static point of view. It describes precisely what are the elementary components embedded in an automotive middleware. The problem of the deployment of these components, their scaling, the validation of an embedded distributed system, etc. is not addressed by AUTOSAR. More precisely, how to specify the rules that each basic component has to follow is out of its scope. It has to be done along the development process of an application at functional, design and operationnal levels. If we focus on the communication services, the problem is rather complex. In fact it is relevant of an discrete optimisation problem under real time constraints. For example, at application level, only elementary *signals* are considered; these signals can be modelled by a length, a production rule (a period if the signal is periodically emitted, a minimal interarrival, if it is sporadically emitted, etc.), and, for real time signals, a freshness constraint imposed by their consumers. At network level, the objects that are exchanged are frames; they are characterized by their size and certain information that a designer has to specify, according to the used protocol; for example, for a CAN network [25], at least the priority of each frame (its identifier) and the emission rule must be fixed and, for a FlexRay network [25], the slot allocated to a frame, if this one is relevant of the static part of a FlexRay cycle, etc. At this level, certain characteristics of the network have to be provided such as the throughput, the specification of the FlexRay cycle, the length of its static / dynamic parts, the length of its slots, etc. As briefly described formerly in this section, the basic software components realizing the communication services of the AUTOSAR middleware are configured according to a lot of parameters regarding the communication:

- a signal can be queued or not at the RTE layer;
- a signal can be "triggered" or "pending";
- a I-PDU is composed of signals; the composition has to be specified;
- a I-PDU can be "direct", "periodic", "mixte", "none";
- a transport layer can be present or not; if yes, the splitting or gathering rules has to be specified.

Furthermore, the configuration of the certain networks has to be specified; for example, a FlexRay network is specified by the length of a cycle, of a macrocycle, of the static part, the length of the slots in the static part. For further information on several algorithms relevant to these problems, the reader can have a look on [26], [27], [28], [23] for the design of CAN oriented services and [16] for the design of TDMA-based oriented services.

The following section introduces a usually employed solution for mastering the development of a distributed application. As mentionned in the introduction, this solution is complementary and relies on a reference architecture model as the one provided by AUTOSAR. The underlying concept is based on the principles of Architecture Description Language that was introduced in order to support the development of a complex software architecture. In avionics domain, such a language is already put in practice; so the main stakeholders of automotive industry has defined such a language. The purpose of the next section is to introduce the concept of ADL, the interest of such an approach for embedded system design and to exhibit the language EAST-ADL developped for automotive.

III. Development of in-vehicle embedded systems

A. Roles and objectives of an Architecture Description Language

Several problems in the developpment of in-vehicle embedded systems are to be taken into account. First, the different partners do not play the same role in the development process. Carmakers provide the specification of all or parts of an embedded system; they focus on the specification of requirements (functional requirements and properties) and possibly on a first partitioning in functions. Suppliers realize the different parts of this specification; so, they handle on

the one hand, specification refinement and, on the other hand, hardware component and implementation characteristics. Finally, carmakers are responsible of the verification that each subsystem, furnished by different suppliers, is compliant with its initial specification. Then they have to verify its integration in the whole system. So a main purpose of the modeling language is on the one hand, to support the description of a system at different steps of its development (requirement specification, functional specification, design, implementation, tuning, etc.) thanks to different points of view and, on the other hand, to ensure a consistency between these different views. Another important aim of such a modeling language is its ability to reflect the structure of the embedded systems as architecture of components (hardware components, functional components, software components). So, the principles brought by Architecture Description Languages [21] are well suited to these objectives. The last point is concerned by the need of stringent validation and verification activities to be applied all along the development process. In fact, any error detected during the integration step leads to a costly feedback on the specification or design activities, and it must be avoided. So, in order to improve the quality of the development process, the different partners implicated in this process, apply more and more methods and techniques ensuring the correctness of subsystems as early as possible in the design stages and a new important trend is to consider the integration of subsystems at a virtual level [17]. This means that the semantic of the modeling language is mainly driven by the validation and verification purposes. In this context, the European project Eureka – ITEA EAST-EEA [3], defined EAST-ADL, an Architecture Description Language dedicated to the design of in-vehicle embedded systems.

As mentioned in the former section, the development of critical embedded systems requires specific frameworks that support functional and extra-functional specification capabilities. How to ease the design and implementation of predictable (real-time) embedded systems was the purpose of several former studies as, for example, the HOOD method [7], the BASEMENT approach [18] or the MARS project [20].

In order to master the software engineering process for large software architectures was developped the ADL concept (Architecture Description Languages). The ADL formalisms are used to describe the structure of a system by means of components interconnected by connectors in order to form configurations [21]. Another purpose of ADL is to ease the reusability of software components. So the component based software engineering paradigm (CBSE), that promotes reusability and safety, is a component centric approach that supplies the designer with an ADL, and provides methods and tools to verify the expected properties and derive a correct implementation (see for instance [5] for real-time systems). The description of a system according to an ADL that is purely software-oriented is done diregarding the implementation details, one of the objectives being the mastery of the more and more complex structure of modern systems. So the composition (associated to hierarchy) used to specify the assembly of the elements constitute the fundamental construction. In fact, an ADL is defined by several concepts: components, connectors and architectural styles that is a concept closed to design pattern and drive the composition activity.

