A Partitioning-Centric Approach for the Modeling and the Methodical Design of Automotive Embedded Systems Architectures

Dissertation

A thesis submitted to the

Faculty of Electrical Engineering and Computer Sciences
of the
Technical University of Berlin
in partial fulfillment of the requirements for the academical degree of
Dr.-Ing.

by

Augustin Kebemou

Berlin 2008
D 83

A Partitioning-Centric Approach for the Modeling and the Methodical Design of Automotive Embedded Systems Architectures

vorgelegt von
Dipl.-Ing. Augustin Kebemou
Berlin

Von der Fakultät IV - Elektrotechnik und Informatik -
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

- Prof. Dr.Sabine Glesner (Vorsitzender)
- Prof. Dr.-Ing. Ina Schieferdecker (Gutachter)
- Prof. Dr. Jakob Rehof (Gutachter)

Tag der wissenschaftlichen Aussprache: 06.05.2008

Berlin 2008
D 83

Acknowledgments

"Acquiring knowledge is useless unless it makes us better servants of humanity".

The achievement of this thesis would not have been possible without the intervention and the support of many people. I am very grateful to all those who supported me during the work that led to this thesis.

In particular, I would like to thank the supervisor of this work, Prof. Dr.-Ing. Ina Schieferdecker (holder of the chair for Design and Testing of Telecommunication Systems at the TU Berlin and concurrently leader of the research group MOTION at the Fraunhofer Institute FOKUS) for her advices, for her reviews and her suggestions that improved both the technical and the scientific quality of this thesis. I also thank Ina for the friendly working atmosphere that gave me additional self-confidence to go ahead.

I am also deeply grateful to Prof. Dr. Jakob Rehof, the director of the Fraunhofer Institute for Software and Systems Engineering (ISST) and concurrently holder of the chair for Software Engineering at the university of Dortmund, who accepted spontaneously to support me, invested his time to examine my work and undertook the co-supervision activities that considerably contributed to the achievement of this thesis.

Taking this opportunity, I would further like to thank Rainer Mackenthun, the head of the Department of Dependable Technical Systems at the Fraunhofer Institute for Software and Systems Engineering (ISST) in Berlin, Prof. Dr. Herbert Weber, Dr. Alexander Borusan and Dr. Volker Zurwehn, also from the ISST institute in Berlin, for their psychological support. I did appreciate your fairness and the way you judged my work during my hard days.

I would also like to address my special thanks to Angelika Becker, who played a central role in the accomplishment of this thesis. Thank you very much, Geli, for the nights and the weekends you spent to review this thesis and to correct my textual formulations. Thank you for your advices. Thank you for the time you invested in intensive discussions with me. Thank you once more for your emotional support.

I finally wish to express my gratefulness to my family and all of my friends for their patience during the years of this thesis as well as to my colleagues at the Fraunhofer Institute for Software and Systems Engineering (ISST) who participated in the fruitful discussions that made this thesis possible.

Thank u all.

Once more, remember: *"Acquiring knowledge is useless unless it makes you a better servant of humanity"*.

A. Kebemou

Abstract

Because of the increasing demand for more comfort, security and environmental compatibility, the development of E/E-systems (Electric/Electronic) has become a central task for automobile manufacturers. In the actual context that is characterized by the rapid increase of software- and electronic-based components in modern vehicles and the related hard competition in the automobile market, it is necessary to design optimal architectures for automobiles' E/E-systems in a relatively short time. An optimal architecture of an E/E-system must minimize the usage of the hardware (i.e. processing units, memory elements, communication cables, etc.) as well as the operating costs (energy and fuel consumption, maintenance, waste disposal, etc.) and concurrently optimize the functioning and the quality (i.e. performance, reliability, safety, security, etc.) of the system. In this thesis we suggest to solve this problem by means of CAD-supported tools. This requires drastic changes within both the established development methods and the design processes. We propose a system-oriented design process and an automatic partitioning method with appropriate modeling techniques to support the model-based definition of the architectures of automobiles' E/E-systems.

Abstrakt (Deutsch)

Vor dem Hintergrund der steigenden Nachfrage nach mehr Komfort, Sicherheit und Umweltfreundlichkeit ist die Entwicklung von E/E-Systemen (elektrik/elektronik) zu einer zentralen Aufgabe für die Automobilhersteller geworden. In der gegenwärtigen Situation, die durch die rapide Zunahme von software- und elektronikbasierten Bestandteilen in modernen Fahrzeugen und den damit verbundenen harten Konkurrenzkampf auf dem Automobilmarkt gekennzeichnet ist, ist es notwendig, optimale Architekturen für automobile E/E-Systeme in relativ kurzer Zeit zu entwickeln. Die optimale Architektur eines E/E-Systems muss die Hardwarenutzung (d. h. Prozessoren, Datenspeicher, Kommunikationskabel, I/O Module, etc.) und die Betriebskosten (Energie- und Treibstoffverbrauch, Wartung, Verschrottung, etc.) verringern und gleichzeitig die Funktionstüchtigkeit und die Qualität (Leistung, Zuverlässigkeit, Sicherheit, etc.) des Systems verbessern. In dieser Arbeit schlagen wir vor, dieses Problem durch computergestützte Werkzeuge zu lösen. Dies macht einschneidende Veränderungen sowohl in den Entwicklungsmethoden als auch in den Designprozessen erforderlich. Wir schlagen einen systemorientierten Entwicklungsprozess vor, der die modellbasierte Definition der Architekturen von automobilen E/E-Systemen unterstützen kann. Diese Arbeit definiert die passenden Modellierungstechniken und die Algorithmen, die eine automatische Partitionierung von System-level-E/E-Modellen ermöglichen.

Contents

1 General introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem definition . . . 3
- 1.3 Procedural method to solve the problem . . . 9
- 1.4 Contributions . . . 12
- 1.5 Publications . . . 13
- 1.6 Scheduling of the thesis . . . 13

2 Automotive Embedded Systems **15**
- 2.1 Automotive electronics . . . 15
  - 2.1.1 Embedded systems . . . 15
  - 2.1.2 Automotive and embedded systems . . . 16
  - 2.1.3 Example: The Active Cruise Control . . . 17
  - 2.1.4 The ACC functional connectivity . . . 19
- 2.2 Automotive connectivity . . . 19
  - 2.2.1 Automotive communication protocols . . . 19
  - 2.2.2 All-rounder automotive communication protocols . . . 20
  - 2.2.3 High-speed and real-time protocols . . . 21
  - 2.2.4 General-purpose protocols . . . 23
  - 2.2.5 Other in-vehicle and smart network protocols . . . 24
- 2.3 Conclusion . . . 25

3 Embedded systems design **27**
- 3.1 Embedded systems design methods . . . 27
  - 3.1.1 The sequential design process . . . 27
  - 3.1.2 The concurrent design method . . . 28
- 3.2 Design activities . . . 29
  - 3.2.1 The specification . . . 29
  - 3.2.2 The partitioning . . . 29
- 3.3 The implementation . . . 30
  - 3.3.1 The software synthesis . . . 30
  - 3.3.2 The hardware synthesis . . . 31
  - 3.3.3 The synthesis of the interfaces . . . 32
- 3.4 Conclusion . . . 32

4 Automotive systems design **35**
- 4.1 Design of AES . . . 35
  - 4.1.1 Top-down and bottom-up . . . 35
  - 4.1.2 Current OEM design practice . . . 35
  - 4.1.3 Limitations of the current OEM design practice . . . 36
- 4.2 Proposed design approach . . . 37
  - 4.2.1 Factors of the problem resolution . . . 37

4.2.2 The system-oriented design approach . . . 38
4.2.3 Analysis . . . 39
4.3 Conclusion . . . 40

5 Modeling AES: State-of-the-art 41
5.1 Modeling AES . . . 41
5.1.1 Model-driven system development in the automotive engineering . . . 41
5.1.2 AES are heterogeneous and complex systems . . . 42
5.2 AES modeling needs . . . 42
5.2.1 AES modeling prerequisites . . . 42
5.2.2 Features expected from an AES model-based design solution . . . 43
5.3 AES basic modeling concepts . . . 44
5.3.1 Abstraction levels in the AES design . . . 44
5.3.2 AES architectural modeling concepts . . . 45
5.3.3 AES behavioral modeling concepts . . . 47
5.4 Conclusion . . . 48

6 The value of AES modeling languages 49
6.1 The evaluation framework . . . 49
6.1.1 AES modeling requirements for the partitioning . . . 49
6.1.2 Related work . . . 50
6.1.3 Classification criteria . . . 51
6.1.4 Criteria for evaluating the level of support . . . 53
6.2 AES modeling languages . . . 55
6.2.1 General-purpose languages . . . 55
6.2.2 Automotive domain-specific modeling languages . . . 58
6.3 Evaluation and classification of AES modeling languages . . . 61
6.4 Conclusion . . . 62

7 Inputs for the partitioning 67
7.1 Required inputs for the partitioning . . . 67
7.2 Specifying the system's functionalities . . . 68
7.2.1 Relevant modeling concepts . . . 68
7.2.2 The FN: The modeling solution for the functional specification . . . 69
7.3 Specifying the hardware platform . . . 72
7.3.1 Relevant modeling concepts . . . 72
7.3.2 The HN: The hardware platform . . . 75
7.4 The partitioning . . . 75
7.4.1 Formal definition of the partitioning problem . . . 75
7.4.2 Relevant attributes for the elements of the input models . . . 77
7.5 Conclusion . . . 79

8 The synthesis Model 81
8.1 Definition of the synthesis model . . . 81
8.1.1 Requirements for the synthesis model . . . 81
8.1.2 The synthesis model . . . 82
8.2 Annotations for the synthesis model . . . 85
8.2.1 Concurrency, sequencing . . . 85
8.2.2 Annotations for the nodes . . . 85
8.2.3 Annotations for the edges . . . 86
8.2.4 Annotations for the tokens . . . 86
8.2.5 Formal definition of the synthesis model . . . 87
8.3 Applications . . . 88

8.3.1 The weight of an edge in a $CDFM$ model . . . 88
8.3.2 Model transformation . . . 89
8.4 Conclusion . . . 89

9 The partitioning: State-of-the-art 91
9.1 The partitioning problem . . . 91
9.1.1 Frames of the problem . . . 91
9.1.2 Requirements for the partitioning algorithm . . . 92
9.2 Partitioning methods . . . 93
9.2.1 Exact and heuristic methods . . . 93
9.2.2 Constructive partitioning techniques . . . 95
9.2.3 Iterative improvement techniques . . . 97
9.3 Conclusion . . . 101

10 The partitioning algorithms 103
10.1 The partitioning strategy . . . 103
10.1.1 A three-step process . . . 103
10.1.2 Definitions of terms . . . 104
10.1.3 The main procedure . . . 105
10.2 The pre-clustering . . . 105
10.2.1 Definition . . . 105
10.2.2 The pre-clustering algorithm . . . 106
10.3 The clustering . . . 107
10.3.1 Closeness metrics . . . 107
10.3.2 The closeness function . . . 110
10.3.3 The QT clustering algorithm . . . 111
10.3.4 Conclusion . . . 113

11 Evaluating and improving a partition 115
11.1 The CAN: A frame-oriented communication protocol . . . 115
11.1.1 Organization of a CAN network . . . 115
11.1.2 CAN frames . . . 116
11.1.3 Format of a standard CAN data frame . . . 116
11.1.4 Frames multiplexing . . . 117
11.1.5 Relations with the partitioning . . . 119
11.1.6 Practical considerations for the frames multiplexing . . . 119
11.2 The value of a partition . . . 121
11.2.1 The cost function . . . 121
11.2.2 The cost as a bin packing problem . . . 122
11.3 Bin packing techniques . . . 123
11.3.1 The Next Fit, the First Fit and the Best Fit strategies . . . 123
11.3.2 Off-line packing strategies . . . 124
11.4 Investigating the cost of a partition . . . 125
11.4.1 The FFD strategy for the cost estimation . . . 125
11.4.2 The frames packing algorithm for the cost investigation . . . 126
11.5 Improving the partition . . . 127
11.5.1 The Kernighan & Lin strategy . . . 127
11.5.2 The improvement technique . . . 128
11.5.3 The improvement procedure . . . 130
11.6 Conclusion . . . 132

12 Applications **133**
12.1 The application case . . . 133
12.1.1 Presentation of the application case . . . 133
12.1.2 Objectives and scenario . . . 134
12.2 The investigation . . . 135
12.2.1 The models . . . 135
12.2.2 The attributes of the tokens . . . 137
12.2.3 The partitioning . . . 139
12.3 Results . . . 139
12.4 Conclusion . . . 140

13 General conclusion **141**
13.1 Summary . . . 141
13.2 Outlook . . . 145

A Zusammenfassung der Dissertation **1**
A.1 Motivation . . . 1
A.2 Problemlösung . . . 3
A.3 Wissenschaftliche Beiträge der Dissertation . . . 5

Glossary of Terms and Abbreviations **i**

List of Figures **v**

List of Algorithms **vii**

List of Tables **ix**

Bibliography **xi**

Chapter 1

General introduction

In this chapter we give a general presentation of this thesis. Beginning with the genesis of the problem that is to be solved, we define the concerns of our work and we provide an overview of our solution schemata. The problem of finding a CAD-supported partitioning method that is applicable to system-level specifications of automotive embedded systems (AES) is elucidated and its necessity for today's design process of AES is justified. The following problem definition clarifies the purpose and the goal of our investigations. It also describes the scope of this thesis. We then provide our solution schemata and an outlook on the possible expansions and the usability of the solutions.

1.1 Motivation

Today's vehicle manufacturers must produce pretty, reliable, safe-functioning vehicles with powerful engines, robust mechanics and high comfort, all that in mass-production, where stringent quality requirements go together with the demand for low costs and low maintenance needs, but high safety and security levels, high dependability, absolute reliability and short time-to-market. Thereto, vehicles underlie severe legal constraints such as the required environmental compatibility prescribed for example in [3–5]. Sophisticated embedded electronic controllers are used to cope with these constraints and to satisfy the steadily increasing expectations of the consumers. In this context, the quantity of automotive electronic- and software-based functionalities is likely to grow continuously, requiring efficient development and production methodologies to continue to build high-quality and cost sensitive vehicles. The required methodologies must not only assure the best product quality, but they must also facilitate the maintenance and finally the disposal of the resulting vehicles and increase the productivity of the vehicle manufacturers.

Automotive systems are actually very complex systems the software-based functionalities of which are distributed on several embedded components including electronic control units (ECU), sensors and actuators which do not only cooperate with each other, but also depend on each other. Figure 1.1 shows the typical architecture of the platform of a modern personal vehicle. In this car, there are five high-speed CAN (Controller Area Network) networks operating at 500 kbit/s: The power-train bus, called "*Antrieb-CAN*" in the figure, networks the engine, the gearbox, the transmission, the airbags, the all-wheel controller units and several braking, stability and steering assistance controllers. Another high-speed CAN bus connects the instrumentations controller ("*Kombi*" in the figure) with the rest of the system through the central gateway while the car exhaust management system ("*NOx,...*") uses the engine controller as gateway to the other parts of the system. This complex system around the high-speed CAN also includes the ESP-Cluster and the diagnosis bus (i.e. the one enclosed by dotted lines) that feeds the entire system. Concurrently, a low-speed CAN protocol running at 100 kbit/s is implemented for the infotainment and other comfort functions (here "*Komfort-CAN*"). In addition, the system runs two LIN networks. LIN stands for Local Interconnect Network [10]. The first LIN network clusters the multi-functional steering wheel sensing system ("*MFL*") and the second one interconnects the wipers, the mirrors,

the rain detecting sensors and so on. Thanks to the global connectivity enabled by the central gateway, the combination of these different networks can work like a unified system. For example, a power-train member function can communicate with the infotainment system to order the emission of audio signals corresponding to a particular alert. This example shows the dimension order of the complexity of AES. Nevertheless, even though the CAN [1] protocol is the most used for the inter-ECU communication in the automobile engineering, today's automotive systems often additionally include some MOST (Media Oriented System Transport [11]), FlexRay or Byteflight networks for multimedia, infotainment, safety-relevant and hard real-time applications.

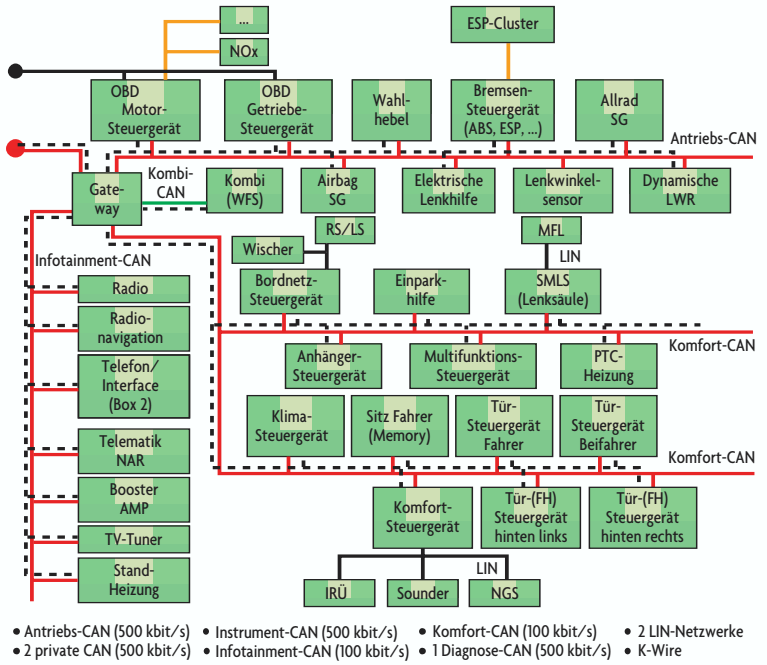


Figure 1.1: A subset of a modern passenger car's architecture: A network of sub-networks (source: Elektronik Automotive 01/2005, pp. 82)

It appears in figure 1.1 that each device is conceived for a given circumscribed set of features that generally participate all-together in the achievement of a defined functionality. On the way toward electronic all-rounder commercial vehicles, more and more electronic-actuated functionalities will be added to these already complex systems, with consequences to the design constraints, the product cost, the time-to-market, the design cost, the cost of ownership (e.g. energy and fuel consumption, maintenance efforts) and the quality goals (performance, safety, etc.). This issue is currently widely identified among the automotive design actors as a big challenge since it is clear that the number of automotive electronic devices cannot beneficially grow proportionally to the number of the required electronic-actuated functionalities. As we cannot stop the ever-growing consumers' requests for more vehicular features, the challenge can be roughly formulated as following: **How to continue to produce reliable and safe-functioning vehicles that provide every possible feature but are still salable and cost-effective?** The most emerging solutions propose radical changes in the design process, standardization efforts, harmonization of the development processes and the emergency of new technologies.

A change in the design process is necessary: The current AES design process relies on a component-driven approach. The electronic components and their software are designed each for a particular given functionality. New functionalities are introduced through the integration of new devices, resulting in too many devices and consequently a high inter-device communication, highly time-consuming system integration, poor performance over cost ratios due to the usage of more hardware resources than necessary. To implement more software functions on fewer components, it is necessary to change from the actual components-oriented development style to a system-oriented

development one. A system or at least a function-oriented design will enable the control of the growth of the number of the devices and simplify the functional scalability of the system.

Standardization is needed: In this industry, where the collaboration between OEMs and their suppliers is crucial, where the basic functions are so identical that they can be shared across the manufacturers, where a product is generally a variant of another, the need of a clear basis for the communication and a systematic reuse of solutions is obvious. AUTOSAR [22] and FIBEX [47] are promising solutions dealing with standardization efforts in order to afford reuse and interoperability within the automotive domain. AUTOSAR proposes standardized software components and interfaces while FIBEX proposes standardized configuration methods for the system communication.

