An Architecture Description Language For In-Vehicle Embedded System Development

Jean-Pierre Elloy, Françoise Simonot-Lion

AN ARCHITECTURE DESCRIPTION LANGUAGE FOR IN-VEHICLE EMBEDDED SYSTEM DEVELOPMENT

Jean-Pierre Elloy[(1)], Françoise Simonot-Lion[(2)]

[(1)] IRCCyN (UMR CNRS 6597) - ECN
B.P 92101 44321 Nantes CEDEX 03- France
[(2)] LORIA (UMR 7503) – INPL
2 avenue de la Forêt de Haye - 54516 Vandoeuvre-les-Nancy-CEDEX- France
Jean-Pierre.Elloy@irccyn.ec-nantes.fr, simonot@loria.fr

Abstract. This paper presents the AEE project, a French cooperative research and development program whose purpose is to specify new solutions for in-vehicle embedded system development. The Architecture Implementation Language (AIL) has been defined to specify and to describe precisely any vehicle electronic architecture. AIL supports the AEE design process, and is used by all designers as the backbone of the architecture development. Finally AIL is used to define reusable architecture objects.

Keywords: Distributed computer control system, Embedded systems, Architectures, Modelling, Concurrent engineering, Software project management, Automotive control

1. INTRODUCTION

1.1. Context

Currently, the part of embedded electronics in domestic cars is rapidly increasing (annual growth rate of 10%) due to the electronic injection systems, automatic cruise control, advances in comfort and safety. Moreover, using electronic functions in braking, active suspension, steering functionalities implies the respect of hard timing and fault-tolerant constraints. So the design of electronic systems starts defining eligible systems, i.e. satisfying these constraints, then producing the best one according to cost criteria. Consequently, design and development of such embedded systems induce new "multipartner" cooperation between carmaker(s) and OEM supplier(s).
In this context, AEE Project[1] is a French cooperative research and development program whose purpose is to specify new solutions for in-vehicle embedded system development (AEE, 1999).

[1] This work is granted by the French Under-Ministry for Industry. It involves French carmakers (PSA, RENAULT), OEM suppliers (SAGEM, SIEMENS, VALEO), EADS LV company and research centers (INRIA, IRCCyN, LORIA)

1.2. Main problems

From dedicated component to system architecture: Nowadays, most embedded sub-systems (hardware and software) are separately defined and developed. Each one is dedicated to a single functionality and is designed and tested as closed systems by an OEM supplier according to the carmakers specification requirements. Consequently, any design is specific to the developed application, and components are not easily reusable for subsequent projects. Moreover, as hardware and software resources are specifically defined for each functionality, embedded computers are costly over-sizing. Problems addressed by this state are the characterization of the basic embedded architecture components and the perimeter definition of the re-usable elements.
To reduce costs and optimize the use of hardware elements, AEE project defines a specific development method of embedded system. In the first step of this method, functionalities are defined and validated independently of their implementation. Then an allocation mechanism maps the specified functions onto the computers of the embedded architecture. Finally, local task execution and frame transmission are optimized. With this approach, capitalization is therefore no longer focalized on computers, but on the implemented functions through validated hardware and software modules.