The ADL used to specify real time systems support not only the specification of the functional aspects of the system, but also its extra-functional ones (timing properties, dependability properties, safety properties), and other transformation and verification facilities between design and implementation, while maintaining the consistency between the different models. In fact, for this kind of application, ADLs have to address both software and hardware architectures. In particular, certain points of view cannot be ignored:

- the behaviour of the system that has to react correctly to stimulus from the environnement and therefore an accurate model of the environment; this part is generally by ADLs suited to reactive systems and is modelled, among other, by a set of extra-functional requirements (timing, dependability, safety requirements, etc.)
- the hardware platform that imposes certain performances (execution time, transmission time) and the strategy used by the operating system and the network protocol to share processors and network; this aspect is addressed by ADLs that are able to manage tasks and to deal with schedulability analysis (for example, this point is directly taken into account in METAH and AADL);
- a model of the perturbation that can affect the system; this point is generally not covered at the present time.

Several ADLs target the real-time domain as MetaH, AADL (SAE) in the avionics domain or ADL_Transport, EAST-ADL (European Eureka-ITEA project), in the automotive one. MetaH [30] is a domain-specific ADL dedicated to avionics systems which has been developed at Honeywell Labs since 1991. In 2001, MetaH has been chosen as the basis for the definition of an Avionics Architecture Description Language (AADL) standard under the SAE authority [12]. AADL supports system and execution platforms modeling and provides a non ambiguous language to describe control, data flow, extra-functional aspects as timing requirements, fault model, safety requirements, time and space partitioning and certification properties (these latter aspects are specifically important in avionics). The language itself is completed by a set of tools oriented verification as schedulability analysis. The Cotre Project [2] targets the same objective for the development of real-time avionic systems by specifying in a formal way the links between the description of such a system expressed thanks to an Architecture Description Language and the formal techniques relying to verification activities. The used formalisms for verification activities are Transition Systems, Time Automata and Time Petri Nets (tools Uppaal, Tina or Aldebaran). In general, while ADL enable a hierarchical view of systems, V&V activities use a flat description of its architecture (for example, a configuration of tasks or a list of functional properties) and therefore 1- is based on just one level of abstraction and 2- is not able to propagate properties proved at one level to another one. Recent works [6] address this problem using properties associated to a component (a correct proved abstraction) in order to establish properties of the system by composition of the components of

the architecture.

In French automotive industry, recent efforts brought a solution for mastering the design, modeling and validation of embedded systems, and a first result was obtained by the French project AEE (Embedded Electronic Architecture [1]). AIL_Transport (Architecture Implementation Language for Transport) allows the specification in the same framework of embedded architectures, at several abstraction levels. The highest one captures the requirements and give a functional view. The lowest level models an implementation [11]. Two tools were developed in order to automate scaling and verification activities. They take, as input data, the system description in AIL_Transport. The first one is devoted to optimal distributed code generation realized by Syndex tool [19]. The second one is dedicated to performance properties verification by using Opnet simulator. A similar work is proposed in CAROSSE [8] through a language for implementation description (tasks exchanging messages over a communication architecture) and a timing property verification tool hiding the complexity of the required models. AIL_Transport has exhibited the main concepts that have to be present to support the description of in-vehicle embedded systems all along their development process. The following section presents EAST-ADL that is based on the same identified concepts.

B. EAST-ADL: an Architecture Description Language for automotive industry

EAST-EEA project [3], [9], [14] is an Eureka-ITEA European project in the automotive domain, with multidisciplinary approaches and various partners (car makers, automotive suppliers, research institute). It investigated from July 2001 to June 2004, through a more complete approach, automotive embedded architectures and development aspects. It aimed to unify the necessary concepts for automotive software development and moved towards a common notation, provided by the cornerstone of the project: the architecture description language, named EAST-ADL. EAST-ADL has been defined taking into account the development process and the architectural artifacts (see below). It allows capturing all information needed during development, from early analysis to implementation. It offers common modeling concepts and notations for the architectural artifacts, using a UML2.0 compliant approach. The language elements have been defined according to several language domains: structure (structural relations elements), behavior, requirements, V&V. This language was defined with intent to be, as well as possible, compliant to UML 2.0 . One of the purposes for the common notation is that suitable verification and validation (V&V) tools have to be supported by the underlying approach. Among the addressed, we focus, in the next section, on the Architecture Description Language, named EAST-ADL, that was specified. Then, in the following one, we demonstrate how validation and verification activities can be involved through this language. GME2000 tool, developed at the Institute for Software Integrated Systems, at Vanderbilt University, was used for the specification of the meta model describing EAST-ADL and as a support for the specific graphical editor developed for each view above

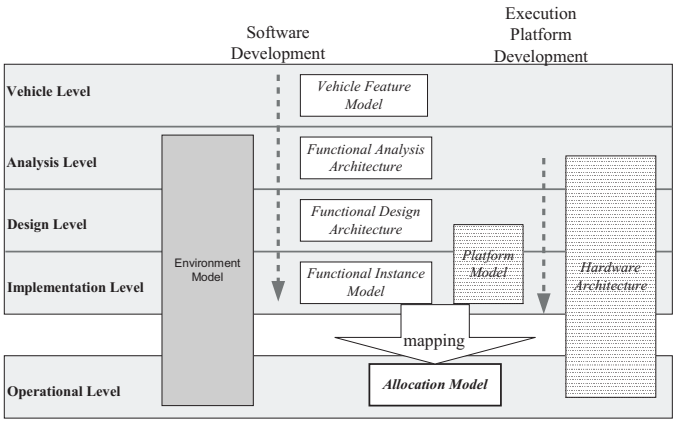

Figure 3. The abstraction layers of the EAST ADL

described. A checker verifying the consistency of a model, according to EAST-ADL semantic is attached to these editors. A complete model of a system is stored in a XML database.