The emergence of new technologies will evidently solve a certain number of problems: Prominent examples here include the evolving mechatronic- and by-wire-based innovations and the new revolutionary communication protocols. In fact, to cover the ever growing communication needs, automotive designers currently need more powerful communication facilities, i.e. with higher transmission rates, larger bandwidths, etc. This is achieved on the one hand by designing powerful and specialized communication protocols, e.g. TTCAN, TTP, MOST, FlexRay, Byteflight, LIN protocols, etc., and on the other hand by adding new buses/cables in the system. A very popular way of bandwidth optimization with respect to the state-of-the art in the design of AES is to organize their communication networks in hierarchies, i.e. cluster networks made of simple bus networks are joined on a backbone network through gateways. The hierarchical organization induces the following four dimensions of the system design, in the ascending order:

1. The devices with their components
2. The simple bus networks, i.e. a bus networking a set of devices
3. The cluster networks, i.e. those that are connected on the gateways
4. The complete system network including the gateways and the cluster networks

The architecture of the vehicle shown in figure 1.1 is a typical example of an hierarchical network organization: The devices are placed on bus networks according for example to their level of criticality, their similarities and the dependencies between their functions, their consumption of power, etc. Then the bus networks are bundled into cluster networks (e.g. see *Komfort-CAN, Antriebs-CAN, Infotainment-CAN, Kombi-CAN,* etc., in figure 1.1) according to their functional domain (power train, chassis, safety, multimedia, telematics, MMI, comfort). The cluster networks finally communicate through gateways and other central nodes, building the complete system. Although these solutions appear to solve the problem, they are definitely not sufficient! The real-time performance of CAN and others is limited, the transmission medium's capacity is limited, etc. While it is necessary to reduce the system cost despite the demand for higher AES performance, adding buses implies to extend the cable harness, that means an increase of the weight, of the energy consumption, of the sales price of the vehicles and possibly the appearance of a wide range of technical problems.

1.2 Problem definition

The design of an AES goes through several conceptual levels. Each level is a refinement of the preceding one. As shown in figure 1.2, the development of AES is concerned with the requirements engineering, the design and the implementation of the system functionalities as well as the validation of the system.

The earliest phase of the creation of an AES begins with the treatment of the requirements. This includes the capturing and the organization of the requirements, the dissolution of inconsistencies and the transcription of the requirements into functional and non-functional objects as well as the enumeration of the related constraints. Actually, the requirements engineering has

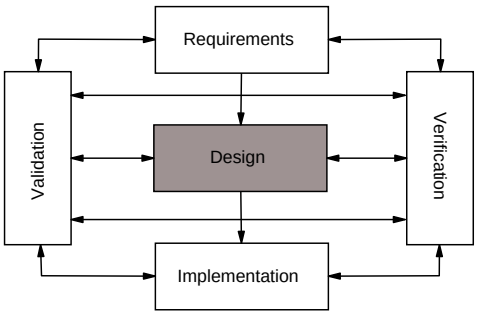


Figure 1.2: Activities in the AES development

proposed notable solutions to identify, capture, analyze and manage the requirements in the automotive engineering domain [92,108]. Some advanced solutions are successfully integrated in CARE (Computer Aided Requirements Engineering) tools, e.g. DOORS (www.telelogic.com), Analyst Pro (www.analysttool.com), CARE (www.sophist.de), ClearSpecs Composer (www.livespecs.com), RequisitePro (IBM Rational), etc. But, these solution are all discrete solutions, i.e. totally separated from the following design activities. Unfortunately, independently of its quality, a discrete requirements engineering solution is not the most viable, since it does not enable a continuous system engineering. Furthermore, as we pointed out in [82], as long as there is no solution proposing formal specifications of the requirements, the requirements engineering will still be a very hot research area. However, these problems will not be discussed in this work.

In the design phase of the development of an AES, the system functionalities are specified and then, implemented on a hardware platform following the design constraints. Here, decisions are made about the logical architecture of the system, the composition and the topology of the hardware platform on which the system's application will run as well as the coding and the deployment of the functional specification on the platform. As the functionalities of an embedded system can be implemented on different architectures built each of different hardware components, the choice of the hardware and the quality of the implementation are decisive for both the economy and the performance of the system. An optimal resource usage can considerably reduce the cost of the AES platform. We can thus achieve the goal of cost reduction if we reduce both the design cost and the hardware usage of AES.

We can reduce the hardware usage if we optimize the architectures of the system in a way that will reduce the quantity of the hardware needed to run the required functionalities, and use optimally the hardware units installed in the system and the cables provided for the inter-device communication, i.e. The operation by which the architecture of an AES is designed is called the partitioning. The partitioning aims at finding the most cost-sensitive hardware platform and distributing the system working load within the available resources of this platform so that the functioning of the system is optimized by concurrently avoiding resource underutilization. More concretely, during the partitioning, the system architect is concerned with questions like:

- Which hardware components are needed to realize the functionalities of the system?
- How many devices (ECUs, sensors, actuators, gateways) are needed to implement the given functionalities?
- Which hardware units (processing units, memories, etc.) will be installed in each device?
- Where must each device be geographically located in the vehicle?
- Which communication systems are optimal for the chosen configuration of the platform?
- etc.

Thus, globally, the partitioning involves three activities:

- The allocation, i.e. the choice of the hardware components of the platform,
- the mapping, which is the assignment of the elements of the functional specification of the system to the components of the platform and
- the deployment, i.e. the distribution of the computing power and the memory space of the platform among the elements of the functional specification.

A good partitioning must minimize the usage of processing units and memories as well as the quantity of cables used for the inter-device communication. However, to achieve a good architecture, it is necessary to design all the parts of the system simultaneously in order to ensure that they will conjointly meet the given performance and economic goals of the design. The concurrent design of the components of an AES will allow the coordination of the resource allocation across the boundaries of the devices, facilitate the system integration and enable the system scalability with positive consequences on the economy and the performance of the system. The concurrent design is made possible only by a system-oriented design style. Regarding the system orientation, we roughly distinguish three levels of conception in the development of AES:

- The functional level, where the functionalities of the system are specified in terms of coarse and abstract functions.
- The implementation level, where the functional architecture of the system is defined in terms of communicating software components, tasks or processes.
- The platform level, that deals with the configuration of the platform's physical devices, their topological positioning, the deployment of the software, the cabling of the devices and the transport of electric signals for their communication.

Intuitively, the reduction of the costs of AES can be achieved by reducing or at least slowing the growth of the number of devices in the vehicles. Once more intuitively, this can be done by acting on each of the above conception levels. For example, one can imagine following solutions:

1. Re-engineering the platform in order to reduce the number of devices. This can be achieved by grouping the devices that are close in order to build aggregative devices.
2. Building consequent functional modules with the software components defined in the implementation level or with the functions defined in the functional level of the system's specification.

Two devices are close if they have some common approaching properties, e.g. when they incorporate functions that are related. Two functions are related when they communicate with each other or when they underlie approaching constraints, for example when they can share resources, share data, have common accessors or when they underlie constrained relationships with each other [81], etc.

– Re-engineering the platform means to revise the system's organization and then, either merge very close devices in order to build larger devices or cluster them in order to place them on the same bus. Only, building larger devices through the merging of several smaller ones is problematic since the individual processing capability and the memory space in a device are limited. Moreover, this approach might produce heavily unbalanced computation and communication loads that may prevent the system to take full advantage of the capability of the platform and compromise the scalability and the maintenance of the system (e.g. the exchangeability of the system components). On the other hand, placing too many devices on the same bus involves the risk of non-economic topological placements. The available bandwidth for the communication is limited and the required communication rates might be different from one device to the other.

– Following the second solution, we can process either by splitting too large functional components or by grouping highly related functions to build more cost-effective modules. However, as

the functional level specifications are too rough to support the necessary analysis, we can do this reasonably only at the implementation level. Thus, following this approach, software components will be assigned to a location on the hardware platform depending on their contents and their environments. Heavily related software components will be placed on the same bus or better, on the same device. This solution needs a wide overview on the functional specification and the properties of the software components including their performance requirements and their costs, but it is the most realistic solution. However, with this approach, the partitioning process depends on the substance of the software components, i.e. their granularity and their behavior. Depending on the granularity of the software components, the partitioning process might include the mapping or not, i.e. if the software components are all designed as tasks that can run each as a sequence of uninterrupted instructions on a processor, then we just need a judicious definition of the corresponding platform and an intelligent deployment of the software components on the processors and the memories. Else, we must firstly distribute the software components among the devices before deploying the contained tasks and processes on the available processors and memories. In this case, the purpose of the partitioning is as shown in figure 1.3 to find the best sample of devices and the necessary communication systems, and determine the software components that must be implemented on the same bus or on the same device so that the cost and the performance of the system can be optimal with each intelligent deployment.

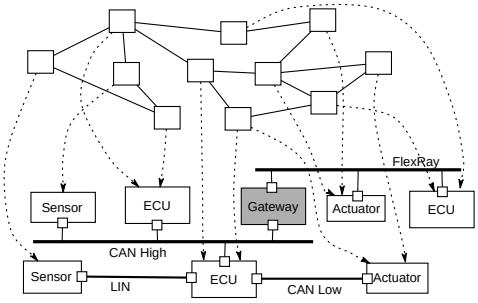


Figure 1.3: The partitioning with coarse-granular software components

In a realistic AES design process, it is absolutely not possible to specify the functionalities of the system in the form of schedulable chunks of behaviors at the system-level, but rather, in the form of software components implementing each a function of the application software that is made of several tasks that can run concurrently or sequentially. This means that we have to consider the partitioning of coarse-granular software components (see figure 1.3). However, as we map the entire content of each software component and not a part of it on a device, it is important that the software components are atomic components. Otherwise, it will be necessary to split them in order to achieve atomic components. In the automotive engineering, this important issue is commonly perceived like a creative process. The designer is responsible to define the system's functional modules following his feeling an his intuition. Due to the lack of methodical support, the outcome of this phase of the design is totally dependent on the experience of the people working for the system development. Excepting the current AUTOSAR initiative to identify the software components that are common in AES in order to standardize them as atomic entities, we are not aware of any systematic or methodical approach to define the atomic functional components of an AES. But, we think that a good deal of the expertise provided by the SOA (Service oriented Architecture) domain can help here.

However, we consider in this work that the system's application is made of atomic software components. Note that this is a pretty realistic assumption, since the partitioning will not begin as long as the functional components of the system are not considered to be each atomic. Atomic in this context means that a component can be assigned to a device only entirely or not. The corresponding partitioning process is shown in figure 1.4. During the mapping, the system architect assigns each functional component of the system to a given set of devices depending on the

redundancy requirements of its implementation. This activity results into clusters of functions that represent the logical devices of the system. Two components that are assigned to different devices must communicate through an inter-device communication channel. As the automotive devices usually communicate through bus networks running frames-oriented communication protocols, the mapping must pack the inter-device communication data into the frames and assign the frames to the available inter-device communication channels. In contrast with the mapping, the deployment deals with the scheduling of the tasks and processes within the devices. The deployment can integrate a typical software/hardware co-design in which it is decided which tasks will be implemented in software to run on processors and which ones will be implemented in hardware, e.g. as ASICs. However, the mapping and the deployment are interdependent.

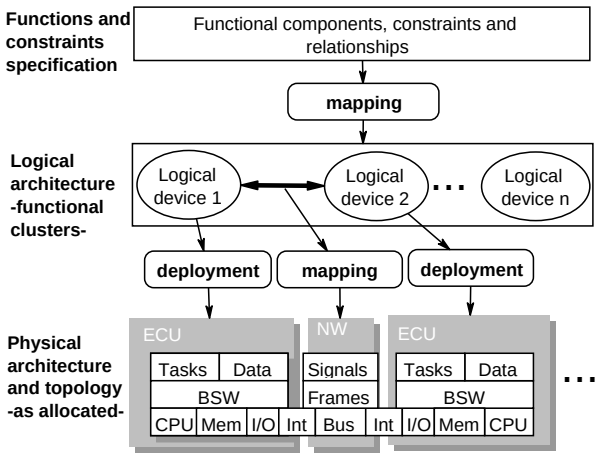


Figure 1.4: The allocation, the mapping and the deployment

The order in which the partitioning operations are executed has an important impact on the result of the partitioning. In a partitioning process, one can decide to execute the allocation, the mapping and the deployment in this order or not, while another one might prefer to intertwine them. However, with coarse-grained software components, the mapping will normally precede the allocation. These operations are based on the runtime resource consumption of the elements of the system specification, e.g. the space needed to store the code, the heaps, the software data and the stacks as well as the computation power required by each functional component, its consumption of energy, the magnitude of its collaboration with the other members of the device and a full range of constraints and other relationships between the components of the functional model, such as those induced by the strategic concerns of the AES design. If these attributes are given with sufficient details so that we can design the necessary platform, then we should begin with the allocation, then the mapping and the deployment. This is the most straightforward execution of the partitioning. If not, we have two possibilities:

1) We can begin with the mapping, i.e. we firstly cluster the software components in order to build the logical devices. Then, based on the detail specifications of the logical devices, we determine the platform for each device and then for the whole system. The deployment can then be done normally. With this process, we can achieve the best possible economy of the hardware since we chose the hardware platform depending on the calibration of the functionalities of the different devices. But, it is quite difficult to define the right dimensions for the logical devices. Without this information, there is no means to stop the clustering at the right point.

2) The second possibility is to separate the allocation into a gross-allocation and a detail-allocation. With the gross allocation, we determine the devices and their capacity. With a detail allocation, we determine the effective components of the devices. The capacity of a device can be given defined in terms of computing power or memory capacity. Given this information, we have the sizes of the devices and thus the stopping points of the clustering. The partitioning can therefore

begin with a gross-allocation, then the mapping, and with the result of these operations, we proceed a detail-allocation and then the deployment. This process is for example the solution if we have a stock of devices at our disposal. It might involve many loops of allocation, mapping, detail-mapping, deployment and evaluation, but it is the most realistic process since the partitioning is normally given the functional specification of the AES under construction, and the system designers know approximately the capacity of the available devices.

In this work, we refer to a partitioning process that follows the above second process, i.e. it firstly executes the allocation, then the mapping and finally the deployment. Since we are concerned with the design of the architecture of the system, the allocation is reduced to the determination of the number of devices (i.e. gross-allocation) while the internal equipment of the individual devices and the corresponding deployment are partially under the responsibility of the components suppliers. Following this separation of duties, the most actual objective of the system architecture design is to minimize the inter-device communication. With frame-oriented communication protocols, this objective is to be achieved by reducing the number of frames used to exchange information through the buses. To do that, the mapping must assign the most heavily communicating components of the functional specification to the same device. However, reducing the number of frames is not the unique optimization goal of the mapping. Within this design process, the mapping is heavily constrained by the allocation. The mapping must result in executable partitions, i.e. each devices' functionality must be schedulable and respect the capacity of the corresponding device. Furthermore, each device functionality must use the hardware that is installed in the corresponding device optimally, with respect to the defined room that must be reserved for the possible future extensions of the device functionality. The allocation depends on the working load of the system. We can only determine the equipments of the devices for a given working load if we can measure the execution time, the size of the software code, the magnitude of the communication of the components of the system's functional specification, etc. This information can only be provided if the partitioning is given a consequent description of the behavior and the functioning of the system, what is not always the case. In fact, depending on the functions that are assigned to a device and the design constraints, an automotive device might be equipped with all kinds of processing units (including ASICs, micro-controllers, DSPs, etc.) as well as all kinds of memories and several intra-device communication systems (e.g. SPI, I2C, etc). Figure 1.5 shows a possible result of the detailed deployment with an ECU for which the system partitioner only defined the application software and the communication matrix. For all these reasons, the scope of this work is limited to the mapping. In the remainder of this work, the partitioning will refer to the mapping whenever there is no need for precision.

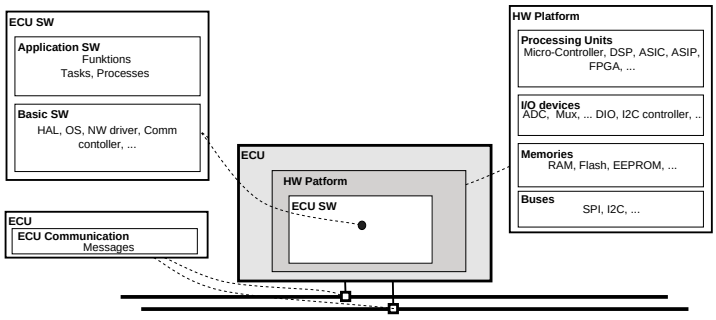


Figure 1.5: Detailed allocation and deployment

The design cost is also an important factor of the price of a vehicle. We will reduce the cost of the design if we provide efficient and CAD-supported techniques for the implementation of AES. In the current practice, the partitioning is done manually by highly experienced OEM designers, usually called system integrators. The partitioning is currently limited to the addition of the new software components on the existing system without changing the precedent contents of the devices. When the existing devices are overloaded, the system architect generally decides to add

new devices to implement the new functionalities. This optimistic approach of the partitioning is justified by the fact that the existing systems are well-functioning and reliable configurations with stable communication matrices. A new design of the system architecture is practically equivalent to a design from scratch, economically unsupportable in this industry where the competition is extremely hard. In fact, the AES domain is a fast-evolving engineering field where new features become rapidly customary. The time to market is vital for each OEM. However, the direct consequence of this practice of the partitioning is the difficulty to enhance the system's functionality. This shortcoming is illustrated by the excessive number of buses and processors installed in the new vehicles. Thereto, during the partitioning, a system architect must take hundreds of often contradictory and competitive constraints into account. Keeping this information for a long time in mind is not easy for a human intelligence. Furthermore, done per hand the partitioning is guided by vague estimations and is so poorly documented that it is difficult to add new functionalities into an already partitioned system, e.g. after delivery, without adding new hardware. A further important disadvantage of manual partitioning is the fact that it is not possible to examine a large number of system architectures. The design space is reduced to the solutions that are familiar to the designer and those that are deemed by his feeling to be potential good solutions. In this context, new, innovative and revolutionary architectures cannot be realistically expected. A CAD-supported partitioning will allow automotive systems architects, so-called systems integrators, to investigate new architectural options. Automated partitioning will be time-saving and produce well-documented and good system architectures since much more solution alternatives can be consulted. If provided, this will be an effective contribution to the goal of optimizing the economy and the performance of AES.

1.3 Procedural method to solve the problem

For the partitioning, the exploration of the design space allows designers to find optimal implementations of the system specification by analyzing various alternatives of both the architecture of the system and the hardware platform. This requires powerful and complete models. The existing approaches for the partitioning input very low-level, fine-granular specifications (e.g. logical and arithmetical operations or simple assignments) at the level of abstraction given for example by the programming languages like C, C++, assembler, Java and similar specification languages. Unfortunately, because of the complexity of the automotive electronics, this dimension of granularity is difficult to achieve when following a system-oriented design scheme. System-oriented development requires a global view of the system, resulting in the handling of very complex models. As a special domain of interest, important works have addressed the specification of AES, producing appreciable results. Near general-purpose embedded systems-qualified modeling tools (UML [39, 60], SDL [8], SysML [12],...), domain-specialized modeling languages have been proposed for the development of AES (e.g. EAST-EEA[40], AADL[16], AUTOSAR[22], etc.). But as these solutions were mostly focused on the definition of modeling languages, neglecting the substance of modeling itself, i.e. its potential methodological support for the design process, it is necessary to examine if AES specifications presented at this level of abstraction are apt to support the partitioning of the system.

This observation leads to the structuring of this work in two main parts. We first need to define the input specification for the partitioning. This must be a model that fulfills the requirements of the partitioning. In this part of this work, we need to investigate the level of support that is provided to system architects by the existing AES modeling solutions with regard to the partitioning. Here, we concretely need to answer questions like:

- Which information is needed in a specification to support the partitioning?
- Which modeling features are needed to provide this information?
- Do the actual modeling techniques provide these features?