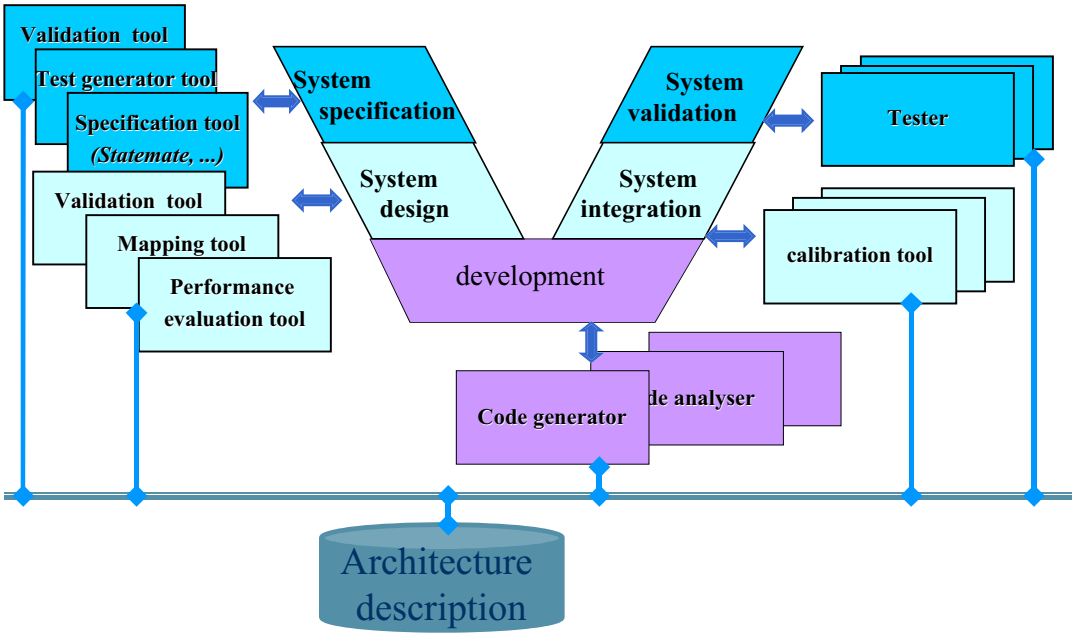

Fig. 1. Development process activities

A cooperative development process: Strong cooperation between OEM suppliers and carmakers in the design process implies the development of a specific concurrent engineering approach. In order to specify this process, synchronization rendezvous across the cooperative development model has to be identified and information exchanged at these points must be characterized. Furthermore, a unique syntax of the exchanged information has to be defined. For this, the AEE project has specified a business model designed to the architecture development together by carmaker(s) and OEM supplier(s).

Quality improvement: Performance of a vehicle embedded system may be evaluated from different points of view, according to the analyses of whole or parts of the system necessary at each development step. Usually carmakers attempt to optimize the number of ECUs used to implement the vehicle functionalities. Besides, system designers attempt to optimize performances of communication networks. Finally, OEM suppliers have to demonstrate their COTS are compliant with the carmaker requirements, etc. The AEE approach improves these different analyses and optimizations enabling the connection of various industrial and academic software tools to the architecture description. These tools are dedicated to analyze, test, simulate, validate, comment, and generate code of the described electronic architecture. For this, every tool extract a specific and coherent model from the architecture description by means a repository integrating all the pertinent data of the architecture modelling. This repository is the skeleton of the AEE development process, as shown in Fig 1.

In order to build this repository, a language to specify any vehicle electronic architecture has been defined. It is called Architecture Implementation Language (AIL). The AIL language integrates the AEE design process, and thus is used by all designers as the backbone of the architecture development. Moreover, AIL is the "source" language to define the reusable architecture objects.

Section 2 presents a brief state of the art of the architecture description languages. The AIL language is described in section 3 while section 4 shows how the underlying concepts are prototyped. Finally, section 5 concludes this presentation.

2. ARCHITECTURE DESCRIPTION LANGUAGES

In computer science, architecture description is a well-known technique that was largely explored by studies on development of Architecture Description Language (ADL) (Taylor and Medvidovic, 1997) or on modularity principles that led to object approach. But these techniques cannot be entirely adequate to the specification of vehicle electronic architectures because they do not include the specification of all devices integrated in such architectures (organs, sensors, actuators, computers, networks). Moreover these ADL generally don't merge temporal characteristics into the hardware and software component descriptions, and then do not allow real-time analysis of the architectures. Note: these possibilities are partially integrated in METAH, but only for applications without network communications (Vestal, 1993; Vestal, 1995). Besides, in the current solutions, some limitations appear in the analysis of ADL architecture models. For example,

simulation techniques were proposed in RAPID using the POSET formalism (Luckham, 1996), whereas model checking is used in SAM (He, *et al.*; 1999) or in WRIGHT (Allen and Garlan, 1997), but all these methods are based on simplified models of hardware component, insufficiently accurate to evaluate real-time performances of the architectures. Furthermore these techniques do not elaborate a specific description of architecture at each step of its development and thus are inadequate to support a continuous development process of architecture from the requirement specification to the code validation.