The purpose of EAST-ADL is to provide a support for the non-ambiguous description of in-car embedded electronic systems at each level of their development. It provides a framework for the modeling of such systems through 5 abstraction levels populated by 7 views (also named artifacts) as shown in Figure 1. One can note that some particular views are distributed over several abstraction levels. It is only the desire not to clarify the refinement of certain view that is responsible that an artefact can be distributed on several abstraction levels: an example is the hardware architecture. Some of these views are mainly concerned with the software development while other ones are linked to the execution platform (ECUs, networks, Operating Systems, I/O drivers, middleware, etc.). Nevertheless, all these views are tightly coupled allowing traceability among the different entities that are considered during the development process. Besides the structural decomposition, which is typical for any software development or modeling approach, the EAST ADL has also support for modeling cross-cutting concerns such as requirements, behavioral description and validation and verification activities.

1) Artefacts in EAST-ADL : Each artefact in EAST-ADL, is the result of a specific activity among the development process. We describe below briefly these different artefacts. For further information on this language, the reader can refer to the public manual available at [3].

a) Vehicle Feature Model : It is the top-level view for a vehicle product line and it gathers all the low level artefacts (e.g. Functional Analysis Architecture, Hardware Architecture, etc.) for this product line. It supports the description of user visible features. Examples of such features are anti-lock braking or windscreen wipers. A Vehicle Feature Model is composed of a list of vehicle parameters that are relevant for the configuration of the embedded electronic systems (e.g. engine size, country, class, and possible customer choices such as cruise control, etc.) and a list of vehicle sets which describe set of vehicle instances sharing common user visible properties (e.g., all vehicles with cruise control and proposed in Europe).

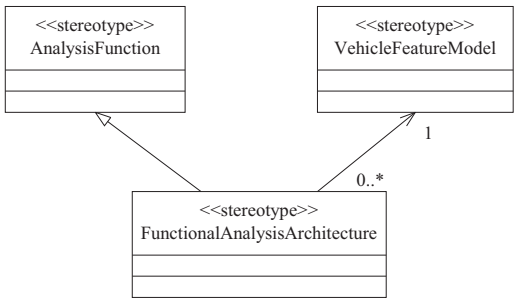

Figure 4. Functional Analysis Architecture

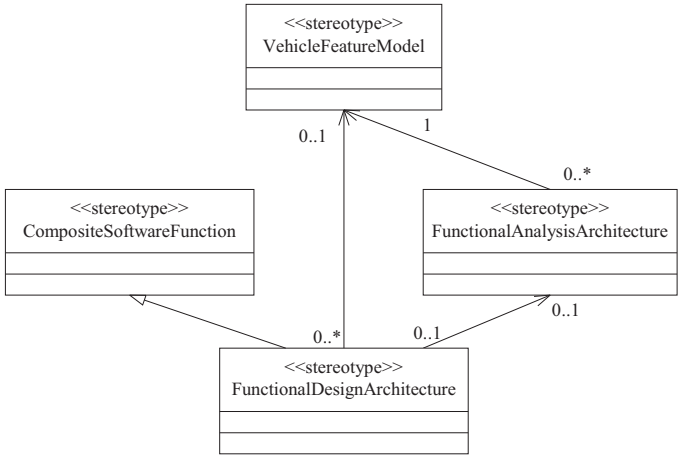

Figure 5. Functional Design Architecture

Thanks to these two kinds of parameters, the Vehicle Feature Model allows the description of the variation points among a product line.

b) Functional Analysis Architecture : This artefact represents the functions realizing the features, their behavior and their cooperation. The figure 2 shows the links between the concerned entities in EAST-ADL at this level. The FunctionalAnalysisArchitecture is a top level AnalysisFunction. It is supposed to implement all the functionalities of a in-vehicle embedded system, as specified at Vehicle level, by VehicleFeatureModel. There is an 1-to-n mapping between VehicleFeatureModel entities and FunctionalAnalysisArchitecture entities, i.e. one or several functions realize one feature. In order to be compliant with industrial practices, a VehicleFeatureModel can directly be refined at the design level; so, in this case, there is no link between a VehicleFeatureModel and a FunctionalArchitectureModel. A constraint is associated to the meta-model given in figure 2: a FunctionalAnalysisArchitecture may not be used as type of an AnalysisFunction (it is always a top level entity). An AnalysisFunction object can be refined and structured in several AnalysisFunction objects. Each AnalysisFunction object describe the functionalities at functional analysis level. They can interact with other AnalysisFunction objects through FunctionPorts. Furthermore, behaviour specification can complete the description of the object.

c) Functional Design Architecture : A FunctionalDesignArchitecture aims to model the refinement of one Func-