- How capable are the modeling languages used?

We would appreciate if there is a modeling solution that fulfills our expectations. Else, we must define a convenient input specification for the partitioning. In the second part of this work, we will design the partitioning algorithms that will find the optimal system architecture. The two parts are described with more details in the following subsections.

Defining the input specification for the partitioning

The development of AES has incontestably experienced a great leap forward during the last decade. On the way toward its maturity, the AES design has adopted the model-driven development scheme. The purpose of model-driven system development is to use models to study the artifacts of a system before building it. Model-driven development offers an effective way to decrease the technical and financial risk of try and error and improves the savings of design time and system resources, etc. and improves the quality (reliability, soundness, performance, electromagnetic compatibility, etc.) of the system. Furthermore, model-driven development has the potentiality to boost the innovation, afford collegiate work and simplify the product maintenance. All these concerns are quoted to be vital in the automobile industry. Unfortunately, the state of the art in modeling embedded systems in the context of the automotive engineering does not yet allow the designer to take the best possible advantages from model-based development. In fact, even if modeling is current practice for today's automotive systems designers, models are still considered as simple description and communication media, although in the context of hard competition that rules the automotive industry, modeling can unacceptably continue to be a task that unnecessarily consumes time instead of being helpful and easy.

Hence, even though models are abstractions of the reality, useful specifications must highlight the system characteristics, motivate the design options and facilitate the design decisions. In brief, a model should bear all necessary information needed for the subsequent design operations, for example the partitioning. AES modeling is concerned with the specification of the architectures, the communications, the behaviors, the constraints and the non-functional requirements of the automotive-embedded software and its electronics as well as the process of the system development and the contained transitions. Architectural models include the description of the structure of the system and its topology while behavioral models describe the system's operation. Transition models are needed to capture the evolution along the design process. Modeling the transitions includes for example the description of the mappings as well as the rules governing the relations between the abstraction levels, the instantiations, the configurations and the deployment. However, the value of a modeling solution that is eligible for the AES design depends not only on the way it handles these artifacts but also on how it considers the modeling of the automotive-specific domains of interest such as hardware platforms, product lines, non-software components (e.g. driver's interaction), and vehicles environment (e.g. the road). High resolution, clear encapsulation, execution and synthesis tools are needed in both the high- and the low-level design, while clear modularity is essential in the higher levels of the design to support the partitioning.

We have evaluated the most common modeling tools that are usual in the automotive design on their ability to support the partitioning. We examined low-level languages, i.e. programming languages and hardware description languages (HDL), and high-level languages including general-purpose modeling languages (UML, SDL, SysML) and automotive domain-specific modeling languages (EAST-ADL and AUTOSAR). Unsurprisingly, we found that when the design follows a top-down strategy, none of the above languages can be expressive enough to be used efficiently for all purposes along the design process, since each of them offers in reality only a limited set of features. Otherwise, we are not aware of the existence of an all-rounder general-purpose modeling language. At each step of the development process, the most adequate language should be selected depending on the actual conceptual layer, the level of abstraction and the intended use of the model. Within a system-oriented design scheme following a top-down strategy, the design of AES begins with high levels of abstraction for which the modeling languages like UML, SDL,

SysML, EAST ADL or AUTOSAR are adequate. Although all these languages claim sufficient orientation to the implementation, they still remain very abstract and lack synthesis and execution tools. However, as domain-dedicated languages, EAST ADL, SysML and AUTOSAR provide the most convenient features and the best precision needed to model automotive AES at the high levels of the design, but they remain very insufficient to support the automated partitioning of the system. Firstly, because they are not synthesizable. Secondly, the semantics of ports, interfaces and connectors are fuzzy.

As we are dealing with a domain where the partitioning shall be done on high-level models, a promising solution to the first drawback is to combine these languages with synthesizable languages such as programming languages, HDLs, etc. Two questions arise here: How should the languages be combined? and What are the best combinations? These questions are not in the focus of this work since even the best combination will not be the ultimate solution for supporting the partitioning of system specifications at high level. However, if EAST ADL and AUTOSAR languages are enhanced with the missing capabilities, i.e. precise computations and communication modeling tools, accurate time and data handling, etc. so that the QoS of the model elements can be extracted and analyzed, then they will represent appreciable solutions to build partitioning-compliant models of AES. We defined a modeling solution, the FN-for functional network-, that is fit for a CAD-supported partitioning at high level by adding some rules to the concepts imported from EAST ADL and AUTOSAR. In addition to the partitioning-friendly features provided by these languages, i.e. components detachability, standardized interfaces, QoS modeling, the FN provides clear screening of the communication paths and tracing of the communication data.

Partitioning AES

With the FN, AES functional specifications are modeled in the form of separable building blocks, where each building block represents a functional component of the system. As the FN allows to clearly identify the boundaries of each model element, it enables to move each component of the system and assign it individually to a given device. Furthermore, due to the concepts of ports and interfaces, the communication data can be properly specified with the FN, at least statically. In fact, although the FN is sufficient to model the structure of the system, it is lacking the appropriate concepts to describe the behavior of the system. Instead of fine-granular and high-resolution modeling solutions that are desirable for the allocation, the FN, like all the standard solutions that are adequate for the system-level specification of AES, proposes high-level modeling tools like state machines or communication, interaction, sequence, data flow diagrams, etc. to specify the behavior of the system. Because of the low resolution of these behavioral modeling tools, it is not possible to produce a system specification that can be used for a detailed system allocation, and much less for the deployment. In fact, the allocation and the deployment rely on the QoS attributes of the model elements. These attributes include the runtime resource requirements of the functional components, e.g. the code size, the execution time, the communication load, etc.

In order to specify the system behavior so that these attributes can be extracted, we need more detail specifications. These can be provided only for a single component that is also not too complex if we do not want to face an order of explosion of the size of the specification that will lead to the loss of visibility and reduce the navigation within the specification. Note that this is the main reason why the mapping precedes the deployment. After the mapping, we can produce manageable behavioral models for each device and use them for the detail allocation and the deployment. Although this strategy gives rice to much more loops of allocation-mapping-allocation-deployment-evaluation-allocation-mapping-... than with a straightforward process, we adopted it because of the incapacity of the actual state-of-the-art in the modeling of automotive systems to provide more detailed specifications of the behaviors at the high level of the design. However, even if the runtime resource requirements of the functional components cannot be predicted with the FN, we can appropriately use the syntax of communication diagrams, sequence diagrams and

timing diagrams to specify their communication at the level of abstraction that is inherent to an AES system-level, with a precision that can be sufficient to determine the quantity of the data exchanged between two components, the frequency of the data exchange, the timing organization of the communication and the constraints on the communication. This solution provides sufficient means to measure the closeness between the system's functional components, essential to guide the mapping. We sampled the information extracted from this solution in a Components Data Flow Machine, the $CDFM$.

The $CDFM$ is a modeling format that enables the analysis and particularly the synthesis of the data flow within the system's specification. We solved the problem of defining the optimal partition of the input functional specification of an AES using a combination of constructive and iterative partitioning algorithms. A constructive algorithm was used to generate a partition that was refined iteratively. As the number of devices to be installed in the system to run the required functionality was not predefined, we designed a Quality Threshold (QT) partitioning algorithm to construct the initial partitions. The threshold is given by the capacities of the allocated devices. The refinement of an initial partition is achieved by the means of a specific adaptation of the Kernighan & Lin algorithm [86] whereas the quality of the partitions is based on the execution of a bin packing algorithm that realizes the multiplexing of the communication frames, i.e. a mapping of the inter-devices communication data on the communication channels of the inter-devices communication network.

1.4 Contributions

The goal of this thesis is to provide a design process and the partitioning algorithms that will enable to automatically determine the architectures of AES. In order to implement a CAD-supported partitioning tool for the system-level architectural design of AES, it is necessary to define adequate input models, formalize the relationships between the different models and within the models, formalize the partitioning constraints, design the partitioning algorithms, optimize them, apply them on the inputs and evaluate the results of the partitioning. To achieve these goals, this thesis provides a broad overview of the design concerns of AES. It begins with a clear definition of AES that includes the identification of the relationships between automobiles, electronic and software systems that outline the importance of the latter in vehicles. Then, a detailed presentation of the reasons that motivate the need of CAD-based partitioning tools gives rise to the current challenges identified in the automotive industry. The solutions proposed from both the academia and the industry to cope with these challenges are evaluated and commented. A partitioning method is provided.

Thus, although this thesis focuses on the design of the architectures of AES, its most important technical contributions include:

1. A valuable sample of background information relative to the constitution of AES and their development. This includes a particularly profound review of the specification, the modeling and the architectural design issues of automobiles' E/E systems and the related design processes as well as the documentation of the state-of-the-practice, the actual requirements, the actual and the feature challenges of the AES design.

2. The definition of a framework for the evaluation and the classification of the modeling solutions that are used in the development of AES.

3. The definition of reference modeling solutions that are fit to support the automatic definition of AES architectures. This contribution includes a revealing evaluation and a categorization of the main modeling languages that are used in the AES design with regard to their ability to support the implementation.

4. A study of the interdependencies between E/E design processes, E/E models, abstraction levels and E/E design operations.

5. The definition of a novel, original modeling format for the synthesis of AES high-level specifications. This format can also be used for validation purposes, e.g. for simulation, verification or test.

6. The definition of a partitioning process that is adapted to AES high-level system specifications. This contribution includes the formalization of complex closeness factors in the context of a CAD-supported design of AES architectures and the development of powerful clustering and partitioning algorithms for the design of AES architectures.

7. The definition of the means for the evaluation of AES architectures and the related metrics. A correlated but also important contribution is the formalization of a frame multiplexing approach for the packaging of signals within frames.

1.5 Publications

By now, we have published some results of this thesis in international conferences [81] and [85]. Some others are actually submitted for publication [84], [83] and several papers are in preparation. However, the contents of each of these papers are presented in details in this document as described in the following section. We resume our publications in relation with this thesis as follows.

- Partitioning Metrics for improved Performance and Economy of Distributed Embedded Systems. A. Kebemou in IESS proceedings on IFIP TC10 Working Conference, pp 289-300, Aug. 15-17 2005.

- AutomotiveArchitect: A Partitioning-Centric Modeling and Architectural Design Environment for Automotive E/E Systems. A. Kebemou and I. Schieferdecker in INFOS 2008 proc. of the 6th International Conference on Informatics and Systems, 27-28 March 2008, Cairo, Egypt.

- Evaluating Modeling Solutions on their Ability to Support the Partitioning of Automotive Embedded Systems. A. Kebemou and I. Schieferdecker, International Conference on Embedded and Ubiquituous Computing, Taipei, Taiwan, 12.2007.

- A Model-Based Design Approach for the Partitioning of Automotive Embedded Systems. A. Kebemou and I. Schieferdecker in INFOS 2008 proc. of the 6th International Conference on Informatics and Systems, 27-28 March 2008, Cairo, Egypt.

- The Components Data Flow Machine: An Intermediate Modeling Format for the Design of Automotive E/E Systems Architectures. A. Kebemou and I. Schieferdecker in DIPES 2008 proceedings on the IFIP Working Conference on Distributed and Parallel Embedded Systems, Milano, Italy, Sept. 7-10,2008

1.6 Scheduling of the thesis

As illustrated in figure 1.6, the rest of this thesis is organized as follows: We provide an illustrative definition of AES in the chapter 2. We begin with an outline of the relationships between AES and embedded systems. Then, based on an example of functional inter-actions in AES, we introduce the automotive communication systems and we discuss the purpose of the automotive connectivity. Chapter 3 gives an overview of the usual design methods of embedded systems and the related design activities. Also in this chapter, we compare the sequential design method with the concurrent design method in order to provide a basis on which an efficient design process can be proposed for AES. This is done in chapter 4. The second part of this thesis is concerned with the purpose of modeling. To mark the beginning of this part, chapter 5 presents the state-of-the-art in modeling AES. In chapter 6, we evaluate the modeling solutions used in the AES design on their ability

to support the partitioning. As these modeling solutions cannot provide the best support for the partitioning, we define our modeling solutions for the desired inputs in chapter 7. Then, the $CDFM$, the modeling format for the corresponding synthesis models is defined in chapter 8. In the third part of the thesis, we investigate the partitioning of $CDFM$ models. As $CDFM$ models are formalized as graphs, the first chapter of this part, i.e. chapter 9, gives an overview of the state-of-the-art in the partitioning, particularly the partitioning of graphs. Then our partitioning algorithms are described in the chapters 10 and 11 and the results of their applications are summarized in chapter 12. A general conclusion, given in chapter 13 closes the thesis.

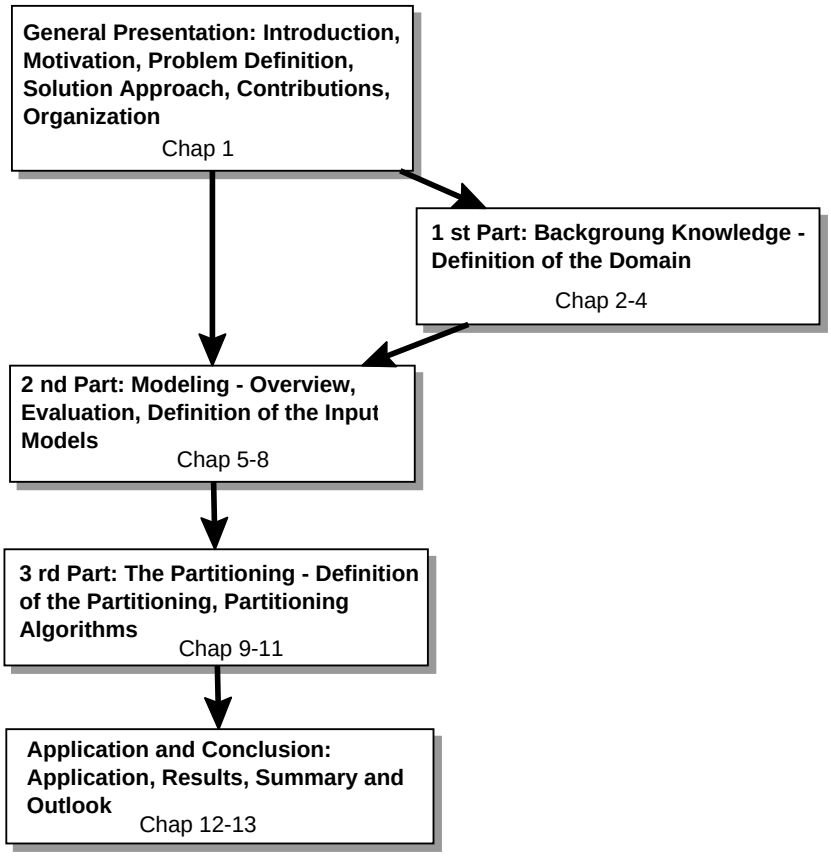


Figure 1.6: Vertical logic of the thesis

Chapter 2

Automotive Embedded Systems

In this chapter, we present the relationships between automotive systems and embedded systems. As AES (automotive embedded systems)[1] *will be the subject of investigation of this work, it is necessary to define the term AES in a way that canalizes every imaginative representation of the discoursed item. We begin by defining the term embedded systems. Progressively, we show how AES are concerned with embedded systems and then following an example, we give a survey of the networking of embedded systems in today's vehicles. We discuss the means by which interdependent automotive embedded functions that are geographically separated communicate with each other. Then we give a brief survey of the different communication protocols used in the automobile industry. The most famous of them, the CAN protocol, will be more extensively presented in the part of the work dealing with the problem resolution.*

2.1 Automotive electronics

2.1.1 Embedded systems

Most of today's life facilitating instruments embody a processing unit which undertakes the computation and control tasks needed to realize their functionalities. Telecommunication systems (e.g. mobile and fixed sets, routers, switchers, ...), household appliances, multimedia and medical equipments, manufacturing facilities for the industry, transportation systems (aircrafts, trains, automobiles, ...), etc., embed such electronic-based components. The term embedded system is used to designate a computer system that is not perceived as such. It is a computer system which is integrated in a larger system that hides it from the user. Originally, embedded systems were specialized on a predefined functionality and were not able to work independently from their environment. They could only operate when triggered by another system, e.g. a sensor or the user. This conception of embedded systems refers to a piece of silicon implementing a given functionality. In the last decade, the technological evolution has afforded the emergence of electronic devices that integrate several functions. Embedded systems are no more restricted to single-function products that are conceived for a fixed and predefined functionality. They are becoming very complex and they can no more be conceived to react exclusively to the inputs of other systems. Modern embedded systems are increasingly requested to control complex and heterogeneous systems. For example, a high-definition TV handset controller must incorporate control and signal processing functions, audio and video data processing features, wireless and wired communication modules, etc. Such an embedded system can contain very heterogeneous functional blocks including DSPs, ASICs, memories, converters, etc. and sometimes it even might integrate a whole system on a single piece of silicon. The latter is referred to as a SoC (System-On-Chip).

As the emergence of billion-transistor chips is just around the corner, the next generation of SoCs will implement more complex functions. Actually, Network-On-Chips (NoCs) implement

[1] In this context, automotive embedded systems (AES) is synonymous with automotive E/E systems

switches, routers as well as different communication formats and protocols to connect SoCs on a single chip. NoCs will enable the integration of an exceedingly large number of computational, logic, and storage blocks in a single chip. Embedded systems are mostly used in systems that are typically mission-critical, but also most of the time safety- and business-critical. Therefore, embedded systems are expected to operate reliably and faultlessly. They generally underly very high performance and cost requirements. Thereto, to cope with the actual changing world, embedded systems must be scalable and upgradeable. This is why they are more and more implemented in software.

2.1.2 Automotive and embedded systems

The automobile is originally a machine construction product. Until 1967, the engine ignition and the lighting systems of personal vehicles were controlled exclusively by electrical circuits [24]. The electrical components were connected with their actuators and their consumers by means of dedicated peer-to-peer cables. With the electronic fuel injection controllers, the first microprocessor-based devices appeared in personal cars in the '70s when the worldwide oil crisis boosted the need for control devices to reduce the fuel-consumption. The subsequent introduction of the electronic cruise control, the central doors locking systems and the gear steering units marked the beginning of massive integration of electronic units in the automobile. Over the years, the technological evolution has turned the vehicles into high-tech systems. Although the first image of a car might still be that of a large piece of hardware, one must be aware that there are hundreds of computing systems and millions of software code lines implemented in each modern vehicle. Today's automobiles are heavily equipped with computers and associated software to control everything from the engine to the brakes, the cruise and all kinds of new on-board navigation and communication systems.

Each electronic control unit (ECU) embeds one or more processing elements and memories, each of which is either a general-purpose, an application-dedicated embedded system or a combination of both. Figure 2.1 shows a layered architectural representation of an ECU. From the bottom upwards: The first layer contains the material components used in the ECU, e.g. the processing units (e.g. DSP, MCU, ASIP, etc.), the IPs (e.g. ASIC, ASSP, FPGA, ...), the memories and the communication media (buses, cables). The MCAL (Micro Controller Abstraction Layer) contains hardware-specific software components called drivers. A driver is a software component that enables to interact with a hardware device. Possible contents of the MCAL include the MCU drivers, the memory drivers, the I/O drivers, the drivers for the communication controllers, etc. The HAL (Hardware Abstraction Layer) contains further hardware-closed software components that serve either as pilots for the I/O devices and the memory devices or as interrupt and communication managers. The layer with the operating system (OS) builds an abstraction layer between the application software and the hardware details. It provides the API services that are necessary to access and manage the resources (i.e. hardware, drivers, pilots, service managers). The OS provides the scheduling and the interrupt services, the resource/memory and tasks management services as well as the task intercommunication and synchronization solutions that are needed by the application that runs on the ECU. In the realm of embedded systems, OS are generally required to be RTOS, i.e. they must provide deterministic timing behavior.