The object-oriented modelling is another technique to specify re-use constraints. This technique often is used in order to specify the software components, possibly to generate their code (Gamma, *et al.*, 1997). UML (Unified Modelling Language) has been defined by OMG as a general object-oriented modelling language, and a lot of industrial software tools supporting UML syntax are currently available. At this moment, extensions of UML are submitted to OMG in order to introduce the specification of temporal characteristics (Terrier and Gérard, 2000) and to improve the model analysis (Mellor, 2000; Miguel, *et al.*, 2000)).
Therefore, in order to take advantages of both ADL and object-oriented approach, AIL has been defined as a modelling language dedicated to the specification of architectures described as an assembly of standard components. Every component is an instance of class belonging to a generic model, and include all pertinent characteristics necessary to the subsequent analysis of the whole architecture: interaction consistency, logical behavior, real-time performances and fault tolerant properties. AIL is formalized using UML syntax.

3. ARCHITECTURE IMPLEMENTATION LANGUAGE (AIL)

3.1. Abstraction level concept

Using AIL, any designer describes the representation of an embedded architecture according to five different levels of abstraction. Each level models a particular point of view of the architecture. Entities specified at high levels ("vehicle project" level and "functional" level) are abstract components. Entities specified at low levels are, on the one hand, ECUs and communication networks in the "hardware" level, on the other hand, the software and the organs (sensors and actuators) in the "software" level and the "operational" level. At each level the designer describes the architecture as an assembly of objects instanced from predefined classes; then he specifies interactions between these objects using predefined connection types. The Fig .2 illustrates these levels and their relationships. The development process associated to this representation links components introduced at different levels.

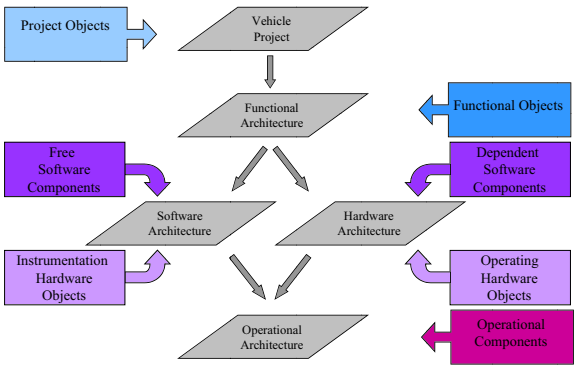

Fig. 2. Architectures and main classes

3.2. Vehicle level and functional level

Vehicle Project Level. The upper level describes an embedded application at vehicle point of view. At this level, objects shall represent services required by a vehicle (ABS, cruise control, air conditioning, etc), the different variant of these services (manual or automatic climate control system, for example) and the set of vehicle versions "on the shelf". Four classes are used to describe this architecture level: `Vehicle project`, `Vehicle type`, `Vehicle`, `Service`, `Variant`. Mainly, these classes document the architecture and support the project validation in terms of model coherence.

Functional Level. After building a validated vehicle model, the functional level describes one (or several) graph of elementary functional components realizing the services specified at the vehicle project level. Every graph of these components is specified disregarding the distribution and implementation aspects. This level is described using usual specification notations (data flow diagram, state machine diagram, object diagram), methods and tools. The model supports a hierarchical specification of functions and flows. `Function`, `Functional Flow` and `Functional Architecture` are the main classes used to build graphs of elementary functions. Documentation, formal validation and test generation are the main process activities fulfilled at this step.

3.3. Software level and hardware level

Hardware Level. This level models the electronic components of architecture as a set of processors, micro-controllers, electronic devices connected by networks. The main classes are

- `Operating Hardware Objects` whose main subclasses are `ECU` (Electronic Control Unit for the computation nodes) and `Network`,
- `Dependent Software Component`. These components are closely linked to hardware devices: `Network Protocol`, `COM BSC`, `OS BSC` and `Driver BSC` which are software components implementing communication services, operating system services and driver services,

- `Hardware Architecture` that specifies how each node is connected on one or several networks.