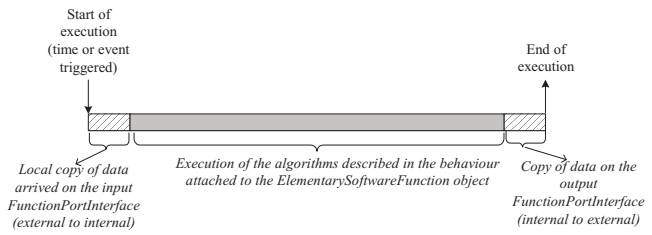

Figure 6. Execution model of ElementarySoftwareFunction object

tionalAnalysisArchitecture or of one VehicleFeatureModel. It is a container for the application software of an electronic embedded systems. A constraint is associated to the meta-model given in figure 3: a FunctionalDesignArchitecture object may not be used as type of a CompositeSoftwareFunction one (it is always a top level entity). A CompositeSoftwareFunction object can be refined in several CompositeSoftwareFunction ones. The last refinement produces objects, termed ElementarySoftwareFunction, that constitute an entry point for the Function Instance Model definition. Therefore, an ElementarySoftwareFunction cannot be decomposed. The interactions between CompositeSoftwareFunction and / or ElementarySoftwareFunction objects are done through ports, termed SignalFunctionPort (note that similar ways are provided for the interactions with system services and client/server communication services through SystemPort and OperationFunctionPort). Ports are parts of CompositeSoftwareFunction or of ElementarySoftwareFunction objects. The complete description of a SignalFunctionPort or of an OperationFunctionPort is achieved by specifying the interface requirement of the port (required / provided - period of the data transmitted through this interface, if necessary). So, SignalFunctionPortInterface objects are defined in Functional Design Architecture. These parameters specify only the input / output of data from a component point of view and so they don't precise the data flows into an architecture of components. Therefore, the ConnectorSignal objects aim to complete this specification by a description of the signal types that are exchanged through the port interfaces of CompositeSoftwareFunction or ElementarySoftwareFunction objects. This allows a unique description for several signals. For example, the vehicle speed is a data that is instantiated through several signals in an embedded architecture but each of them refers to the same ConnectorSignal object. A data type characterizes each ConnectorSignal object. Furthermore, precedence constraints which are imposed on the execution of ElementarySoftwareFunction objects can be specified through an association to an object, termed Precedes.

Finally, in EAST-ADL, a behavior is associated to each ElementarySoftwareFunction object: it is assumed to follow the execution model illustrated in figure 4.

d) Function Instance Model: The purpose of this artefact is to prepare the allocation of software components and exchanged signal to OS tasks and frames. It represents an intermediate step between Functional Design level where some decisions are independent of a particular allocation and implementation level where these decisions are to be completed. For example, at design level, the level of refinement of the

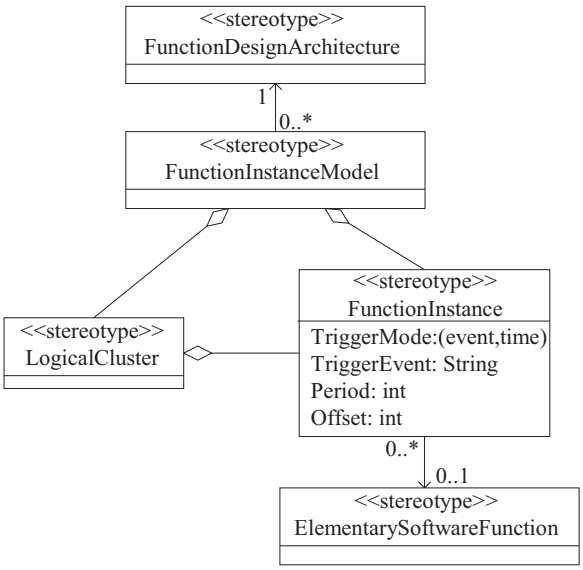

Figure 7. FunctionInstance specification

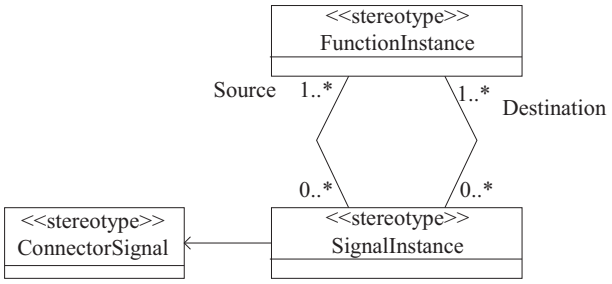

Figure 8. SignalInstance specification

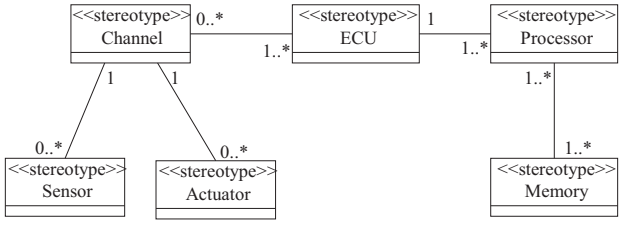

Figure 9. Hardware Architecture Level

objects allows only a global allocation of these objects to a specific ECU, but doesn't allow to specify their characteristic with respect to a local scheduling. A detailed description of software components has to be provided for this scheduling purpose. This is the role of the Function Instance Model. It is, in fact, a flat software structure where the Functional Design Architecture entities have been instantiated. This level provides an abstraction of the software components to implement on the Platform Model. The leaf functions of the Functional Design Architecture are an input to this artefact. The entities that appear in Function Instance Model are mainly FunctionInstance, SignalInstance and LogicalCluster objects (the specification is partially given in figures 5 and 6).