Automobile electronic devices (ECUs) are rarely stand-alone components. They must communicate to fulfill their duties. To achieve efficient communication, automobile ECUs are networked on different architectures (bus, ring, star, tree) running several communication protocols, e.g. CAN, MOST, LIN, FlexRay, etc. We identify the collectivity of such networks and the connected electronic devices as automotive embedded systems (AES). Automotive embedded systems thus consist of sensors, control units, actuators and several communication networks. Sensors are used to collect the information about the operational and environmental conditions of the vehicle, e.g. temperature, speed, lightness, etc. Most sensors in automotive systems are "intelligent", i.e. they are able to transform the collected physical or chemical values into electrical values that can be

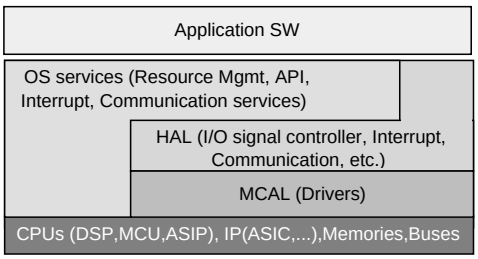


Figure 2.1: ECU architecture

used by a computer system. Aside from the sensors, multiple keys and buttons are used to transmit the instructions of the user to the control units. A control unit processes the received data by running corresponding algorithms. Then, depending on the results of the computations, it triggers an actuator. An actuator is the electronic part of a vehicle component that realizes what the user is seeing and feeling. Examples of actuators include motors, fans, air blowers, switches, window regulators, brakes pushers, sound emitters, door locks, fuel injectors, etc. Figure 2.2 shows a set of electronic devices and the communication networks that constitute an AES. As the value of a vehicle is increasingly determined by the number and the quality of the integrated electronic-based features, the automotive industry is moving from an electromechanical-based industry to an IT-oriented industry. The embedded systems design will thus continue, more than in the past, to be a first-class player in the automotive engineering.

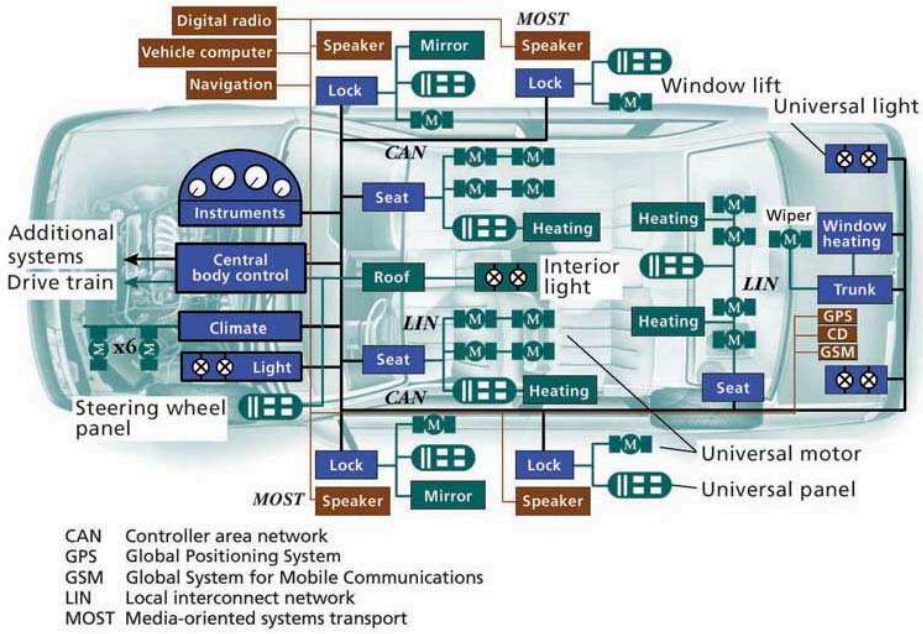


Figure 2.2: Automotive embedded systems

2.1.3 Example: The Active Cruise Control

Since the successful introduction of the first electronic devices in automotive systems, the demand for automotive electronics is steadily growing. Today's automobiles are heavily equipped with ECUs that perform the control and management operations. A personal car in the high class contains already about 80 ECUs. The design of each ECU is a highly challenging task and networking them turned out to be at least as difficult. The Active Cruise Control (ACC), also commonly referred to as "Adaptive Cruise Control" is a typical illustration of the complexity of automotive

electronic systems. The ACC is an enhanced version of the automatic speed limitation control, also called "Tempomat" in some countries. In addition to the basic functions performed by the previous automatic speed limiter, i.e. adjusting the car speed to a selected upper limit, the ACC provides automatic collision detection and security distance adjustment. More succinctly, the ACC adjusts the vehicle's speed to the changing flow of the surrounding traffic. It uses a three-beam radar sensor (see figure 2.3) to monitor the road in order to detect if an object moves ahead of the vehicle.

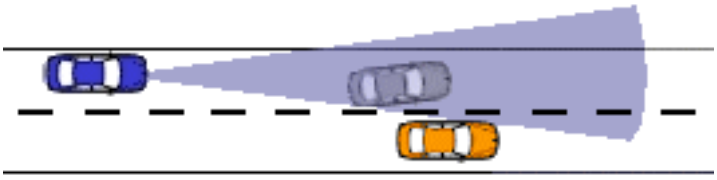


Figure 2.3: The ACC radar sensor

When the radar eye of the ACC detects an object ahead of the vehicle, the sensors pick up the information concerning the location (e.g. the relative distance), the relative speed and other attributes describing the movement (e.g. angle) of the moving obstacle. Based on this information, the ACC's electronics determine if the object moves, and if so, the ACC will check if the object moves in the vehicle's lane, then it immediately decides whether the car's speed needs to be adjusted or not. Depending on the situation, the ACC autonomously initiates the slowing down or the acceleration of the car. This is done through the activation of special engine management and brakes activation functions. Moreover, when driving downhill, the ACC can automatically adjust the car's speed to the pitch gradient. The ACC is also able to recognize when the vehicle enters a curve. In this case, the ACC is able to adjust the speed of the vehicle to the accuracy of the coming bend. If a car is moving ahead of the vehicle in the curve, the ACC will additionally take care to keep a minimum (security) distance between the vehicle and the preceding car.

Further advanced features of the ACC include the driver support and the comfort. For example, the speed adaption when driving downhill or in curves is not only a security remedial action, but it is also a passengers comfort feature. In every critical phase of the traffic, the corresponding information (current status of the traffic, driving instructions, etc.) is displayed to support the driver. For example, when a moving obstacle is detected in the vehicle's lane, the ACC will autonomously display advising information at the same time as it starts the procedure that adjusts the speed of the vehicle to the actual situation of the traffic. As soon as the obstacle is out of the lane, the ACC releases and the vehicle's speed goes back to the control of the driver, with respect to the chosen maximal speed if any has been set. Thereby, the setting of the ACC is made comfortable for the driver as the ACC settings (selection of maximal speed, minimal security distance) can be done through all kinds of switches, steering arms or buttons from the dashboard. Figure 2.4 shows the electronic control unit running the main of the functionality of the ACC.

Figure 2.4: The ACC device

2.1.4 The ACC functional connectivity

The above description of the functioning of the ACC shows that the ACC needs to run functions that are located on different devices. Moreover, in addition to the data supplied by its sensors, the so-called ACC device needs information from other parts of the system. For example, the ACC combines the data supplied by its sensors (e.g. relative distance of the obstacle, relative speed, etc.) with the car's current course attributes (e.g. wheel revolutions, vehicle pitch, centrifugal forces acting on the vehicle, etc.) to calculate the tightness of the approaching curve and to adjust the car's speed to maintain the necessary security distance. In fact, the ACC collects data from the accelerator pedal sensor, from the brake pedal sensor, from the setting buttons of the instrumentation and the steering block, etc. The ACC also needs to activate some functions that are located on different ECUs, e.g. braking functions, accelerators, lighting signals senders, etc. This necessitates the communication with the other devices.

Figure 2.5 illustrates the level of interconnection needs for the functioning of the ACC. The ACC is linked to the braking system, the stability control, the lighting system, the engine management, the dashboard, the steering control and the gears control, so that it can access data that is necessary for its computations and can automatically activate the brakes, adjust the gear, display information, instruct the fuel injection and the lighting announcements, etc. The functioning of such a distributed application is highly dependent on the communication, the coordination and the synchronization of different functions across the whole system and the real-time constraints that apply to the various functions performing the distributed functionality are determinant for its successful working, and thus for its acceptance. If the system does not react on time, it is useless and dangerous. If the system is informed on a critical situation too late or if the system's instructions are received too late, the vehicle will probably not react as expected. However, in addition to the need of predictable behavior, automotive electronic-actuated features underly stringent costs constraints that depend on both their resources and the design time.

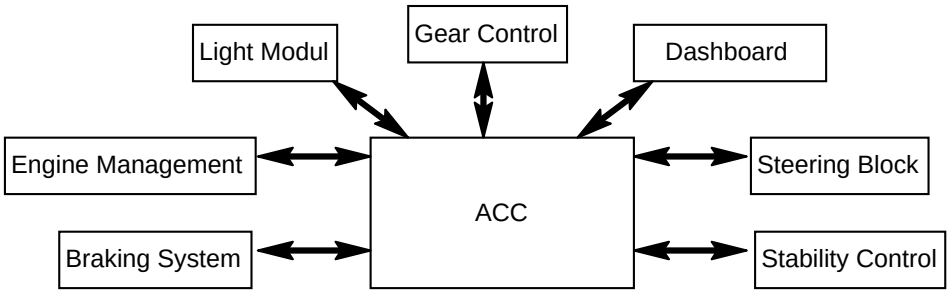


Figure 2.5: ACC interconnection

2.2 Automotive connectivity

2.2.1 Automotive communication protocols

Several network technologies have been defined to cope with the particular requirements of AES. The transmission rate, the bandwidth, the real-time performance, the immunity against noise, the scalability, and the implementation costs are the most decisive attributes of these solutions. As the different networks are based on different technologies, they also present different interfaces, i.e. the word length, the header format, the physical medium, the bit rate, etc. according to the networking solution. Gateways are used to interconnect the networks (see figure 1.1). Following, we survey the most popular communication protocols that are used to interconnect the automotive devices. The survey includes automotive domain-specific communication protocols, general-purpose protocols that have been implemented in vehicles or are identified as competitors for some automotive-specific protocols and the communication protocols that appear as trends for

future generations of vehicles. In order to be brief, we only pick out the principal characteristics of these network technologies that can help us to understand their applications. For example, the medium access strategy and the arbitration method are important performance attributes that address the throughput, the transmission delays and the determinism of the network behavior. The physical medium has consequences on the weight, the shielding, the robustness, the price, etc. of the given network technology. The bandwidth and the triggering control of the network are also important indicators of its performance. The triggering control (event, time) indicates the ability to react to asynchronous or synchronous events. Further important attributes, that are also useful for our understanding of the automotive networking include the size of the networks, the art of addressing, the handling of priorities and the jitter. The target application fields, the quality of the existing implementations and the future (evolution) of the surveyed technologies will also improve our understanding of the criteria that guide the choice of the automotive designers.

2.2.2 All-rounder automotive communication protocols

Originally designed for automotive systems, some communication protocols have been adopted in various domains such as in industrial production installations, home connectivity, etc. This is the case with the CAN (Control Area Network) and the VAN (Vehicle Area Network). We also consider the communication protocols such as the SAE J1567 and J1850 protocols, that originally proprietary, where adopted automobile industry-wide.

The **CAN (Control Area Network)** [1, 2] is the most popular automotive communication protocol. CAN is used to network "intelligent" devices (ECUs, sensors and actuators) within a system. The CAN protocol was defined at Robert Bosch GmbH to enable automobile in-vehicle embedded communication. CAN has found wide application in the industrial automation, the aircraft and aerospace, the building automation (e.g. lift and escalators), etc. In automotive systems, the CAN is generally implemented on a bus topology. CAN is an asynchronous, serial transmission, multi-master and broadcasting communication system, using a CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) media access method. That means that all CAN nodes are able to transmit data and several CAN nodes can request the bus simultaneously. In CAN networks, data integrity[32] is assured by sophisticated error detection mechanisms and re-transmission of faulty frames. CAN is defined in the ISO standard ISO 11898-1. The protocol supports single POF (Plastic Optical Fiber) or twisted pair cupric cables at the physical layer. A single cupric wire is allowed when the transfer rate is lower than 100 kbit/s. The bit rate depends on the bus length: short bus, high rate. The maximum specified transmission rate is 1 Mbit/s (in the high-speed version). This value applies to networks up to 40 m. For longer distances, the data rate is reduced, for example for distances up to 500m a speed of 125 kbit/s is possible, for transmissions over more than 1km the reasonable data rate is around 50 kbit/s. Theoretically, a CAN bus can support a non-limited number of nodes. But in the practice, an acceptable QoS is obtained by an average of 64 nodes.

CAN is an event-triggered bus concept, by which the frames are identified rather than the nodes. The identifier of a frame determines its content and also the priority that the frame enjoys in competition for the bus access. Therefore, every frame identifier is unique within the whole system. The priorities of the frames are laid down off-line during the system design and cannot be changed dynamically. This concept allows the modular design of CAN networks, that facilitates the network extendability. CAN networks offer high transmission reliability, but no deterministic frame latency. In the automobile industry, the high-speed CAN is usually implemented in vehicle's chassis and power train applications, on shielded wires in linear bus architectures. The main application domain of the single-wire CAN-bus is in the comfort electronic. As the CAN does not offer the degree of real-time deterministic behavior that is necessary for safety-critical applications, it has been enhanced to build a time-triggered variant, the TTCAN. However, the great success of CAN (i.e. CAN is implemented in most of the European cars and other high-volume markets like industrial

production controls, domestic and medical appliances, etc.) led to a low-cost implementation of CAN devices and therefore guarantees the presence of the CAN in the future. CAN still has good days in the automotive engineering.

The **SAE J1567** or **C^2d (Chrysler Collision Detection)** protocol was defined by Chrysler as a sensor network protocol with the goal to minimize the software overhead needed for the integration of network nodes [79]. The C^2d protocol prescribes a bus topology. The transmission medium is a twisted-pair of cables on which serial SCI-interfaces can be connected. The SAE J1567 protocol supports a pulse width signal modulation and a multi-master media access (CSMA/CR) with a bit-by-bit arbitration. The theoretical transmission rate is around 7.812 kbit/s. J1567 has been implemented by Chrysler for diagnosis and data sharing in the vehicle body, the chassis and the power train, for example in the "'Grand Cherokee"'.

The **SAE J1850** was seen in the last decade as the emerging standard for U.S. vehicles. J1850 is designed as a combination of SAE J1567 with other proprietary protocols like the Ford's HBCC (Hosted Bus Controller Chip) and the General Motor's DLCS. It supports two multiplexing methods: a pulse width signal modulation (PWM) with a rate of 41.6 kbit/s and a variable pulse width signal modulation (VPWM) with 10.4 kbit/s. The transmission medium is a two-wire bus for the PWM version and a single-wire bus for the VPWM version. J1850 is a CSMA/CR protocol, that means that there is no unique master on the bus. DaimlerChrysler, General Motors and Ford have implemented the J1850 for diagnosis and power train services including engine management, transmission as well as the functions of the instrumentation and the ABS connections. But today, the SAE J1850 has been replaced by the CAN protocol.

2.2.3 High-speed and real-time protocols

To interconnect multimedia and safety-critical applications, the automotive engineering has designed high-speed and real-time communication protocols. Although the MOST (Media Oriented System Transport) protocol is actually the most famous protocol for multimedia applications, D2B (Domestic Data Bus), GigaStar, MML (Mobile Media Link) and other well-known general-purpose communication protocols like FireWire/IEEE 1394 and USB are also serious candidates for multimedia applications. Moreover, a time-triggered version of the CAN, the TTCAN (Time-Triggered on CAN), is competing with the other real-time protocols including TTP, FlexRay and Byteflight in the area of safety-critical applications.

A - Multimedia protocols

Most modern vehicles offer entertainment for the passengers (TV, movies, etc.) as well as vocal and visual assistance for the driver (navigation, representation of the images from video cameras). These applications need a broadband communication with real-time performance. We present only the MOST protocol because of its extremly dominant position in the list of the automotive multimedia-capable protocols.

The **MOST (Media Oriented System Transport)** protocol specification [11] is a standard of the MOST Cooperation, that actually comprises BMW, DaimlerChrysler, Harman/Becker and OASIS Silicon Systems. With a bandwidth of 25Mbit/s, the MOST can connect up to 64 nodes in every possible topology, using a POF (Plastic Optical Fiber) at the physical layer. This is a considerable advantage in terms of electromagnetic compatibility, cost and weight. It is therefore not surprising to find MOST implementations in practically every actual car in which multimedia data must be transported. MOST is a peer-to-peer network protocol that can be configured as single or multi-master network. The MOST specification defines a multiplexing method that clearly distinguishes the time slots for (synchronous) data transfer from those that can be used for

asynchronous frames. With its 25Mbit/s, today's MOST is apt to supply most multimedia services at a good quality level. Since the evolution in the multimedia application domain is related with the magnitude of the data to be handled and eventually to be transported, other high-speed protocols with larger bandwidths like FireWire (400Mbit/s, 800Mbit/s and probably up to 3.2 Gbit/s) and USB 2.0 are competing to replace the MOST in the cars of the future.

B - Safety-critical application-compliant protocols

Real-time and deterministic behavior are particularly important for safety-critical applications. TTP, Byteflight, FlexRay and TTCAN are automotive domain-specific protocols with these properties.

The **TTP (Time-Triggered Protocol)** is known in the avionics and the railway (www.ttagroup.org). TTP is designed for stringent fault tolerant real-time distributed systems. It is said to ensure that there is no point of failure in the system. TTP is a TDMA protocol where the arbitration is done once at the initialization time. TTP can be found in two versions: The low-cost version TTP/A that is classified following the SAE (Society of Automotive Engineers) taxonomy as a class A protocol, is a master-slave UART-based protocol, used in the automobile industry for sub-networking, i.e. branching several sensors (cluster) to a single ECU that itself is related to a backbone bus. TTP/A applications are known in engine control and car body applications, as competitor of the LIN and the low-speed CAN protocols. The higher performance version, designed for safety-critical and fault tolerant hard real-time applications is the standardized TTP/C[107]. TTP/C offers a bandwidth of 25Mbit/s on a bus topology. Because of its hard real-time capacities, the TTP/C is proposed in the automobile industry as a competitor of the TTCAN, and moreover of the FlexRay, particularly for x-by-wire applications.

The **TTCAN (Time-Triggered communication on CAN)** is an extension of the CAN with the time-triggered paradigm. The TTCAN[19] is defined as a higher layer protocol on top of the standard CAN protocol's physical layer. The goal of the TTCAN is to provide the determinism needed in the design of hard real-time systems. This is achieved by avoiding the latency jitters for the media access and ensuring deterministic communication patterns. The TTCAN provides the mechanisms to schedule the frames in a time-triggered way as well as in an event-triggered way. The TTCAN bus arbitration relies on a special, hybrid TDMA method, where every node accesses the medium at predefined time slots, depending on the priority of the frame it is sending (see ISO 11898-4). In order to support time-triggering and to handle asynchronous frames conveniently, the TTCAN protocol allows the definition of tree different sorts of time windows: The preassigned time slots, called exclusive time windows, support the time-sharing medium access. The arbitrating time windows support the handling of asynchronous frames. A node must compete for an arbitrating time window in the same way as in a CAN network to send asynchronous frames. The third class of time windows, the free-time windows, serves for eventual extensions of the network. A free-time window can be converted in exclusive windows for new nodes or in arbitrating windows to extend the bandwidth. TTCAN is implemented in the power train, i.e. for engine management, transmission, chassis control applications, and in some security- and safety-oriented car body applications that need deterministic communication behavior. Since TTCAN is well adapted for x-by-wire applications, its future is considered very hopeful. Indeed, the implementation costs are much lower than those of its competitor FlexRay.