Software Level. This level models the software subset of the architecture. Two sets of classes are used. The first one is derived by class refinement of the functional architecture: the `Free Software Components` (running on "any" node), the `Software Flow`, the `Instrumentation Hardware Objects` (Sensor and Actuators) and the `Software Architecture`. The second set of classes models the distribution of all software entities. For this, `Software Component` are decomposed in `Logical Task` communicating using `Software Input` and `Software Output` which are linked to `Software Flow` of the functional level. Moreover, activation the policies of `Logical Tasks` are specified (timed or event triggered).

4. BUILDING AN AIL ARCHITECTURE

4.1. Major role of the software architecture

Objects modelled in the last architecture, i.e. the operational architecture, are data (local data or data encapsulated in network frames) and OS tasks. OS tasks model the code of the implemented functionalities. In the software architecture, this code is specified in some attributes of the `Software Component.` These attributes may describe:

- the logical behavior of the component during its execution, production of `Software Output` and consumption of `Software Input`,
- a temporal characterization of the `Software Component` and the timing constraints to be satisfied (periodicity, deadline, time interval),
- a dependability characterization describing information signaling a non tolerated error and policies to tolerate some internal errors.

These characterizations lead to the verification of properties of the whole vehicle architectures:

- the global behavior of a software architecture can be deduced formally from the elementary behaviors of `Software Component,`
- using the temporal characterizations, an allocation tool affects the `Free Software Component` to the `ECU` with the satisfaction of the timing constraints, so that processors and networks are never saturated,
- using the dependability characterization, specific tools evaluate the safety of whole architecture. To improve this safety; the designer may add new software components in the software architecture to build fault tolerance mechanisms. Using the same process, a designer also may introduce working modes of architecture adding components and data dedicated to particular coordinated management of the functionalities.

These characterizations qualify the services provided by a vehicle electronic architecture. That is why the software architecture has a major role in the building of architectures.

Before to present the gathering process used for an architecture building, the ex-nihilo building process is described in the following subsection. The steps of this process, ordering differently, are used in the actual gathering process. In any case, a final architecture is considered as achieved only when all objects of operational architecture are defined and when all object attributes are evaluated.

4.2. Ex-nihilo vehicle architecture building

Ex-nihilo building process doe not reuse components issued from a previously defined architecture. From top to bottom, this process browses the steps presented in Fig. 3.

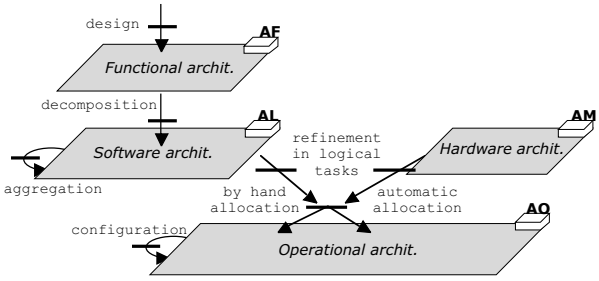

Fig. 3. Ex-nihilo building process

Without validation steps and recursive iterations, the main steps of this process are:

- the building of functional architecture,
- the decomposition of elementary functions, the definition of devices and the specification of data. This step leads to the software architecture,
- the parallel design of the hardware architecture defining `ECU`, `Networks`, serial links, and the `Dependent Software Components,`
- the mapping of the software architecture on the hardware architecture to produce the operational architecture. This mapping allocates the OS tasks on ECU in order to satisfy the timing constraints and to generate the configuration tables and the frames of the networks.