A FunctionInstance represents an instance of an ElementarySoftwareFunction while a LogicalClusterobject gathers all the FunctionInstance objects that have to be statically scheduled within the same OS task (Operating System) for performance reasons. Consequently, a LogicalCluster object has to be allocated on one and only one ECU. SignalInstance objects are used for communication within or between LogicalCluster objects. A SignalInstance object corresponds to a ConnectorSignal defined at the upper levels.

In order to model the implementation of a system, EAST-ADL provides, on the one hand, a way for the description of the hardware platforms through the Hardware Architecture artifact and their available services (operating systems, communication protocols, middleware) and, on the other hand, a support for the specification of how a Function Instance Model is distributed onto a platform. For this purpose, three additional views are necessary: the Hardware Architecture, the Platform Model and the Allocation Model.

e) Hardware Architecture: The Hardware Architecture artifact contains ECUs, channels, sensors and actuators. The ECU (Electronic Control Unit) models the computer nodes of the embedded system. ECUs consist of processor(s) and memory(ies) and may be connected via channels (serial links, networks) to sensors, actuators and other ECUs (see figure 7).

f) Platform Model: At the Platform Model level the models of the operating system and/or Middleware API and the services provided (schedulers, frame packing, memory management, I/O drivers, diagnosis software, download software etc.) are given. It is, in fact, the programmer's view of the Hardware Architecture.

g) Allocation Model : The Allocation Model represents the tasks, managed by the operating systems (OSTask objects), the frames, managed by the protocols (Frame objects) or the communication buffer used for internal communications (CommunicationBuffer objects). It is the result of the deployment of a Function Instance Model onto a Platform Model. Using a particular policy, specified through a FunctionDeployment object (resp. SignalDeployment object), a LogicalCluster object (resp. a SignalInstance object) is assigned to a particular OSTask object (resp. a CommunicationBuffer or Frame object). On this lowest abstraction level, all implementation details are captured. Figure 8 represents the deployment activity.

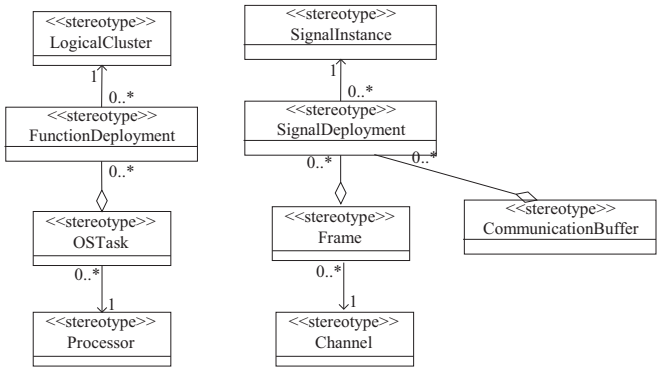

Figure 10. . Deployment model

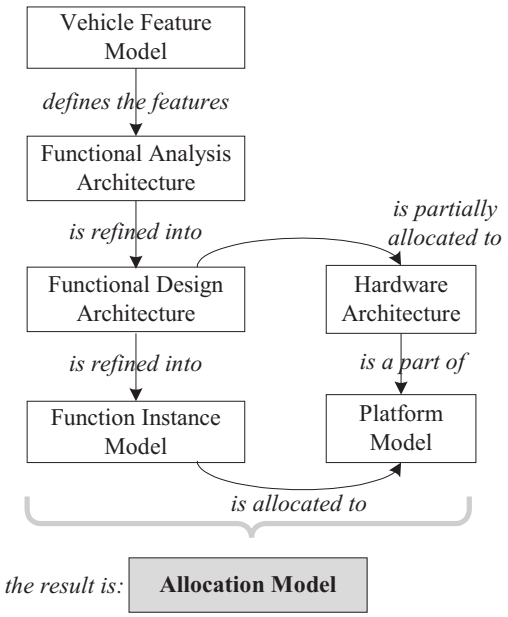

Figure 11. Links between artifacts as supported by EAST-ADL

2) Consistency of EAST-ADL models : A system described at the Functional Analysis level may be loosely coupled to hardware based on intuition, various known constraints or as a back annotation from more detailed analysis on lower levels. Furthermore, the structure of the Functional Design architecture and of the Function Instance Model is aware of the Platform Model. Nevertheless, in order to improve the quality of the development process, EAST-ADL provides a way to ensure the consistency within and between artefacts belonging to the different levels, at a syntactic and semantic point of view (see figure 9). This leads to make an EAST-ADL based model a strong and non-ambiguous support for building automatically models suited to formal validation and verification activities.