The **FlexRay** protocol is defined by the FlexRay Consortium (BMW, Bosch, DaimlerChrysler, Freescale, General Motors, Philips and Volkswagen) to be a protocol combining the attributes of the Byteflight and the TTP protocols, including fault tolerance, deterministic behavior, collision-free transmission, guaranty frame latency, arbitration-free medium access, flexibility in both bandwidth

and system extension, functional scalability and on-board diagnosis, support of optical and electrical physical layers. Moreover, the FlexRay Consortium (www.flexray.com) is guided in its initiative by the need to simplify the design and manufacturing of vehicles, master the cost of the automotive micro controllers and other electronic devices while simplifying the system scalability. FlexRay intends to be a low-cost communication system, not only because of low component and implementation costs, but also because of its ability to provide system scalability and its energy-efficient behavior. For example, FlexRay supports a wake-up/sleep functionality (via bus) that addresses the power management needs. The FlexRay specification targets on power train, chassis and body control applications. The FlexRay protocol provides a combination of static and dynamic frame transmission, incorporating the advantages of familiar synchronous and asynchronous protocols. FlexRay can be configured as asynchronous (event-triggered), synchronous (time-triggered) or a mix of both. The communication medium consists of a single POF or two cables that act as two buses, each offering a bandwidth of 10Mbit/s. A FlexRay network can connect up to 64 nodes. FlexRay is designed to support bus, star and cluster star topologies in single as well as in dual channel modes. Some FlexRay implementations have been tested in Mercedes and BMW cars. The first commercial productions with FlexRay were expected by the end of the year 2006.

Byteflight (www.byteflight.com) is a FTDMA (Flexible Time Division Media Access) in-car electronic communication protocol defined by a consortium around BMW to provide a means for fault tolerant communication with deterministic behavior for safety-critical systems. The flexible bandwidth management of Byteflight enables easy on-board diagnosis. Furthermore, Byteflight is defined to be collision-free and low-cost. Similar to the CAN, Byteflight uses a frame-oriented addressing via the awarding of identifiers. Hard real-time signals (alarm, etc.) can be prioritized. The physical layer is generally implemented on a star topology of POF (Plastic Optical Fiber) supporting half-duplex communication by up to 10Mbit/s. But bus and cluster topologies are possible. Since 2001, Byteflight is implemented in BMW's cars for passive (e.g. air bags triggering and crash sensing) and active safety functions (e.g. belt tensioner), and for the body applications.

2.2.4 General-purpose protocols

Because of its severe cost constraints, the automobile industry is constantly looking for cost-efficient communication facilities. Hence, asides from the automotive systems specific communication protocols, the automobile industry has adopted a full range of general-purpose communication standards that are common for example in the consumer and building automation. USB, Bluetooth, SPI and I^2C are such cost-efficient general-purpose communication protocols used to connect automotive devices.

USB: With a bandwidth of 480Mbit/s and its plug-and-play capacity, USB 2.0 (Universal Serial Bus)[15] is well-adapted for audio and video data transfer between the car and external peripheral devices. USB 1.1/2.0 [14] is currently used to connect portable devices, e.g. navigation systems, external memories, flash card readers, video cameras, MP3 players or CD-writers, etc. to the vehicle. From the 4-wire communication medium of the first versions, the USB has evolved to offer wireless communication possibilities. Some car makers are actually trying to connect vehicle's embedded multimedia devices with each other through USB networks. Profiting in addition from its broad bandwidth, the USB is supposed to gain more place in the automotive engineering in the near future.

Bluetooth: Bluetooth (www.bluetooth.com) is an open specification for short-range (10-100 meters) wireless networking at low cost providing instantaneous connections between Bluetooth-enabled devices. Potential usage of Bluetooth in the automotive domain includes the wireless connection of mobile phones, portable DVD/CD drives or diagnostic equipments. Most automotive brands worldwide offer optional or standard Bluetooth-enabled communication systems in their

new models. Applications that enable hand-free usage in cars and synchronized operation of mobile phones with the car's audio system are becoming popular. Unfortunately, although Bluetooth is a high-secured network (using authorization, authentication, FFH (Fast Frequency Hopping) mechanisms), we are yet not aware of commercial Bluetooth applications enabling vehicular external connections, such as remote car handling. This is due to some deficiencies of the security in the implementations of Bluetooth connections. In fact, the Bluetooth special interest group faces implementation weaknesses that can cause unsecured Bluetooth communications, such as car whispering[2], cabir[3], bluesnarfing[4], etc.

SPI: The SPI (Serial Peripheral Interface), a 4-wire serial communication interface that is often used by microprocessors to interface with peripheral smart chips in industrial and building installations is a simple-to-implement synchronous serial protocol for connecting devices at low-speed (up to 5 Mbit/s). SPI connections follow a master/slave principle. Connected peripherals can be processors, converters (A/D and D/A), memories (EEPROM and flash), real-time clocks, sensors (temperature, pressure) or others (signal mixer, potentiometer, LCD controller, UART, CAN controller, USB controller, amplifier). When connecting two processors or two systems, the SPI can be configured as a single-master-multi-slaves as well as a multi-masters-multi-slaves network with two control and two data lines. Since SPI lacks a standard specification, its implementations are different from one chip to another, from one developer to another. SPI is used in the automobile industry as a sub-bus (for smart networks). For example, SPI has been used to connect sensor's microchips in car's instrumentations.

I^2C: The I^2C (Inter-IC bus) is like SPI also a control bus that provides communication links between integrated circuits. I2C is a 2-wire serial bus standard (www.philips-semiconductors.com), supporting multi-master configuration, frame-priority and bi-directional communication from 100kbit/s in standard mode to 3,4 Mbit/s in high-speed mode. Only two lines (clock and data) are required for full duplexed communication between multiple devices. The I2C protocol specifies a range of 127 addresses. Each IC on the bus has a unique address. The I2C bus protocol allows collision detection, clock synchronization and hand-shaking features for multi-master systems. This simple bus concept is widely used for system control and data transfer from e.g. temperature sensors, voltage level translators, general-purpose I/O, A/D and D/A converters, CODECs, and microprocessors of all kinds. The automotive engineering uses I^2C protocol for applications similar to those of the SPI application areas [7], i.e. for smart on-board networks.

2.2.5 Other in-vehicle and smart network protocols

LIN, ISO9141, K-Line, BSD, RS-232, RS-485, SAE J2058 protocols are implemented in secondary in-vehicle networks (such as sensor networks subsidiary to a backbone net). The LIN protocol is (at least in Europe) almost the standard in this class, even if its specification itself is not yet standardized.

LIN: The LIN (Local Interconnect Network) is defined to be a cost-saving (compared with CAN) protocol that can connect (intelligent) sensors and actuators through a single-wire medium [9]. LIN communication is based on UART/SCI interfaces, very common and thus easily available at very low price. LIN is particularly useful where the CAN is too powerful and too expensive, i.e. LIN

[2]The car whisperer is a software tool developed to enable the connection with the Bluetooth kit of a car in order to send and receive audio and video data. Potentially, an individual using the tool can be remotely connected to a car and communicate with it from an unauthorized device, for example to send false audio information to the driver over the speakers of the car or the illegal visitor can spy the cars occupants by retrieving information from the car on its remote device

[3]The Cabir is a malicious software that when installed on a phone, uses Bluetooth technology to send itself to other similarly vulnerable devices. Due to this self-replicating behavior, it is classified as a worm.

[4]Bluesnarfing allows hackers to gain access to data stored on a Bluetooth-enabled device, e.g. phone, using Bluetooth wireless technology without alerting the user

is considered as a "junior CAN". It provides a 20kbit/s bandwidth for low-end applications. LIN's main application field is in the car body, i.e. for applications like doors and sunroofs, servo steering, seats, climate regulation, lighting, mirrors, wipers control and similar applications. LIN networks are implemented in the vehicles of Smart, Mercedes, BMW, VW and a lot of other constructors. The size of a LIN network is typically under 12 nodes (actually 1..10). LIN guarantees a maximal frame latency. The LIN medium access is organized on a single-master arbitration. Each frame has a unique identifier. The LIN master is charged to pilot the slaves (generally sensors and actuators), and at the same time it is the communication interface between the LIN (cluster) network and the backbone network, e.g. CAN. Nodes can be added to the LIN network without requiring changes in other slave nodes. Only the LIN master must be reconfigured when changes occur in the network configuration. Recently, the LIN Consortium has published the version 2.0 [10] of the LIN that includes real-time capabilities, standard driver for all LIN-component hardware and the certification of all LIN-components. This is an initiative to sustain the proliferation of off-the-shelf LIN slaves. When attached to a CAN backbone, the LIN network can access advanced system diagnostics services. As LIN has gained a wide acceptance, the LIN Consortium has good reasons to project a positive future. The standardization of the functionalities of common LIN-slaves will accelerate the vehicle design process, reduce the prices of the LIN-components and enable more viable LIN-networks.

2.3 Conclusion

A premium-class vehicle today contains a considerable number of ECUs and some intelligent actuators and sensors (cf. figure 2.2) that need efficient communication facilities to provide the required functionalities. Currently, most of the communication protocols used in the automobile industry are either standardized or in a standardization process. Standardized communication protocols afford modularity, distributed development, portability, external integration, cost reduction and achieve easy gateway functionality. CAN is the most used network technology in the automotive engineering. A vehicle generally contains several CAN networks, each of them tailored for a particular application domain (see example 1.1. Running at less than 125kbit/s, the low-speed CAN is usually implemented in the car body to network the devices that are concerned with the realization of personal comfort, e.g. seats, doors, sunroofs and windows control. The air conditioning, the internal lighting system, etc. are usually also controlled through a low-speed CAN network. A higher speed CAN, that can runs theoretically 1Mbit/s, but generally operates at about 500kbit/s is used for more safety-critical functions like engine and gearbox control, brake and cruise management, etc.

MOST, USB and Bluetooth are promising solutions in the networking of automotive multimedia and telematics solutions, whereas FlexRay is observed as the future standard for hard real-time and broadband applications and LIN for less constrained communications. As automotive-specific technology, LIN might remain for a long time the standard in the field of sensors and smart networks. TTP yields very good performance, but provides very limited means to treat asynchroneous and even-triggered frames. In the same class with TTP and FlexRay, TTCAN is featured to manage time- and event-triggered frames, but its bandwidth is right limited. A pertinent observation shows that TTCAN and TTP do either not seem to be well-prepared for the competition against FlexRay.

The panoramic view on the automotive communication technologies shown in table 2.6 refers to the most implemented protocols in the automobile industry from the OEM point of view. The information (e.g. topology, implementation) given in this table is based on the implementations of the protocols with regard to the automotive domain. The medium load represents the maximal effective throughput, given as a percentage of the theoretical bandwidth. The maximum number of nodes is the largest size with which the network is able to work with guaranty QoS. The cost is coded as follows: 1 = very low cost, i.e. 0.5-2€; 2 = low cost, i.e. 1-3€; 3 = high cost, i.e. 3-4€; 4 = very high cost, i.e. $>$ 4€.

Although these automotive networking technologies are powerful, they are naturally limited.

Properties	CAN	TTCAN	LIN	Byteflight	MOST	TTP/C	FlexRay
Bandwidth in bit/s	33,3k-500k …1M	33,3k-500k …1M	20k	10M	25M	25M	10M
Triggering	Event	Time + Event	Event	Event/Time	Event	Time	Time
Medium (Length in m)	Twisted-Pair /Single wire (40-1000)	Twisted-Pair /Single wire (40-1000)	Single Wire (40)	POF	POF	Twin axe Cable	POF/ Dual Cable
Latency	Variable	Deterministic	Deterministic	Deterministic	Variable	Deterministic	Deterministic
Topology	Bus	Bus	Bus	Bus, Star	Ring, Star	Bus	Bus (12), Star(24)
Medium Load in %	50	50	???	60	15	80	90
Max Nbr of Nodes	32 (single wire) 110 Pair	32/110	12	22	64	64	64
Main Application Domain	PT, Body	PT X-by-Wire	Smart Subsystems Cluster NW Comfort	Body Passive +Active Security	Multimedia	PT X-by-Wire	PT X-by-Wire
Cost (1,2,3,4)	2-3	3	1	2	4	3	???

Figure 2.6: Some automotive in-vehicle communication technologies

For illustration, the CAN, specified to run 1Mbit/s can only achieve a maximum of 500kbit/s in automotive environments, due to the electromagnetic radiation on the twisted pair cables (shielding is too expensive). The ideal communication technology for the automotive industry does actually not exist. Progress in the automotive E/E-related technologies i.e. including electronics, embedded systems design, communication and telematic will enable the integration of more functions into the vehicles. For instance, the next wave of new features in automobiles will include parking aid, electronic toll collection, collision warning, personal digital assistance, driver alertness monitoring, intelligent brakes, etc. and extend existing functions to more complicated functionalities, for example the new ACC (Adaptive Cruise Control) that is announced by the main European and Asian brands will enhance the actual version with preemptive collision detection. Thus, the automotive engineering must look for methods to manage the system complexity. In our mind, we will be able to solve this problem if we can control the growth of the number of the device and optimize the usage of the capacities of the buses and the processors. It is thus well-advised to start today to look for the solutions before the knock-out arises.

Chapter 3

Embedded systems design

The design of embedded systems is concerned with the development of the software and the hardware. The software is the program code of the system functionality. The hardware is the material part of the system. As embedded systems constraints include the performance, the cost and the time-to-market, only proved and efficient design techniques are adopted. In this chapter, we give a survey of the embedded systems design methods that will help us to identify the similarities with the design of AES. Since the sequential design process does not allow the trade-offs and the efficiency that are so important in the embedded systems design, we particularly focus on the concurrent design method.

3.1 Embedded systems design methods

3.1.1 The sequential design process

The most important activities in the embedded systems design include the specification of the system functionality, the conception of the hardware platform, the analysis and the implementation of the software and the hardware, and the integration of the software and the hardware. The traditional embedded systems design process makes use of a sequential design flow as shown in figure 3.1. Based on the specification of the system-desired functionality, the architecture of the hardware platform is decided. This activity is called allocation. Then the elements of the functional specification are assigned to the elements of the hardware platform. This is called binding or mapping. Afterward the application code is generated and loaded on the target hardware platform. This operation is the deployment.

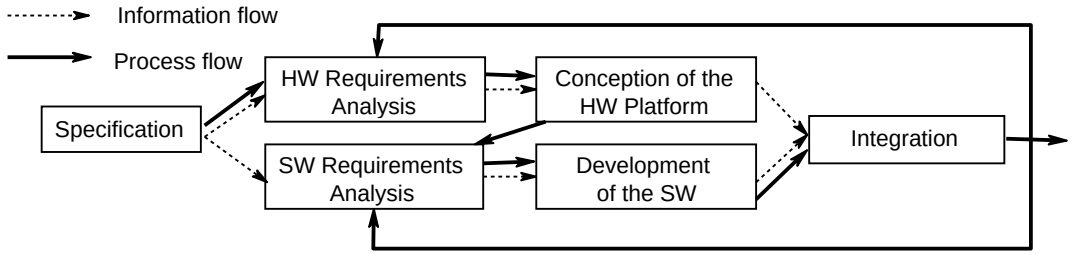


Figure 3.1: Sequential embedded systems design method

The sequential development process can be roughly divided into two phases: The design of the functionality, the design of the hardware platform and the implementation. The implementation is concerned with the mapping, the development and the targeting of the code. The design of the hardware platform is guided by the type and the characteristics of the application. Based on the characteristics of the application, the designer will decide which sorts of processors, memories, buses, interfaces, etc., must be used in the hardware platform. Depending on the calibration of the chosen components (e.g. clock frequency, memory architecture, levels of caches, communication

protocols, synchronization techniques, number of I/O pins, etc.), the designer will decide which number of each type of component is needed.

Typically, the allocation is not an automatic process. It is influenced by several unpredictable factors like the availability of the components, the quality of the development environment of the existing components, e.g. the quality of the coding tools (compilers, debuggers, simulators, etc.), as well as the preferences and the experience of the designer. In the implementation phase, the designer must try to take the best advantage of the processing power and the communication bandwidth that is available on the platform, e.g. the tasks and the data of the application software must be distributed optimally on the different components of the platform. This is achieved by the means of the partitioning. The functional specification of the system has to be captured in a platform-compatible language. Depending on the hardware components on which the function is mapped, the specification languages vary from "high" level (e.g. Java, Pascal, Fortran, C) to "low" level languages (assembler, bytecodes, ...). Note that the notion of "high" and "low" level languages is relative and absolutely contextual. With high level languages, i.e. as defined above, an OS or a middleware is generally needed to link the application software on the hardware platform. The integration is concerned on the one hand with the deployment of the code on the hardware platform and on the other hand with the interconnection of the different components of the platform.

Following this development method, the platform is defined a priori, hopefully by "gurus", based on expert evaluation of the functionality and the constraints of the system. The implementation of the system software is constrained by the chosen platform. The system partitioning is done after the allocation, whereas the designer supposes that the system will be successfully implemented on the available processors that are generally commercial-available processors. It is thus difficult to find the optimal tailoring for the hardware. Specific hardware is joined only when timing or space constraints cannot be met. This is economically sensible, but in the embedded systems development, minimizing the price of the system is not the unique goal. The size, the weight and the power consumption of the system are often more important business criteria, for example in the ubiquitous computing domain. In fact, a higher-sized hardware means more material, larger volume, supplementary weight and power consumption than needed, and thus higher costs. Furthermore, with this method, design errors such as the non-observation of the performance constraints are detected after the integration. This is obviously very late and changes in the design may necessitate to reanalyze the specification. In the extreme case, the specification must be revised. The sequential design method can be a time-consuming, hazardous and thus an inefficient process with consequences on the time to market and the product cost. Various alternative development flows have been proposed. Mostly differing only in the details. One of the most popular design methods proposes the concurrent design of the hardware and the software. This method is called HW/SW co-design.

3.1.2 The concurrent design method

The concurrent design of the different parts of a system is called co-design. In a co-design scheme, the functional specification is partitioned before the implementation begins (figure 3.2). The co-designer supposes at the beginning that some parts of the system should be implemented in hardware to guarantee high performance and the rest in the software that will run on general-purpose processors, thus at low-cost. After the partitioning, the software and the hardware parts of the system are developed concurrently in observation of the communication interfaces between them. Then, the hardware and the software are integrated and the whole system is validated with regard to its performance and its functionality requirements.

Compared with the sequential method, this design method allows to find better trade-offs between performance and cost. The co-design method allows to early detect incompatibilities between the hardware and the software. The communication interfaces between the software and the hardware can be easily optimized when these are designed cooperatively. The design

process is not exposed to so many loops as the traditional process flow. Thus, the design time is shorter. As seen in figure 3.2, the most important activities in the co-design method include the functional specification, the partitioning and the implementation, i.e. the system synthesis and the integration.