4.3. Principles for the building of an architecture by a "gathering" approach

The ex-nihilo building approach above does not take into account the industrial process of gathering preexisting architectural elements. One of the main objective of AIL language is to permit the collection and re-exploitation of architectural elements. Thus a more industrial architecture building would be a gathering of architectural elements realized by other industrial partners or already at-hand, and possibly their modification or the addition of new elements to them. Therefore, the general construction phases are:

- definition of the services to be expected from the architecture,
- consultation with the industrial leading to acquisition of some functionalities or complete

devices for the realization of a part or all of some of the services,
- collection in a data base of already produced architectural elements, the "reused architectural elements", and the "sub-contracted architectural elements",
- specification of new elements which must be added to its architecture in order to complete the entire set of expected services,
- choice of the hardware components and of the communication systems able to support the sub-contracted, reused and new architectural elements. This step may be performed simultaneously with the previous steps. Some of these hardware components may be fixed by some industrial partners or some previous realizations,
- construction of the architecture by assembly, as presented in the section 4.4.

At the second step of this process, the sub-contracted architectural elements can be delivered:
- as a "black box": specifications delivered by the supplier are limited to the service description and the characterization of the inputs and outputs of its element. This characterization defines the communication format and the rate of production (or consumption) of data at the API layer,
- as a "gray box": the internal functionalities of the elements are detailed, their internal data exchanges and external data exchanges are specified and their parameters can be modified. In this case, the only blinded information is the internally implemented code,
- as any form between the previous mentioned ones.

In order to integrate all these cases, the industrial building process begins with a first step specific to each type of architectural element to include in the architecture (sub-contracted, reused or new). The designer can apply any order to integrate these elements: this integration is a commutative operation. After the preliminary steps, software architecture and hardware architecture are defined, and the final steps of the vehicle architecture building are those of the ex-nihilo process.

4.4. Preliminary steps for gathering building process

To illustrate the process described above, the following subsections present the principle of two preliminary steps: how add a new architectural element to an existing architecture, and how add a sub-contracted "black-box" architectural element to an existing architecture.

Adding a new architectural element

The steps of the ex-nihilo process are applied to the building of a new architectural element:
- design of the functional architecture objects: elementary functions and vehicle data;
- design of software architecture by decomposition of these elements.

These steps are presented on Fig. 4 in the case the element to integrate in the architecture is a new functional element without any associated hardware component:

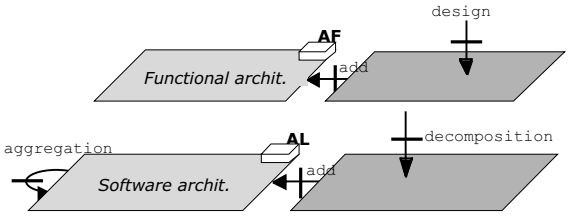

Fig. 4. Construction of a " new architectural element"

Adding a sub-contracted " black box" architectural element

To integrate a "sub-contracted black box architectural element" in its architecture, the designer must "abstract" this element, i.e. models the element as a set of AIL objects. This modelling is an "external" representation of the element (in terms of functionalities, logical behavior, temporal response time). Then, the integration becomes a concatenation of the element representation with the set of architectural elements which are soon created at the software level and at the hardware level:
- adjunction of `ECU`, `Networks` and modelled `Dependent Software Components` of the sub-contracted element to the hardware architecture,
- adjunction of the modelled `Free Software Components` of the sub-contracted element to the software architecture,
- adjunction to operational architecture of the distribution of software entities of the sub-contracted element onto hardware entities of the sub-contracted element,
- if needed, abstraction of a modelled functional architecture of the sub-contracted element from its modelled software architecture.

This is illustrated on Fig. 5 in case of the sub-contracted element is an ECU:

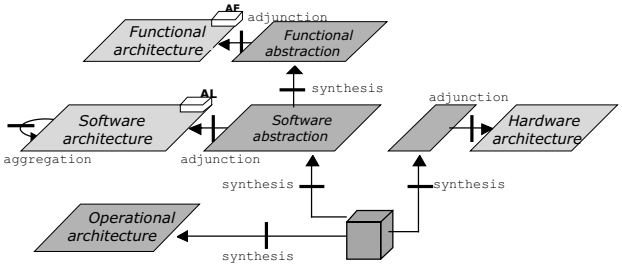

Fig. 5. How add a " sub-contracted element"

4.5. Final step

After applying the preliminary steps, the final steps of vehicle architecture building are the same as in the ex-nihilo process:

- refinement of the `Free Software Components` and of the `Dependent Software Components` into OS tasks;
- allocation of the OS tasks onto ECUs. This phase allocates the entities issued from the sub-contracted and from the reused architectural elements on their native hardware components,
- automatic allocation of the other OS tasks,
- configuration of the communication tables and of the frames for the inter-ECU communications.