3) Requirements modelling : EAST-ADL is completed by several parts that supports the formal description of structured information aiming to help the designer along its various activities. For the purpose of this chapter, two parts are mainly concerned: the requirements specification and the validation and verification process. The "requirement" part provides a way for expressing the requirements that guides the building of a solution at each step of its development. So EAST-ADL defines the different requirements types that can be used and provides a support for linking them, on the one hand to components defined at one or several architectural point of views and, on the other hand to analysis, design and implementation models [31]. Finally the tracing activities between different requirements or between different versions of requirements can be expressed in this language. Five requirements types were identified. Each of them is defined: 1- by a textual description possibly completed by a formal one; 2- by an attribute concerned by the tracing activities; 3- by a status which is to be set to specific values according to the result of particular design, validation or verification activities. The requirements types are:

- EFeatures: an EFeature object (EFeature) describes the required functionalities of an embedded system; this kind of object is used, mainly, for specifying the system at Vehicle and Functional Analysis Architecture points of view. EFeatures may be decomposed into sub-features or "variant features". The last one is important for the description of the variation points that have to be taken into account for a given embedded system. - Interactions: these requirement type specifies the cooperation modes between EFeatures through textual description, semi formal one (as "use-cases") and possibly formal one (for example, "Message Sequence Charts").
- Functional Requirements: they aim to support the requirement imposed to the behaviour of an EFeatures by means of a set of required properties; once more, a formal description ("state-transition diagrams", "Messages Sequence Charts", etc.) can complete the textual one.
- Design Constraints: this requirement type aims to define how the research field for a solution is constrained; for example, such constraints can impose a protocol, a given preliminary allocation, a legacy tool for designing the system or a criteria to optimize (cost, power consumption, network bandwidth, etc.).
- Quality Requirements: they are used to express extra-functional properties; among these Quality Requirements, we can cite performance properties, reliability properties, safety properties, etc.

4) EAST-ADL as a support for Validation and Verification activities : In the first following subsection, we recall the major means for validation and verification that are applied in automotive industry; then we present how the EAST project has defined the validation and verification process in order to support a consistency between the underlying activity, the requirement specification and the entities described thanks to EAST-ADL language.

a) Validation and Verification process in automotive industry : From an industrial point of view, several techniques for assuming the correctness of an embedded system can be identified:

- Validation and verification at a functional level without taking into account the implementation characteristics. These activities, that can be applied to all or part of a system ensure the consistency of the system mainly with respect to the EFeatures, Interactions and Functional Requirements. At this level, simulation or formal analysis techniques can be, for example, used in order to prove some properties on the functional behaviour (lack of deadlock, dependability properties, etc.).
- Verification of properties of all or part of a system at Allocation Model level. For this kind of verification, we consider, on the one hand that each model was checked at the upper levels and, on the other hand that, at this level, the model is consistent with the upper ones. Under these assumption, the validation and verification process, thanks to simulation and formal analysis techniques, will be achieved by taking into account:
  - the characteristics of both the Hardware Architecture

(performances, reliability of hardware components) and the Platform Model (scheduling policies, protocols, frame packing, etc.),
  - and the load that is due to a given allocation of the Function Instance Model onto the Platform one.
- Moreover, these techniques that work on virtual platforms are completed by test techniques in order to assume that a realization is correct: test of software components, test of logical architectures, test of an implemented embedded system.

In the validation and verification context, formal techniques are concerned with analytical approach. This means that the model can be expressed according to mathematical formalism. The provided results, in this case, can be deterministic or probabilistic. These formal techniques can be opposed to simulation approaches. Note that the testing activities as well as the simulation ones consist in providing a scenario of events and/or data that stimulates the system under test or stimulates an executable model of the system. So in both techniques we have to look which events and/or data are produced by the system. The input scenario can be manually built or formally generated. In this last case the test or simulation activity is closely linked to a formal analysis technique. In the other cases, we obtain statistical results.

Certain tools or methods are of course of general interest in the context of validation and verification. We can cite, for example, Matlab / Simulink or Stateflow as well as Statemate for continuous or discrete model validation, timing analysis methods for the verification of the feasibility of a set of tasks and frames, discrete simulation techniques for performance evaluation purpose, etc. In some cases, an interface encapsulates these tools in order to adapt the tool to the automotive context. Nevertheless, before using these tools or methods, a specific model, well appropriate to them, has to be built.

In [9], [10], is presented a case study that illustrates how, thanks to EAST semantic, it is possible to generate automatically such a model; this case study starts with one requirement that is expressed at a high level ("*the system has to be robust to EMI perturbations*"). When refining the specification, this requirement is deployed on several requirements expressed on more concrete objects and, in particular, a timing requirement is applied to a specific signal object, termed Vehicle_Speed ("*the probability that the signal Vehicle_Speed doesn't respect a freshness constraint of 20 ms, has to be less than ε under a given fault model*"). We supposed in the example that the system is completly described and realize a specific parser that exploits the EAST-ADL compliant data base. The parsing process is done according to the following steps: 1- in which frame object F is packed this signal? 2- on which network is deployed the frame F? 3- let us suppose that the network is a CAN network, so, the next question is: which frames are deployed on this network? At this point, the parser has built the set of frames that share the same network. The last step is achieved by applying to this set a suited algorithm, as for example those that are described in [4] whose aim is to evaluate the probability that a frame doesn't respect its deadline by considering the worst case in each explored situation.

b) Validation and Verification process model in EAST-ADL : The EAST ADL language provides support for the description and the documentation of Validation and Verification (V&V) activities along the development process. Note that this description is largely inspired by the testing profile defined in UML2. More precisely, EAST-ADL defines several relevant entities for the V&V purpose and establishes a formal link between these entities and the artifacts describing the system at each level. These relations are illustrated by a simplified view of the proposed profile in figure 10. This view introduces the description of a validation and verification activity as an object (V&V_case). Two kinds of relation of such an object with other ones are particularly important:

- Relation with Requirement objects. The specified association establishes that one V&V activity (one V&V_case object) can contribute to the checking of several requirements, while one requirement can be checked thanks to several V&V activities.
- Relation with objects modeling the embedded system at each abstraction level. Obviously, as the purpose is to validate or verify properties of all or part of an embedded system, a V&V_Case object is associated to a set of objects related to an architecture layer. For example, a timing analysis applied to one Electronic Control Unit is concerned by the set of tasks (TaskOS objects) that are local to this ECU and by the scheduling policy used on this ECU and described by an OperatingSystem object. This set of ADL objects is named here System_under_VV.