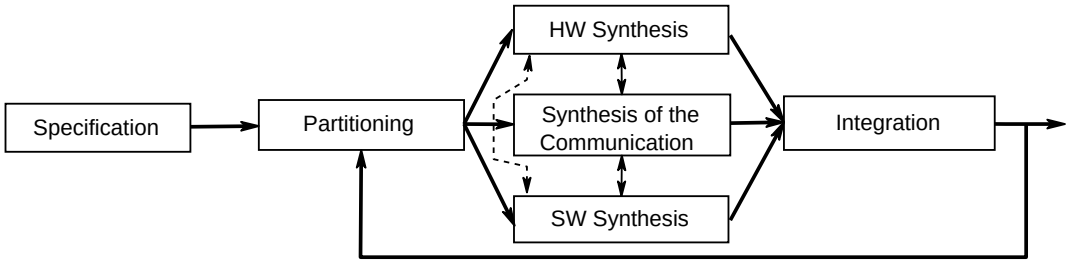


Figure 3.2: Concurrent design method of embedded systems

3.2 Design activities

3.2.1 The specification

The specification is essential in the co-design method. In the specification phase, the system functionality must be described in a way that its structure and its behavior are clearly and unambiguously identifiable. The entire system must be specified as a whole in order to enable an efficacious design space exploration. Thereto, the specification must provide the details that are necessary to make advantageous decisions during the partitioning. The actual trend favors a specification that is free from any constraint on the implementation, e.g. a platform independent model (PIM). However, capturing a specification is a difficult matter, because today's embedded systems are very complex and heterogeneous, and the specification techniques are not always sufficiently featured. For illustration, although time information is very important in the embedded systems specification, it is difficult to be described with precision.

A plethora of languages have been proposed to specify embedded systems in the scope of the HW/SW co-design method (see figure 3.3), but there is no ideal language to uniformly describe the software and the hardware parts of a system. The hardware is generally described by means of HDLs (e.g. *VHDL, Verilog, HandelC, SystemC, SystemVerilog*) whereas the software is specified with traditional software languages like *C, C++, Java, Assembler*, etc. At the system level, beside of general-purpose languages like SDL or MSC, numerous co-design-specific languages are established, for example *HardwareC*, C^x, *SpecCharts* or *StateCharts*, that are used in VULCAN[63], COSYMA[67], SpecSyn[61, 117] and CODES [30].

3.2.2 The partitioning

The partitioning is the most original activity in a co-design method. The goal of the partitioning is to share the computation work among the processing components of the platform and the data among the storage places. In the embedded systems design, the partitioning is done by splitting the system specification into different parts according to the given design, performance and cost constraints and distribute these parts on the resources of the platform so that the system functioning is optimized and the required performance is met within the overall system requirements and constraints, by concurrently avoiding resource underutilization. Each distribution is called a partition. The constraints are generally given on the size, the weight, the power consumption, the cost, the latency, the material, the speeding up of the system, etc. A partition is achieved by analyzing the structure and the behavior of the elements of the specification in order to identify those which should be advantageously implemented together, those that can run on a general-purpose processor and those that must be implemented on specific or as dedicated hardware.

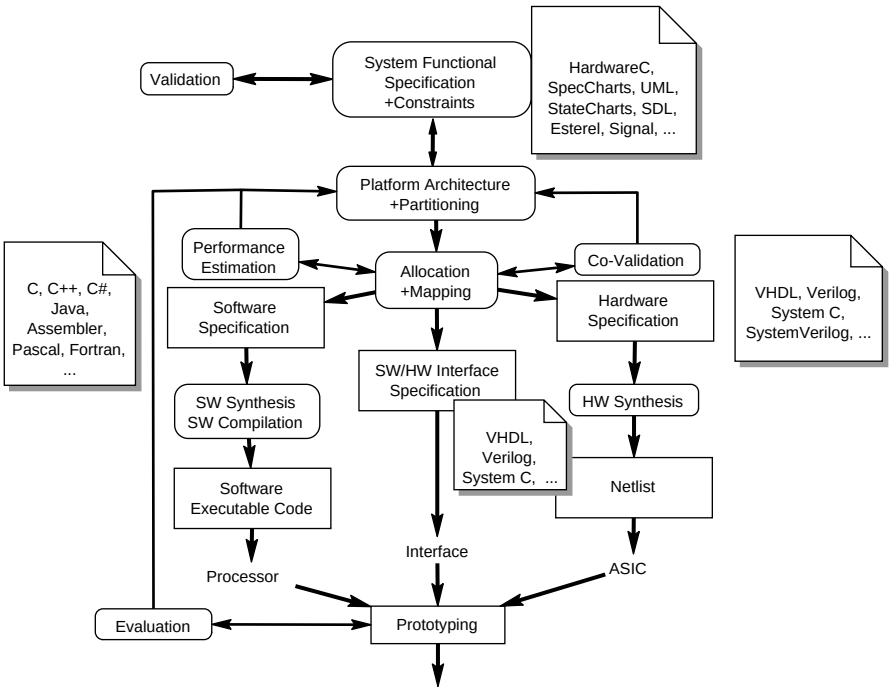


Figure 3.3: Specification languages in the HW/SW co-design method

The partitioning process itself depends on its objectives. In some approaches, the partitioning aims at the definition of the parts of the system that will be implemented as hardware, i.e. custom component, and the parts that can be implemented as software to run on general-purpose processors. This is called HW/SW partitioning. The HW/SW partitioning is generally a functional partitioning, i.e. the functions of the system are grouped into consistent modules that are each assigned to a platform component that is implemented either in hardware or in software. In another approach of the partitioning, the system specification is partitioned among several custom components, e.g. among several ASICs. This is a hardware partitioning. The hardware partitioning produces an all-hardware solution since it does not consider any software component. Generally, in the hardware partitioning, the system is first implemented with (fine-grained) structural devices (e.g. gates), that are then partitioned among several custom components. This technique is called structural partitioning. The all-hardware implementation has the potential to provide high-performance (via hardware speeds and parallel execution of operations) and reduced space occupation, but is expensive in the fabrication of ASICs. The all-software implementation, even when running on high-performance (general-purpose or specialized) processors is in principle the low-cost solution, providing flexibility and upgrading, but it does not guarantee the achievement of the performance constraints and can incur high costs in developing and maintaining complex software. The trade-offs between quality, performance and cost are achieved through mixed HW/SW architectures, i.e. architectures containing general-purpose processors as well as custom processors.

3.3 The implementation

3.3.1 The software synthesis

The software synthesis is concerned with the implementation of the software part of the system, i.e. the embedded software, that will run on programmable hardware units and the configuration of the corresponding OS. This includes the translation of complex software descriptions (e.g. concurrency, timing behaviors or task scheduling) into a software language that can be compiled by common compilers, the coding, the targeting of the software and the optimization of the code, e.g. reduction of the amount of busy-waiting of the processor. This can be done automatically

or manually. Because of its particular requirements, the embedded software development has evolved to a special engineering field. In fact, the embedded software development is significantly constrained, not only because of the tight requirement on the code size, but also because of the limitations concerning the implementation cost, the performance and the functional requirements of embedded systems (dependability, maintainability, timeliness, concurrency, reliability, reactivity, real-time communication with the real world, etc.). For illustration, OS for embedded systems are generally required to behave in real-time and concurrently, because of the shortage of space, embedded systems OS are tailored to provide only the minimum obligatory services. The principal activities of the embedded software development are summarized in figure 3.4.

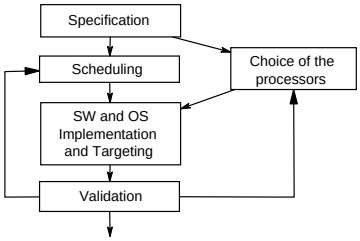


Figure 3.4: Design of the embedded software

Potential targets of the software include GPPs, DSP, MCUs and ASIPs. A GPP is designed independently from any particular application, so that it can be used for a large set of application families. Some examples are RISC, CISC, Sparc, Pentium processors. Nevertheless, some GPPs are optimized and thus specialized on a particular application domain (e.g. video, audio, mobile telecommunication, etc.) but not on a specific application. Examples of domain-specific processors include network processors, multimedia processors, Digital Signaling Processors (DSP), etc. DSPs are processors that are specialized on digital signal processing operations. Their architecture and their instruction sets are optimized for arithmetic operations such as Multiple-Accumulate operations (MAC), pipelining, direct memory access, address coding and decoding, ... that can advantageously accelerate the operations that are usually implemented in signal processing algorithms, such as Fast Fourier Transform (FFT). DSPs are more powerful and space-saving than non-specialized GPPs, but DSP development frameworks are generally tighter than those of non-specialized GPPs. For example, DSPs are generally more efficiently programmed in assembler than in higher level languages like *C*. Microcontrollers architectures are optimized for control operations. Their architectures and instruction sets are based on build-in clock operators and a large variety of I/O devices, such as ADCs, timers, UARTs or specialized communications interfaces like I^2C, SPI or CAN. Microcontrollers have the advantage that they can be easily programmed with high-level languages like BASIC. The range of utilization of application specific instruction set processors (ASIP) is tighter, since they are specialized on the common needs of a specific class of applications.

3.3.2 The hardware synthesis

The hardware synthesis aims at developing custom hardware, generally in form of Application Specific Integrated Circuits (ASIC). This activity is concerned with the process that transforms a description of the hardware part of the system in a description of the corresponding chips in a form from which integrated circuits can be manufactured. An ASIC is a processing unit that is tailored for a task and only for this task. ASICs are very efficient in speed and resource usage, since only the resources that are necessary for the application are effectively used. The ASIC solution promises the best economy of resources, space, weight and power consumption. But the lack of flexibility is dramatic with ASICs. Thereto, ASICs are very expensive in the manufacturing.

The hardware synthesis can be divided in two phases: The behavioral synthesis and the register transfer-level (RTL) synthesis. The behavioral synthesis, also called the algorithmic synthesis inputs

a rough specification (state machines, high-level HDLs), where timing constraints on the tasks are not precise, and outputs a collection of registers with the changes that occur on them from a cycle to another. This is then the input for the RTL synthesis, which mostly deals with the combinational optimization of register transfers (netlist) to build optimized circuits for the application under design. The hardware synthesis is a well-established research field. Several commercial tools (also called silicon compilers) exist, that offer automatic transformation from the RTL to a layout specification level. Field programmable gate areas (FPGA) are also often considered as specialized hardware units. Because of its reconfiguration capacities, FPGAs are the first choice candidates for rapid prototyping. The reconfiguration can occur at compiling time (off-line) or at run time (on-line). A further advantage of FPGAs is their parallelism-capable architecture. FPGAs are easily configurable, easier to implement than ASICs and they offer a certain grade of flexibility that ASICs do not have. But they are less efficient that ASICs, i.e. in resources (area) and performance.

Figure 3.5 shows a comparison between specialized and general-purpose hardware components.

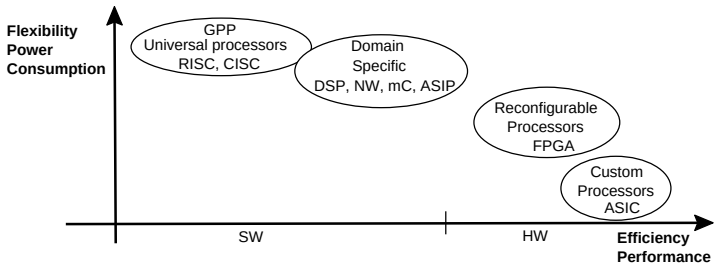


Figure 3.5: Usual processing components

3.3.3 The synthesis of the interfaces

The synthesis of the communication in the context of HW/SW co-design is the concern of the realization of the interfaces between the components of the hardware platform, i.e. the interfaces between the hardware and the software components, the interfaces between the software components, those between the hardware components and the interfaces to the memory elements. After the partitioning, the designer must define the I/O interfaces between the different components of the system as well as the associated communication and synchronization protocols. This also includes the determination of the bus size, the specification of the routing algorithms, the rules for packeting the data for transmission, etc.

3.4 Conclusion

While in the past, hardware architectures dominated the field of high-performance embedded systems design, most of the applications today are implemented in a mix of software and hardware where the software mostly receives the biggest part. The dominance of the software is due to some simple reasons: The flexibility offered by software implementations is welcome in the embedded systems design, as the increasing importance of time-to-market considerations in the business fields using embedded systems can be better satisfied through software implementations. The embedded systems design is actually provided with new (domain-specific) processors that are powerful enough. However, some specific functions will continue to be implemented in hardware for performance and reliability reasons. The HW/SW co-design seems to offer the most adequate design method for developing HW/SW systems, even though it is itself limited. Firstly, the co-design method lacks objectivity: The partitioning is based on the system architecture that is generally decided on the basis of empirical estimations. Other limitation factors include the difficult co-simulation and the difficulty to integrate the hardware and the software parts of the design. A competent co-designer

needs expertise in both hardware and software design, as well as in the design of communication systems.

Notable research have been done to help embedded systems designers and team managers to cope with today's design requirements. "High-level" specification languages have been developed, the partitioning can be automatized (at least partially), there exists a number of software and hardware synthesis solutions of performance. Some industrial tools that are available on the market include: Arexys (SDL, VHDL, C), CoWare (C/C++), LavalLogic (Java to Verilog), Cynlib (C++ to Verilog), Art, Algorithm to RT (C++ to RTL), SUPERLOG (System-level description language). Known academical productions include: POLIS (U.C. Berkeley), PTOLEMY (U.C. Berkeley), VULCAN (Stanford Univ. (Hardware C)), CHINOOK (U. of Washington (VHDL)), COSYMA (U. of Braunschweig (C*)), MEIJE (INRIA and others (Esterel, Lustre, Signal)). But as embedded systems are becoming more and more complex, the level of granularity of the commonly used specification languages in the co-design is too fine to capture large-scale systems. This is particularly the case in the design of AES, the use of hierarchical specification techniques and a specify-refine-verify loop design flow are necessary to cope with the actual complexity.

Chapter 4

Automotive systems design

As a highly constrained domain of production, the automotive engineering needs efficient design methods. In this chapter, we discuss the design of automotive electronics and we present the related problems. In opposition to the actual components-oriented design process, we propose a more efficient design approach, based on the system-oriented perception of the design, that sketches and motivates the idea of co-designing the system's devices. We define the problem to be solved and we precise the solution strategy.

4.1 Design of AES

4.1.1 Top-down and bottom-up

The design of automotive systems can be divided in two views: The OEM's view and the components supplier's view. Figure 4.1 shows the interfaces between the OEM's and the suppliers's responsibilities in the development chain of activities. The OEM is responsible for the whole vehicle. It is its role to define the functionalities of the system, and of course, it is the one that orders the implementations of the corresponding functions. The components suppliers compete to acquire the implementation of the system's components (i.e. in form of ECUs, software components or any other imaginable goods). The successful candidate must deliver the component implementation as specified by the OEM. After validation, the OEM integrates the component into the rest of the system and then validates all together, generally per test or simulation. In many automobile manufacturing companies, the development of the vehicle electronics is planned following some adapted versions of the waterfall model of development, i.e. an activity begins when the preceding activity is supposed to be finished. The relations between the phases of such a development process can be easily represented in form of a V-model like in figure 4.1. This is a top-down design process (left side), completed with a bottom-up implementation process (right side).

4.1.2 Current OEM design practice

Traditionally, the automotive industry sees the development of the electronic components (ECUs, actuators and sensors) like different activities that can be isolated and carried out independently. In the current practice, automotive electronic components are conceived like add-ons for the existing system. As shown in figure 4.2, ECUs and their software are each designed for a particular given functionality. Based on the requirements for the new features, a functional specification is established, then partitioned to define the number of ECUs that should realize the needed functionalities. The OEM then orders the corresponding ECUs from the suppliers and plugs them on the system. After all, a range of tests is done to validate the integration. When the new features are proposed by a supplier, the OEM generally gets a black-box view of the new components that are integrated and tested against their consistency with the rest of the system. This design method

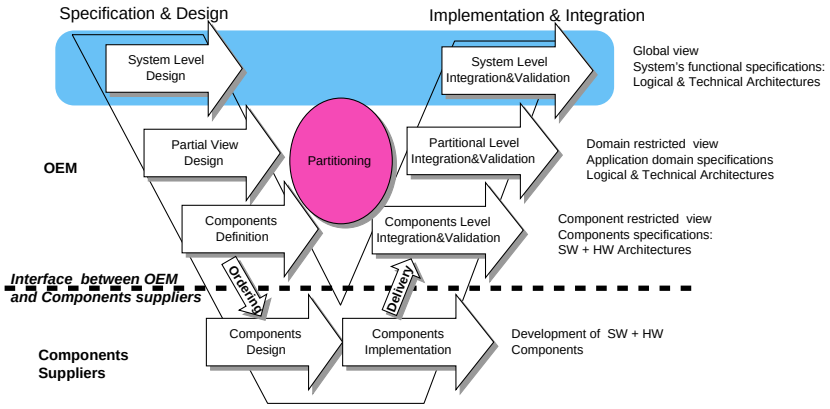


Figure 4.1: OEM and suppliers views in the development process of AES

relies on a component-driven approach. The simplified representation of the development process shown in figure 4.2 makes abstraction of details that are not relevant for our purpose.

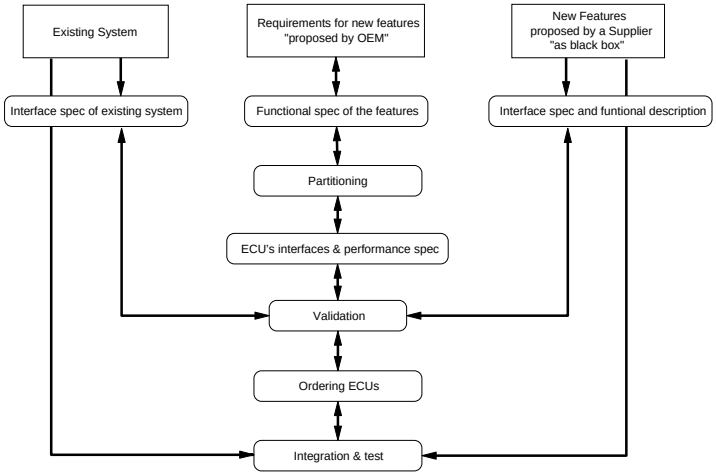


Figure 4.2: The components-based design process

4.1.3 Limitations of the current OEM design practice

In the design scheme described above, each ECU is designed for a fixed given functionality. The OEM's integrators are responsible to bring the different parts of the system together. They try to make this easy by well defining the interfaces between the system's components beforehand. This approach recalls the idea of modularity, well-known in the mechanical engineering, where the automotive engineering, which is originally a machine construction industry, should have accumulated helpful experience. In fact, considering the complexity of automotive E/E systems, the modular design approach is absolutely right. But if the mechanical engineering can easily define stand-alone components, it is because these are essentially hardware pieces (metal, plastics, ...), which do not bear any internal live. The external behavior of these mechanical components is quasi always deterministic and thus, easily predictable. This is not the case for software and embedded electronic components. The clear link between structure and function that is typical to mechanical objects is not evident for intelligent systems. Furthermore, the interactions between the different parts of intelligent systems seriously constrain their distribution. Here, modularity cannot encourage careless separation of things that belong together. Unfortunately, automotive architects today still partition the specifications empirically, according to experience or vague estimations of the inter-components relationships. The consequences are obvious:

- Too many ECUs
- High inter-ECU communication and underutilization of hardware resources, incuring poor cost/performance ratio
- The system integration is difficult and time-consuming
- The quality constraints (e.g. real-time requirements) are difficult to achieve at the same time with the economy goals
- Difficult system scalability, since new functionalities are only introduced through the integration of new ECUs

To solve these problems, the automotive industry designs more and more powerful communication protocols, e.g. TTCAN, FlexRay, Byteflight, TTP, LIN, etc. If necessary, new buses, i.e. wires are added to increase the communication bandwidth. Unfortunately, these solutions are not sufficient! Like the maximal load of the transmission medium, the performance of the communication protocols is limited. Adding buses implies to extend the cable harness, thus to increase the weight, the energy consumption, the product selling price and the technical problems (e.g. robustness). This development process is limited to manage the expansion of the current architecture and is therefore not apt to master the growing complexity.