For sub-contracted or reused elements, the decomposition into OS tasks is already done and this operation is replaced by the direct initialization of attributes of the sub-contracted or reused elements. This initialization is accomplished at the time of their integration (preliminary steps).

5. CONCLUSIONS

This paper presents an architecture description language for specifying and describing precisely the in-vehicle electronic embedded systems to carry out automotive functionalities. The proposal provides a tool to build flexible architectures making easier the introduction of new electronic systems in the cars of the future. Immediate benefit of this proposal is a significant cost reduction and increased pace in the development of architectures.
Associated to the AIL language, a development process has been defined for defining and harmonizing the exchanges of partial architectures between carmakers and OEM suppliers.
In fine, these results lead to software architectures that enable both software/ software and hardware/ software independence

The authors would like to thank Jorn Migge (Loria), Franck Gasnier (IRCCyN), Xavier Hanin (PSA) Evelyne Silva (PSA), Bernard Bavoux (Valeo) and Ziad El Khoury (Valeo) for their essential contributions to the definition of the AIL language.

REFERENCES

AEE (1999), Architecture Electronique Embarquée, http://aee.inria.fr

Taylor R. N. and N. Medvidovic, (1997). A Framework for Classifying and Comparing Architecture Description Languages, *Technical Report*, Department of Information and Computer Science, University of California, Irvine.

Vestal S., (1993). Scheduling and Communicating in MetaH, *in Proceedings of Real -Time System Symposium*, Rleigh-Durham (NC), p.194-200.

Vestal S., (1995). MetaH Reference Manual, *Technical Report*, Honeywell Technology Center.

Luckham D. C., (1996). Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events, *in Proceedings DIMACS Partial Order Methods Workshop IV*, Princeton University.

He X., F. Zeng and Y. Deng, (1999). Specifying Software Connectors in SAM", *in Proceedings of SEKE 1999*

Allen R. and D. Garlan, (1997). A Formal Approach for Architectural Connection, *PhD thesis*, school of Computer Science, Carnegie-Mellon University, Pittsburgh.

Gamma E., R. Helm, R. Johnson and J. Vlissides, (1995). Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley.

Terrier F. and S. Gérard, (2000). Real Time System modelling with UML: currently status and some properties, *in Proceedings of 2nd Workshop on SDL and MSC*, Col del Porte Grenoble.

Mellor S.J., (2000). Advanced Methods and Tools for precise UML: Visions for the future, *in Proceedings of OOPSLA, workshop UML*, Denver.

De Miguel M., T. Lambolais, S. Piekarec, S. Betgé-Brezetz and J. Péquery, (2000). Automatic Generation of Simulation Models for the Evaluation of Performance and Reliability of Architectures Specified in UML, *in Proceedings of 2nd Int. Workshop on Engineering Distributed Objects*, University of California, Davis.

Castelpietra, P., Simonot-Lion F., Song Y.-Q. and Attia M., (2000). Performance Evaluation of a Multiple Networked in-Vehicle Embedded Architecture, *in Proceedings WFCS'2000*, Porto.

Castelpietra, P., Simonot-Lion F., Song Y.-Q. and Attia M., (2001). CAROSSE-Perf : a modular approach fo simulation of in-vehicle embedded architecture, *in Proceedings ESM2001,* Prague.

Migge J. and Elloy J.P., (2000). Embedded electronic architecture, *in 3rd Int. Workshop on Open Systems in Automotive Networks*, Bad Homburg, Germany, Feb. 02-03, 2000.