Furthermore, as introduced in the previous section, a V&V activity is a generic class that can be refined in several subclasses, modeling formal analysis, timing analysis, scenario-based techniques (simulation or test), etc. Because the way to conduct a V&V activity depends on the techniques, several subclasses, and their specific attributes and associations have been identified (SimulationCase, TestSuite, FormalAnalysisCase, StaticAnalysisCase, TimingAnalysisCase, etc.). In the former section, we present briefly an example that focus on timing analysis. For this specific purpose, TimingAnalysisCase defines an object whose attributes specifies, in a textual form, the method to use (V&VMethod), the objective of this activity (Purpose), the additional input which are required for the method (Input) and the concerned property (PropertyDescription). Finally, any V&V technique applied at a virtual level is based on a model of the system expressed in a formalism that is relevant for this technique. A Model object (a configuration of frames in the mentioned example) refers to this model, and a Tool object precises the tool (the algorithm proposed in [4]) that has to be applied for this activity. Last, the final verdict of a V&V activity is given in a report (VV_Report object) and the final decision is done thanks to an Arbiter object.

IV. Conclusions

This paper aimed to highlight the two complementary approaches for supporting the interoperability issues in the multi-partners development process of in-vehicle embedded systems. The first approach, developped by the international

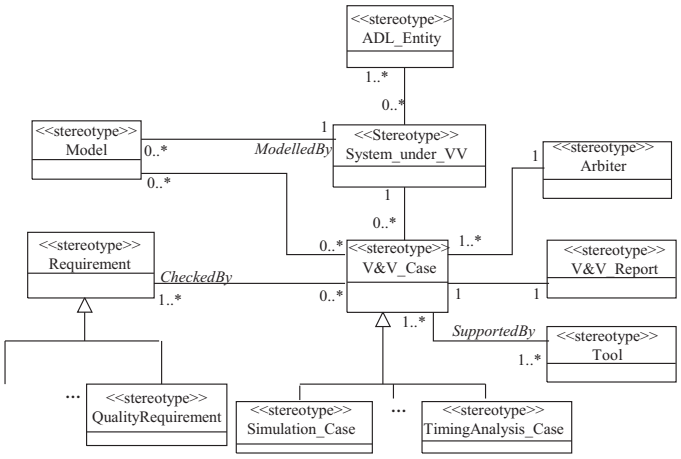

Figure 12. Validation and Verification process modeling

AUTOSAR consortium, furnishes a static reference model of an automotive embedded software architecture. It specifies a "virtual bus" and a middleware; for the latter one, it provides the specification of basic software components in terms of interfaces, services and inter-relation. The second approach is based on the Architecture Description Language concept. We presented EAST-ADL a language defined within a group gathering industrial actors of automotive industry and academic researchers (Eureka – ITEA EAST project, 2001-2004). We focused on the semantic that was brought to this language for making it a support for two coupled aspects, the requirement specification on the one hand, and the validation and verification activities, on the other hand.

These two standards (AUTOSAR model and EAST-ADL) are already applied in automotive industry. Not only carmakers and suppliers are concerned by this evolution but also every automotive industry subcontractors. Furthermore, a big market for software tools is nowaday emerging.

REFERENCES

[1] Architecture Electronique Embarquée. http://aee.inria.fr, 1999.

[2] Composant Temps Réel (Real-Time Component). http://www.laas.fr/COTRE.

[3] ITEA EAST-EEA project. http://www.east-eea.net.

[4] Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over CAN (Controller Area Network). *Journal of Systems Architecture*, 46(7), 2000.

[5] ARTIST. Component-based Design and Integration Platforms. Technical Report W1.A2.N1.Y1, ARTIST - Advanced Real-Time Systems - IST project, 2003.

[6] P. Binns and S. Vestal. Formalizing software architectures for embedded systems. In T.A. Henzinger and C.M. Kirsch, editor, *EMSOFT 2001, International Workshop on Embedded Software*, volume 2211. Lecture Notes in Computer Science, Springer, 2001.

[7] A. Burns and A.J. Wellings. Hrt-hood: a design method for hard real-time. *Real-Time Systems*, 6, 1994.

[8] P. Castelpietra, Y.-Q. Song, and F. Simonot-Lion. Analysis and simulation methods for evaluation of a networked embedded architecture. *IEEE Trans. on Industrial Electronics*, 49, 2002.

[9] Vincent Debruyne, Françoise Simonot Lion, and Yvon Trinquet. EAST-ADL - An Architecture Description Language - Validation and Verification Aspects. In P. Dissaux, M. Filali, P. Michel, and F. Vernadat, editors, *Architecture Description Language*, page 15 p. 2004.