4.2 Proposed design approach

4.2.1 Factors of the problem resolution

Nowadays, the differences between vehicle constructors in terms of manufacturing ability are pretty slim. The ability to design efficient, reliable and cost-optimal systems on time is essential to survive. In this context, the design flow can make a difference. Our aim is to master the galloping complexity of AES in such a way that avoids the explosion of the system cost by concurrently improving the system performance. AES costs are materialized by the vehicle selling prices and the costs of ownership. The vehicle selling price is based on the engineering costs (NRE) and the costs of the material installed in the system. We can control the costs of AES if we provide efficient design processes and innovative design techniques that can induce positive effects on the engineering costs and optimize the material usage. AES performance is measured on the real-time behavior of the computations in the system components and the timeliness of the communication. Each ECU uses a platform of processors (i.e. microcontrollers, ASICs, DSPs, etc.) and memories on which the associated application runs. We will optimize the costs of the material used in the system if we optimize the costs of the hardware in the ECUs and the cost of the communication system. The latter will be achieved by reducing the cable harness in the system while the former depends on the resource management techniques and the scheduling strategies applied in the ECUs.

In order to decrease the engineering costs and the costs of ownership of AES, a helpful design approach must provide time-saving and flexible design processes. CAD-supported design operations enable short design times and allow easier documentation of the design process. For the optimization of the material usage, we need an efficient resource management that can:

- Reduce or at least slow down the galloping growth of the number of ECUs in the vehicles,
- Reduce the inter-ECU communication,
- Facilitate the system updating and upgrading and
- Simplify the system scalability so that OEM can sell software functionalities on demand, which can be flashed into a vehicle after delivery.

Suitably localized functions communicate cheaper and can easily share the resources. Thus, a goal-oriented approach of the partitioning, which can be automated is the key to master the costs and the performance requirements of the design of automotive electronic systems. Several factors are needed to achieve these goals, for example we must be able to:

- assign functions to ECUs independently of the underlying hardware,
- define the "occupation grade" (working load) of the hardware platform,
- provide efficient design processes with CAD-supported design operations.

4.2.2 The system-oriented design approach

In order to achieve our goal. we propose a system-oriented design approach, i.e. we perceive the electronic system of a vehicle from the beginning as a "whole", joined mixture of hardware and software components rather than a simple collection of parts. So, we can be constantly conscious of the dependencies between the parts of the system. We propose a partitioning approach that can be automated, and thus, enable CAD. The flow of the design as we conceive is sketched in figure 4.3.

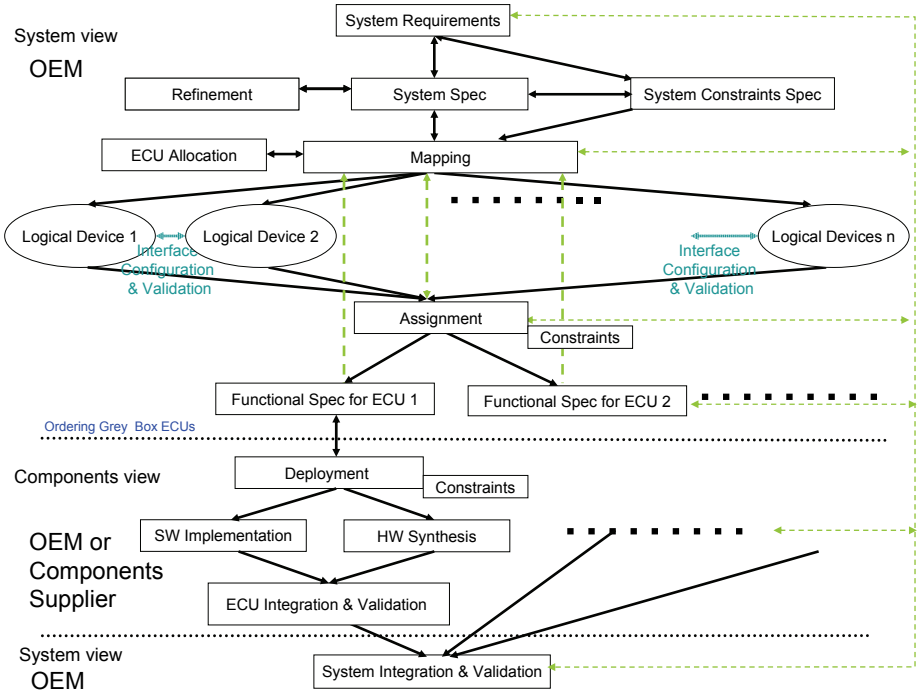


Figure 4.3: System-oriented design approach

The specification: In a system-oriented design scheme, we shall separate the design of the application software from the basic software and the hardware platform, so that the design of the application software is free from the constraints of the hardware and the basic software and vice versa. This separation of concerns allows us to inventory and estimate distinguished design options.

The partitioning: The partitioning is done according to the performance and the cost objectives in conformance with the system constraints. The mapping is the assignment of the elements of the functional specification to the components of the platform, i.e. the functional components are assigned to the devices and the inter-ECU communication data objects are grouped into the frames. The mapping must guarantee the consistencies (in terms of configurability and performance) of the

interfaces between the function clusters. After the mapping, each ECU can be developed following a typical co-design method.

The system integration: The OEM is responsible for the system architecture. It orders the devices from component suppliers and integrates them together. Then, after successful tests, the system is validated.

4.2.3 Analysis

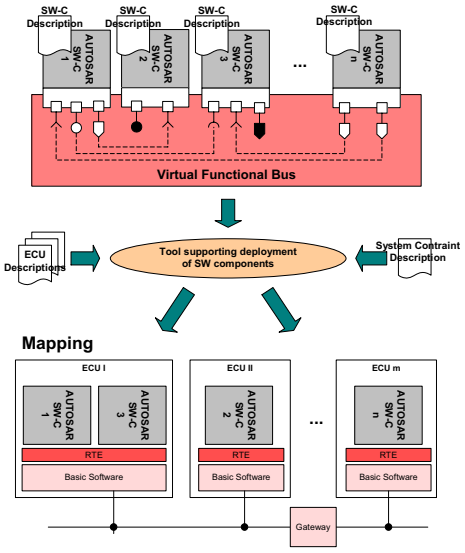
The system-oriented method provides the ability to design more efficient systems, since each component of the system is designed as a part, rather than a stand-alone or a self-completing component. This process reveals several similarities with the concurrent design method of embedded systems as well as with the AUTOSAR design flow:

- The design starts with a system-level specification where the desired system functionality is specified, analyzed and designed as a composite of building blocks,
- Like in the co-design, the partitioning aims at deciding which functions of the specification will be implemented together,
- The logical devices are analyzed, specified and refined concurrently,
- The allocation determines the number, the equipment of the ECUs, the topology of the platform and the communication protocols for the inter-ECU communication, then
- The software/hardware partitioning, the scheduling of the tasks, etc. is done within each device (the deployment) and
- The components of the system, i.e. the devices and the buses, are designed cooperatively and implemented concurrently before being integrated and validated.

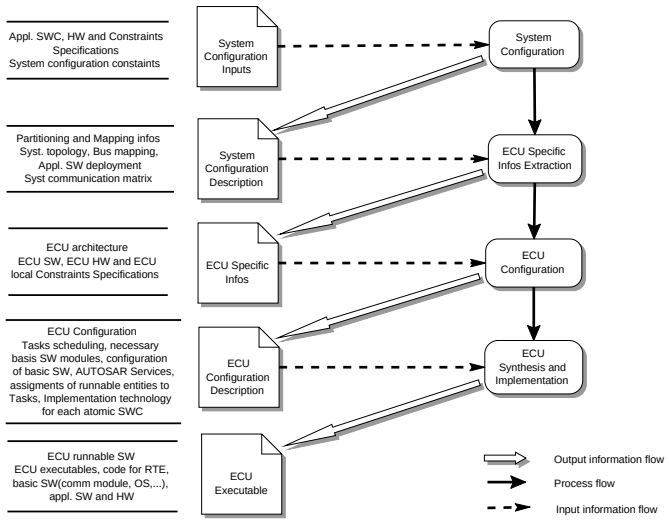
The AUTOSAR proposed design approach begins with a system-level functional description (figure 4.4(a)) where the system functions are encapsulated in AUTOSAR software components that communicate through the VFB. This specification is then partitioned and mapped on the different devices.

Figure 4.4: The AUTOSAR design approach (source: Autosar web content V23.4)

(a) The idea



(b) The activities



The AUTOSAR design flow is roughly sketched in figure 4.4(b). The right-side items represent the design activities and their relationships. The left-side items are the products that result from the preceding design activities. The products are given in the form of documents describing the results of the corresponding activity and the inputs for the following activity. The system configuration inputs include the software components, the allocated hardware and the system constraints. The system configuration, i.e. the mapping of the software components on the devices and the mapping of the frames on the buses outputs the "System Configuration Description" that contains all significant system information such as the topology, the bus mapping, the software components deployment and the associated communication matrix. The system communication matrix is a complete description of the frames (i.e. contents and timing behaviors of the frames) transported around the system. After the system configuration, the devices are configured individually, i.e. for each device, the runnable entities are assigned to the tasks, the tasks are scheduled, the basis software is defined and configured, etc. The result of these operation is described in an "ECU Configuration Description" from which the executables are generated. An ECU executable is the code for the application software, the RTE and the basic software, plus the synthesis of the hardware components of the ECU if some do exist.

4.3 Conclusion

The components-oriented design approach of AES is obviously not capable enough to cope with the problems following the explosion of the number of electronic-actuated features. The most promising solutions to these problems need a system-oriented approach of the design that enables the concurrent design of the components of the system. We propose a co-design approach that makes use of a partitioning of the system-level specification. However, this method will be realizable only if the input specifications fulfill the requirements for the partitioning. Firstly, a system-oriented development method requires a global view of the system, resulting in the handling of very complex models. Thus complexity management and expressiveness are concurrently required for the input specifications. Secondly, a qualified input specification must provide a certain degree of freedom that is necessary to investigate various architectural alternatives. This necessitates platform independent functional specifications and configurable or standardized interfaces of the functional components.

Chapter 5

Modeling AES: State-of-the-art

In this chapter we overview the state-of-the-art in modeling AES. We first present the problematic of the modeling of AES, then we enumerate the features expected from an AES modeling solution. In the following, we outline the techniques used to provide these features and we review the basic concepts used to model AES in the actual state-of-the-art.

5.1 Modeling AES

5.1.1 Model-driven system development in the automotive engineering

The development of automobile E/E (Electric/Electronics) systems has incontestably experienced a great leap forward during the last decade. On the way to its maturity, the Automotive Embedded Systems (AES) design has adopted the model-based development scheme. The purpose of model-based system development is to use models to study the artifacts of a system before building it. Model-based development offers an effective way to decrease the technical and financial risk of "try and error" and improves the economy (design time, material usage, etc.) and the quality (reliability, soundness, performance, electromagnetic compatibility, etc.) of the system. Furthermore, model-based development has the potentiality to boost the innovation, afford collegiate work and simplify the product maintenance. All these concerns are quoted to be vital in the automobile industry. Unfortunately, the state-of-the-art in modeling embedded systems in the context of the automotive engineering does not yet allow the designer to take the best possible advantages from model-based development. In fact, even if modeling is current practice for today's automotive systems designers, models are still considered as simple description and communication media, although in the context of hard competition that rules the automotive industry, modeling can unacceptably continue to be a task that unnecessarily consumes time instead of being helpful and easy.

The overall goal of modeling is to build the system. With a model-based design approach, models are expected to guide the whole design process. That means that all the activities within the life cycle of a car's E/E system, from its conception to its destruction, must be supported by models as far as possible. Thus, here, models are the primary artifacts in the system development process. In our opinion, AES modeling must be outermost concerned with the support for the activities of the implementation phase. Therefore, a model should bear all necessary information needed for the subsequent design operations. Hence, even though models are abstractions of the reality, useful specifications must highlight the system characteristics, motivate the design options and facilitate the design decisions. In the context of embedded systems design, one of the most decisive design operations is the partitioning, i.e. the assignment of the system specification within the available resources. In the current practice, the partitioning is done inputing very low-level, fine-granular specifications (e.g. logical and arithmetical operations or simple assignments). Unfortunately, because of the complexity of AES, this dimension of granularity is difficult to achieve when following a system-oriented design scheme. Coarse granularity is needed to describe such complex systems, but coarse granular models are generally abstract. However, the quality of the

partitioning depends on the information that is available in the input models. As a special domain of interest, important works have addressed the modeling of AES, producing appreciable results. Near general-purpose embedded systems-qualified tools (UML, MatLab, Simulink, SDL,...), more domain-specific modeling languages have been proposed for the development of AES (e.g. EAST-EEA[40], AADL[16], AUTOSAR[22], etc.). But most of these solutions were focused on the definition of modeling languages, neglecting the substance of modeling itself and its potential methodological support for the design process. Some of these modeling solutions are nowadays firmly established in the AES design, but this engineering field is still seeking for a unified, consistent and robust modeling solution that will efficiently support the system development. This is partially due to the very particular nature of AES, i.e. AES are complex and highly heterogeneous.

5.1.2 AES are heterogeneous and complex systems

When considering a system-oriented design style, AES are very complex (in both the size and the functionality) and heterogeneous. In fact, AES are mixed compositions of hardware and software systems that cooperate and also depend of each other. These aggregates are evidently different in their natures and behave almost differently. As AES are multi-dimensional heterogeneous, the design of AES must be able to manage different scales of heterogeneity. For example, even within the hardware and the software, there is a wide range of heterogeneous components, for example GPP and specialized processors, reactive and interactive tasks, event-driven and time-driven tasks, continuous and discrete tasks, control-oriented and data/computation/transformation-oriented tasks, real-time and non-real-time tasks, etc. These items may necessitate different techniques of description and design, but must be handled with the same care. In particular, the design of AES must face the heterogeneity of the components, the heterogeneity of the specification languages, the heterogeneity of the abstraction levels.

The heterogeneity of the components: The electronics of AES consist of ECUs and several communication networks that are realized through cabled, optical or radio wave links running different protocols. An ECU consists itself of hardware and software. Hardware components are of different sorts: GPPs, custom processors (ASICs, ASIPs, FPGAs, ...), memories, I/O devices, timers, etc.

The heterogeneity of the specification languages: The heterogeneity of the components in AES indicates that it may be difficult to describe the whole system in a unique language. Each part of the system may be described using appropriate, e.g. domain-specific languages, resulting in a multiplicity of specification languages for the same system.

The heterogeneity of the abstraction levels: The design process of AES is an agile cycle of specifications, validations and refinements that consequently involves different abstraction levels inducing a vertical heterogeneity. Furthermore, when the elements that have been approved in precedent designs are reused, the same abstraction level may contain objects that are in different stages of development, i.e. in different levels of refinement. This fact yields what we call horizontal heterogeneity.

Hence, handling the complexity and the heterogeneity is a central issue in the AES design.

5.2 AES modeling needs

5.2.1 AES modeling prerequisites

Key prerequisite to manage the complexity and the heterogeneity is a clear separation of concerns. This is achieved through modularization and abstraction principles. Thereto, formal definitions of the modeling concepts would simplify the analysis, the validation and enable design automation. Modularization techniques allow to split the system into smaller parts that are easier to manage. Through this "divide-and-rule" principle, AES are specified as networks of modules, each representing a building block of the system. Usual abstraction techniques include the definition of different

views on the system under construction, the definition of different conceptual and abstraction levels, the use of hierarchical decompositions, and the use of encapsulations. The definition of different conceptual levels of abstraction allows to write valid models with different degrees of precision. So, a specification can begin with non-mature but semantically correct models that will be progressively refined as the design goes forth.

Encapsulation is a means for information hiding, domain delimitation and system isolation. Through encapsulations, the internal properties (structure and behavior) of a model element can be hidden to the others. Encapsulations are necessary to isolate and protect one view from another one, one abstraction level from another one or one element of the specification (e.g. component, connection) from the rest of the specification. Encapsulation affords the detachability and subsequently the reuse of model elements. Anyway, despite the encapsulations, the relations between the views and between the abstraction levels in the specification of a system must be modeled to assure traceability. Traceability across the abstraction levels is modeled by means of the refinement rules. Modularization and abstraction considerably afford the reuse of model artifacts, the flexibility and the evolvability in the design process.

5.2.2 Features expected from an AES model-based design solution

AES system modeling is concerned with the specification of the architectures, the behaviors, the constraints and non-functional requirements of the automotive embedded software and its electronics as well as the transitions within the system development. An architectural model includes the description of the structure of the system and the interactions between its components. It must support modularity, hierarchy and abstraction. Behavioral models describe the system's operation, that is the "sum" of the behaviors of the system components and their interconnections. The behavior of a component is made of two dimensions: The internal behavior and the external behavior. The internal behavior describes the part of the behavior that is concerned with the computations. The computations may be modeled using states transitions and flow graphs, activity schedules or algorithms. The external behavior describes the interactions of the component with its environment, i.e. the communication. Communication models involve the transfer of information between the communicating partners and the relationships (e.g. synchronization) between them. Communication models are concerned with the description of the communication paths between the system modules and the protocols that rule these communications. This may also include further useful information like the scheduling of the communication, the access frequencies, the throughputs, the communication media and the information shared in the communication. Behavioral models must support the modeling of concurrency and time.

Constraints are statements that fix restrictions on the design. They restrain the design space and thus influence significantly the configuration and the implementation of the system. Usually, the constraints and non-functional requirements state obligations on the cost, the performance, the reliability, etc. Transition models are necessary for traceability of the evolution along the design process. Modeling the transitions includes the description of the mappings as well as the refinement rules relating the abstraction levels, the rules governing the instantiations, the configurations and the deployments. This can be done by means of typical deployment and mapping/linkage description tools and some adaptations of feature trees. These points of interest can be placed in different views and treated at different abstraction levels. However, the value of a modeling solution that is eligible for the AES design depends not only on the way it handles these artifacts but also on how it considers the modeling of AES specific domains of interest such as hardware platforms, product lines, non-software components and vehicles environments.

A model-based design framework needs to:

- provide modeling concepts to capture these system artifacts,
- provide modeling languages to express the contents of the models,

- define the design processes, i.e. the steps through which the creation of a system must go, their schedules and the corresponding activities and
- specify the design operations that are needed to perform the activities related to the process of system creation.

5.3 AES basic modeling concepts

5.3.1 Abstraction levels in the AES design

The cost requirements of automotive-embedded systems can only be met if portability and reuse are provided amongst others along the system life cycle. These aims can be achieved through conceptional separation of concerns. Model-Driven Architecture (MDA) proposes a two-layered design approach consisting of platform-independent models (PIM) and platform-specific models (PSM). This approach allows long-term flexibility of the implementation and the integration. In the AES design, the system functionalities are specified as PIMs in so-called "functional architectures (FA)" whereas the resources and the implementation details are handled in more "technical architectures (TA)", so that a FA can be deployed on a wide variety of platforms.

As shown in figure 5.1, the functional architectures focus on the design of the functionalities (functional views) of the system under construction. Typical models found in the FAs include the structural models of product families and the corresponding behavioral models. The system behavior is typically specified with high-level logical information flow models (e.g. sequence, communication diagrams or state charts) or by means of algorithms. The technical architectures contain the models describing the platform (hardware and basic software) on which the application will run (i.e. infrastructural views). Depending on the abstraction level, the hardware platform can be described in the form of a hardware abstraction (indicating which hardware device is installed in the system but abstracting their real properties and locations) or at lower levels with its concrete physical elements (RAM, Flash, ROM, GPP, ASIC, bus, etc.). Typical models found in the TAs include the architectural models describing the platform components and their inter-connections as well as the topological models describing the positioning of the system components. The implementation level is concerned with the actual configuration of the system under construction. It thus focuses on the refinement of the FA with regard to a specific platform. It uses architectural, behavioral and communication models to describe the system. Its primitives include software components and their inter-connections. The implementation models may also include the models to describe the mapping, i.e. the assignment of its building blocks to the platform elements, and the models to describe the allocation, i.e. the equipment of the platform. In contrast, the operational models generally use primitives like processes, tasks and message frames as well as their dynamic aspects (tasks schedules, bus schedules) to describe the system.