[10] Vincent Debruyne, Françoise Simonot Lion, and Yvon Trinquet. EAST-ADL - An Architecture Description Language - Validation and Verification Aspects. In *IFIP Workshop on Architecture Description Languages 2004 - WADL'04*, pages 53–62, Toulouse/France, 2004.

[11] F. Simonot-Lion F. and J.P.Elloy. An Architecture Description Language for In-Vehicle Embedded System Development. In *15th Triennial World Congress of the International Federation of Automatic Control - B'02*, Barcelona, Spain, July 2002.

[12] P. H. Feiler, B. Lewis, and S. Vestal. The SAE Avionics Architecture Description Language (AADL) Standard. In *9th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2003*, Washington, USA, May 2003.

[13] H. Fennel, S. Bunzel, H.Heinecke, J. Bielefeld, S. Fürstand K.P. Schnelle, W. Grote, N. Maldenerand T. Weber, F. Wohlgemuth, J. Ruh, L. Lundh, T. Sandén, P. Heitkämper, R. Rimkus, J. Leflour, A. Gilberg, U. Virnich, S. Voget, K. Nishikawa, K. Kajio, K. Lange, T. Scharnhorst, and B. Kunkel. Achievements and exploitation of the AUTOSAR development partnership. In *Convergence 2006*, Detroit, USA, October 2006.

[14] U. Freund and A. Burst. Model-Based Design of ECU Software: A Component-Based Approach. In *OMER LNI, Lecture Notes of Informatics, GI Series*. October 2002.

[15] S. Fürst. AUTOSAR for Safety-Related Systems: Objectives, Approach and Status. In *2nd IEE Conference on Automotive Electronics*, London, UK, March 2006. IEE.

[16] Bruno Gaujal and Nicolas Navet. Maximizing the Robustness of TDMA Networks with Applications to TTP/C. *Real-Time Systems*, 31:5–31, 2005.

[17] P. Giusto, J.-Y. Brunel, A. Ferrari, E. Fourgeau, L. Lavagno, and A. Sangiovanni-Vincentelli. Automotive Virtual Integration Platforms: Why's, What's, and How's. In *International Conference of Comp. Desc.*, July 2002.

[18] H. Hansson, H. Lawson, O. Bridal, C. Eriksson, S. Larsson, H. Lonn, and M. Strömberg. Basement: an architecture and methodology for distributed automotive real-time systems. *IEEE Transactions on Computer*, 46, 1997.

[19] R. Kocik and Y. Sorel. A methodology to design and prototype optimized embedded robotic systems. In *2nd IMACS International Multiconference CESA'98*, Hammamet, Tunisia, 1998.

[20] H. Kopetz, G. Fohler, G. Grünsteidl, H. Kantz, G. Pospischil, P. Puschner, J. Reisinger, R. Schlatter-Beck, W. Schütz, A. Vrchoticky, and R. Zainlinger. Real-time system development: The programming model of mars. In *IEEE International Symposium on Autonomous Decentralized Systems*, 1993.

[21] N. Medvidovic and R. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1), 2000.

[22] M.J. Morgan. Integrated Modular Avionics for Next Generation Commercial Airplanes. *IEEE Aerospace and Electronic Magazine*, 6(8):9–12, August 1991.

[23] Nicolas Navet. Scheduling messages with offsets in automotive networks - a major performance boost. In *Seminar of the Department of Computer Science of Université Libre de Bruxelles (ULB)*, Bruxelles Belgique, 05 2007.

[24] Nicolas Navet and Françoise Simonot Lion. Fault Tolerant Services for Safe In-Car Embedded Systems. In Richard Zurawski, editor, *The Embedded Systems Handbook*. 2005.

[25] Nicolas Navet, Ye-Qiong Song, Françoise Simonot Lion, and Cédric Wilwert. Trends in Automotive Communication Systems. *Proceedings of the IEEE*, 93:1204–1223, 2005.

[26] Ricardo Santos Marques, Nicolas Navet, and Françoise Simonot Lion. Frame Packing under real-time constraints. In *5th IFAC International Conference on Fieldbus Systems and their Applications - FeT'2003*, pages 185–192, Aveiro Portugal, 07 2003.

[27] Ricardo Santos Marques, Nicolas Navet, and Françoise Simonot Lion. Network bandwidth optimization in automotive middlewares. Technical report, 2004.

[28] Ricardo Santos Marques, Nicolas Navet, and Françoise Simonot Lion. Configuration of in-vehicle embedded systems under real-time constraints. In *10th IEEE International Conference on Emerging Technologies and Factory Automation - ETFA'2005*, volume 1, pages 407–414, Catania, Italy, 09 2005.

[29] Christian Scheidler, Gunter Heiner, Ralph Sasse, Emmerich Fuchs, Hermann Kopetz, and Christopher Temple. Time-Triggered Architecture (TTA). In *Conference EMMSEC'97*, Florence, Italy, month=November, year=1997.

[30] S. Vestal. Metah support for real-time multi-processor avionics. In *Workshop on Parallel and Distributed Real-Time Systems, WPDRTS '97*, Geneva, Swiss, April 1997. IEEE Computer Society.

[31] Weber, M. and Weisbrod, J. Requirements Engineering in Automotive Development - Experiences and Challenges. *IEEE Software*, 20, 2003.