In the "EAST ADL" [40], the EAST-EEA has defined five conceptional levels: The vehicle level, the analysis level, the design level, the implementation level and the operational level, containing all together seven architectural models: The vehicle feature model (VFM), the functional analysis architecture (FAA), the functional design architecture (FDA), the logical architecture (LA), also called function instance model, the operational architecture (OA) or allocation model, the technical architecture (TA) or platform model and the hardware architecture (HA). These layers contain the models that specify the system under construction in the intermediate steps of the design process, whereas each layer is a refinement of the preceding one. Following the same objectives, the authors of "AutoMoDe" [44] defined four methodological layers containing each one of the following models: The FAA, the FDA, the LA and then the TA and the OA while the authors of "AML" (Automotive Modeling Language) [45] defined four levels with the following models: The signals, the functions, the LA and the TA, completed with a special implementation level in which issues like code generation and simulation are handled. The LA and the other conceptually closed models are concerned with the design of a particular product that is in general an instance of a family of

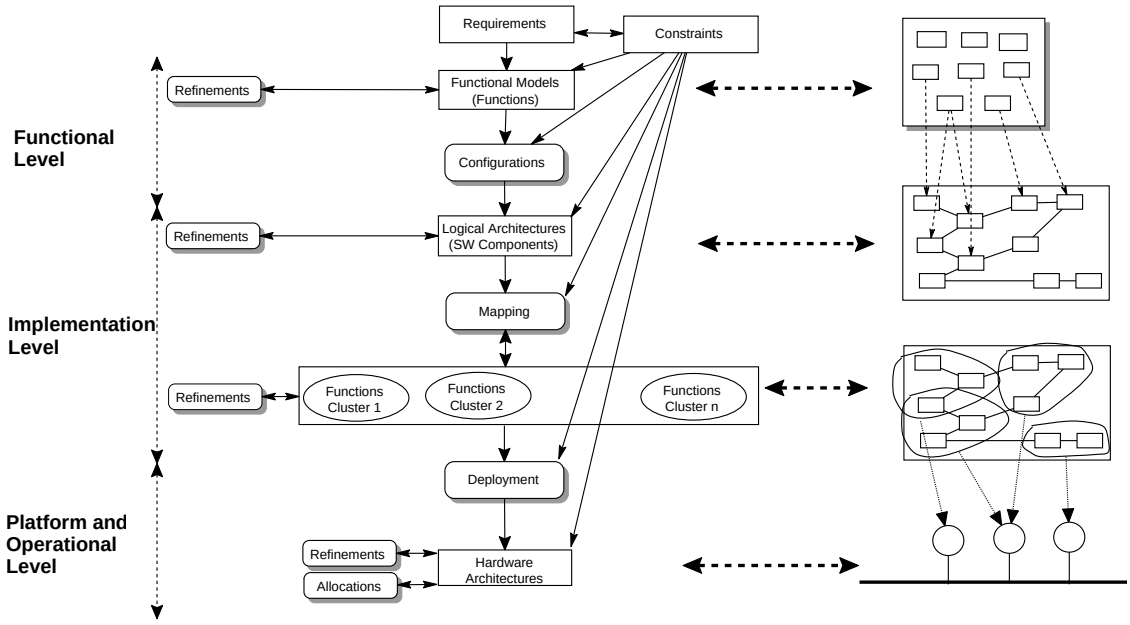


Figure 5.1: AES design conceptual levels

products. As the VFM, the FAA and the FDA are located in the methodological layers that are conceptually above the LA, they support the design of vehicle product families, i.e. product lines, variants and product configurations are modeled in these layers. Thus, the models of these layers describe the system functionalities independently of their integration in a particular product. The VFM is intended to be a direct interpretation of the system functional requirements. Its components are the features that are expected from the system, such as anti-slip control, automatic gear control or automatic lighting control. The FAAs and FDAs deal with the capturing, the structure of the features of the VFM and their decomposition, while the OA is concerned with the deployment of these functions on a given platform. These abstraction levels enable a methodological and conceptual separation of concerns so that the design process can easily begin with rough, abstract models that are progressively refined along the different levels. However, to design systems that are complex and highly constrained like AES, more specialized and methodological conceptual layers might be necessary within the functional views, the implementation views and the infrastructural views (TA) to support the flexibility in the design process and enable methodical and evolutionary model refinements.

5.3.2 AES architectural modeling concepts

AES architectures are modeled as sets of communicating components, i.e. related modules. A system module is either an elementary (atomic) component or a composition (container) of other modules. System modules interact by means of connections. A connection is an architectural building block that models the interactions between other building blocks and the rules that govern these interactions. In some modeling techniques, connections are self-contained modules that in addition to communication protocols incorporate computational behaviors, able for example to transform data or implement concurrency and synchrony mechanisms. Others define connections as simple transportation channels, that just need to have the right caliber to realize the communication. However, accurate modeling of the communication (exchanged information, protocols, bandwidth, medium, etc.) is essential for the partitioning. A connection is realized by a set of connectors. Each connector may represent a channel in which the connected modules may share some information. A module is defined by its substance and its interfacing. In order to enable modular development and reuse, modules are encapsulated. In this case, the interfacing capabilities of a module are specified by its encapsulation, i.e. encapsulations define the interfaces

through which the building blocks communicate. An encapsulation is an architectural building block that hides the proper characteristics of a module to the rest of the world. As the AES design is inevitably concerned with product lines, configuration and refinement rules are needed to assure clear transitions between the different methodological and conceptual layers. For illustration, a particular system is built through configurations of some modules that have been defined as variants in more abstract conceptual layers. Structural configurations describe the rules that govern the construction of valid architectures. For example, they determine if a module is appropriate for a particular product, which modules can be connected, if their interfaces match, if the connectors enable proper communication and if a particular combination of modules results in the desired behavior. The refinement rules describe the rules that guide the transition from an abstraction level to another one. Generally, the functional behavioral modules of the system are identified as functions or (software) components and the links between them are simply seen as connectors. AES functional architectures designers usually do not care about data store modules. Also, in this pragmatic way to simplify things, encapsulations are all supposed to be interfaces with ports [40, 44, 45] like those defined in the UML 2.0. Figure 5.2 shows a UML meta model representing the AES architectures basic modeling concepts as they are generally implemented.

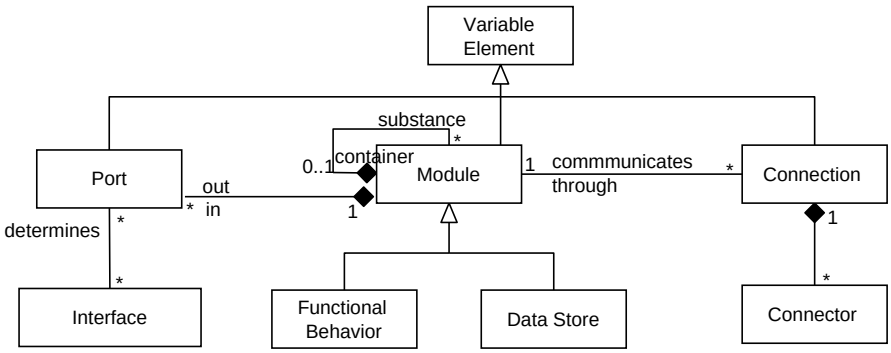


Figure 5.2: AES FAs basic modeling concepts

In contrast to functional architectures that are inherently logical, hardware architectures are described with more physical objects. However, hardware architecture modeling concepts are quite similar to functional architecture modeling concepts. Figure 5.3 shows a meta model of AES hardware platform modeling semantics.

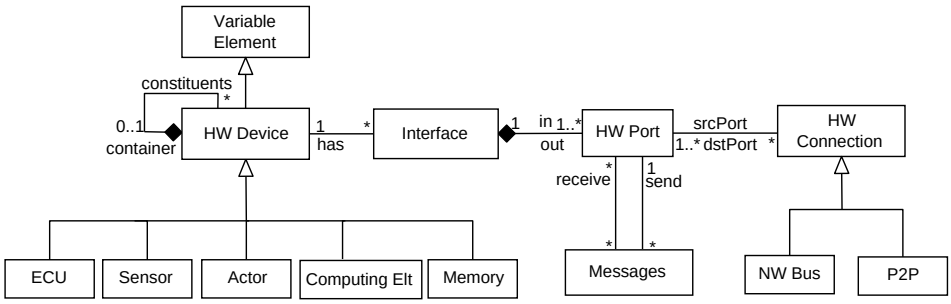


Figure 5.3: AES hardware platforms basic modeling concepts

In hardware architecture models, the basic module is the hardware device. Hardware ports are the points of interaction of hardware devices. A hardware device communicates with its environment through its ports. Hardware ports may stand for fine-granular elements like pins or more confederative entities representing a composition of pins, ports or other hardware devices. The communication is realized over hardware connections. A hardware connection is a hardware architectural building block that models the communication between hardware devices. There are two families of hardware connections: Point-to-point connections and multi-point connections.

P2P connections model exclusive connections between two designated peer ports while multi-point connections model the communication buses. A bus may have several source and several destination devices. Thanks to the concepts of encapsulation (i.e. ports and interfaces), modules and hardware devices are detachable self-contained model elements that might be developed separately. These concepts can be used at all possible abstraction levels. Concerning the partitioning, the quality of an architectural model depends essentially on the grade of precision that is used to formulate these concepts. For example, a hardware device is generally composed of other smaller devices, for example an ECU contains processors and memories while a memory is built up of registers, etc. Depending on the level of precision needed, a hardware device can be a transistor or the whole platform. Figure 5.4 shows some abstraction levels in the modeling of hardware devices.

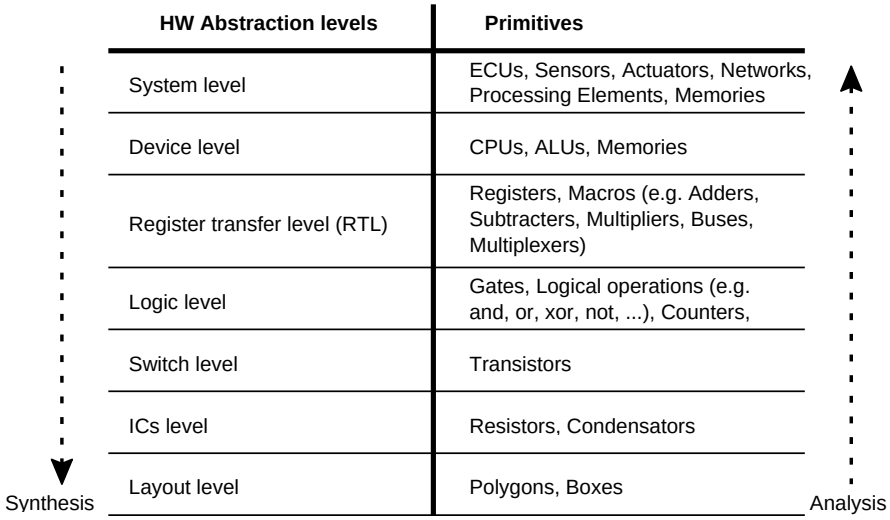


Figure 5.4: Abstraction levels of the hardware devices

5.3.3 AES behavioral modeling concepts

The specification of the behavior of embedded systems is concerned with data transformation, i.e. the computations, and the communication of information. The AES communication schemes typically use a wide range of mechanisms, going from continuous to discrete time communications, synchronous and asynchronous, periodic, sporadic and aperiodic, and time- and event-triggered communications. In the context of concurrency and resource competition between tasks that are omnipresent in embedded systems, synchronization and time paradigms must be supported by AES behavior modeling solutions. Mode transitions, interactions-oriented models (e.g. sequence diagrams), communication diagrams, flow-based models (e.g. data flow graphs, control flow graphs, control and data flow graphs) as well as all possible combinations of these tools have shown efficiency in modeling the behavior of embedded systems[41, 44, 74, 118].

Special adaptations of states, activity, mode transitions, data and process flows have been widely used to capture the behaviors of embedded systems. State-oriented models have been used in form of: Finite State Machines (FSM) [52], Extended FSMs, i.e. FSM extended with data paths (FSMD) [51], FSM with Coprocessors Models, i.e FSMD considering architectures with coprocessors (EFSM, FSMC) [68], Hierarchical Concurrent FSM (HCFSM) [27], Codesign FSM (CFSM) [93, 94]. Examples of Petri nets based-representations of embedded systems include Timed Petri Nets, Timed Place Transitions, Extended Timed Petri Nets (ETPN), Extended ETPN (PURE), Petri Nets-based Representation of Embedded Systems (PRES) [35, 36, 112, 119]. But the most straightforward approaches favor the common programming languages and other effective tools such as Matlab/Simulink, etc. These techniques differently capture the characteristics of embedded systems and therefore, they induce different levels of usage effort with different levels of satisfaction.

5.4 Conclusion

The AES design has adopted the model-based design scheme. The state-of-the-art in modeling AES relies on the components-based paradigm using generic concepts that can be easily adapted to every abstraction and conceptual level. But, as long as these modeling solutions are abstract representations of the system from which accurate information to support the design cannot be extracted, model-driven design will remain a dream. This dream will be realized only if we can achieve useful models at reasonable abstraction levels. In fact, the value of a modeling technique depends on the level of support that it provides for the intended operation. As we intend to partition high-level specifications, it is important to evaluate the fitness, i.e. the level of satisfaction, provided by the available AES modeling techniques for the partitioning. This is the purpose of the following chapter.

Chapter 6

The value of AES modeling languages

In this chapter we evaluate the capacity of the modeling solutions that address the AES domain regarding their ability to support the partitioning. The overall goal of the evaluation is to identify a modeling language that can provide useful specifications for the partitioning. For instance, we need to answer questions like: Which information is needed in a specification to support the partitioning? Which modeling features are needed to provide this information? Do the actual modeling techniques provide these features? How capable are the actual modeling languages? We first enumerate the features that are expected from a model to support the partitioning and we define the framework for the evaluation of the AES modeling solutions. Following this evaluation framework, the most common modeling languages used in the automotive domain are evaluated and classified.

6.1 The evaluation framework

6.1.1 AES modeling requirements for the partitioning

In the design of automotive E/E systems, one of the most decisive operations is the partitioning, i.e. the assignment of the system specification within the available resources. The exploration of the design space allows designers to find the optimal partitions of the system specification by analyzing various alternatives of both the partitioning and the hardware platform of the system. In a model-driven development scheme, design space exploration is enabled by powerful and expressive models. A pool of competing modeling solutions have been proposed to cope with the problems induced by the growing complexity of automotive systems. As the principal features of these solutions are axed around modularization and high level of abstraction, it is necessary to investigate their ability to support the partitioning. An objective evaluation will be helpful to define how each modeling solution should be enhanced for a better support of the partitioning, whenever necessary.

As AES are networks of sub-networks of ECUs, sensors and actuators, we identified two levels for the partitioning in chapter 1 (cf. figure 1.4). The second level of the partitioning, i.e. the deployment, deals with the scheduling of the tasks and the processes on the processing units as well as the scheduling, the synchronization of the communication and the data access proceedings within each device. Making these decisions requires sufficient information about the behavior, the resource consumption, the interdependences, the communication, the performance requirements and the quality constraints of the elements of the functional specification. The first level of the partitioning, i.e. the mapping, aims at distributing the system load among the devices in a manner that the system functioning is optimized by concurrently avoiding resource underutilization. Thus, during the mapping, the functional components of the system specification are distributed within the devices and the inter-device communication data is assignment to inter-device communication channels. The goal of the mapping is to find the most cost-sensitive feasible partition, i.e. a partition that optimizes the resource usage within the devices and minimizes the communication load within the required performance and system constraints (e.g. flexibility, maintainability, power consumption, cost, safety, speeding up, etc.). A feasible partition is one that respects the intake

capacity of the available devices. That is, it must respect the storage capacity of each allocated device, allow executable scheduling of the tasks and enable smooth inter-device communications.

In addition to the resource needs of the elements of the functional specification, the mapping relies on the quality of the information about the inter-components communication, the degree of freedom that is given to the designer and a wide range of relationships between the elements of the functional model, such as those induced by the constraints and the strategic concerns of the design. For instance, the solution space of the mapping depends directly on the degree of freedom that is allowed both by the constraints and the strategical concerns of the design and by the proper definition of the elements of the functional model. The mapping requires that the system components must be each detachable from the rest of the system so that they can be individually assigned to a device independently of the other components. A total freedom is then given for the mapping if each system functional component interfaces the others very loosely. The mapping of the communication data relies principally on the information about the timing behavior of the communication and the magnitude of the communication between the system functional components. Timeliness is determined by attributes like the latency, the max activation time, the transmission time of the components, etc. The magnitude of the communication between two components is given by the access frequencies and the dimensions of the communicated data. To support the mapping, AES input specifications must thus enable to clearly identify the boundaries and the interfaces between the components, identify the communication paths, extract the substance and the heaviness of the communications (e.g. throughputs, access rates, data resolutions, timeliness, priorities, security levels, etc.), find out the dependences and causality relations such as sequentiality, concurrency and the relationships resulting from the constraints and the strategic concerns of the design.

6.1.2 Related work

The aim of the evaluation or the classification of modeling techniques is to measure and compare their potential level of support, their adequacy and their usefulness regarding the requirements of the intended design activity, in our case the partitioning. During the partitioning, the system architect will decide to assign each component of the system to a given part based on attributes like its resource consumption, its size, its needs for computation power, its consumption of energy, the magnitude of the collaboration with the other members of the part and a full range of other significant dependencies. To enable the incorporation of the necessary information in the models, a modeling technique must provide a certain level of precision for structuring paradigms, computation paradigms, control paradigms, communication paradigms and for the specification of the constraints and the non-functional requirements. In this context, precise information about the boundaries and the substance of the system components, the competition for a resource, the timing behavior of the model elements, etc. are very useful.

Several frameworks have been proposed for the evaluation and classification of embedded systems specification tools. The authors of [52] proposed a classification framework based on five specification styles: State-oriented (using state machines), activity-oriented (using transformations), structure-oriented (concentrating on structural architectures), data-oriented (based on information modeling) and heterogeneous. This classification is mainly based on syntactic criteria. However, it can be used to select a specification style depending on the nature of the behavior that needs to be captured. In contrast, the authors of [94] argue for a classification based on the model of computation (MOC) of embedded systems languages. Using the Tagged-Signal Model (TSM) [96], a formalism for the description of MOCs aspects, they focus on timing, concurrency and communication aspects to analyze and classify several MOCs. Generally, as the user is not aware of the MOC of a language, he will not be a priori aware of its quality also. It is thus difficult to apprehend the strength of a language based on its MOC. For instance, a good implementation (i.e. easy and clear syntax, powerful tool support) of a poor MOC is generally far more easier accepted by the user than a poor implementation of a good MOC. However, since a MOC formal-

izes the execution model of a language rather than the style in which specifications are written, this orientation is more objective than the syntax-based classification and is also better adapted to estimate the usefulness of a model according to the task that will be performed with it. MOCs are important characteristics of modeling languages that can not be ignored when evaluating them. Unfortunately, the related taxonomy does not incorporate the important aspects of the AES design such as the structuring capability of a language or its ability to support different abstraction levels. Thus, an exclusive orientation on the MOCs of the languages is not sufficient for our purpose.

Considering the characteristics of specification languages from a very different perspective, Hartenstein [66] used four high-level criteria to classify hardware description languages (HDL): the abstraction level, the application area, the dimension of notation and the source medium of the language. Following this author, the abstraction level characterizes the methodological level for which the language has been designed. The area of application is the type of behavior for which the language has been defined and thus is supposed to be optimized for. The dimension of notation is the general class of information being subject to the description, e.g. behavioral, structural or morphological information. The source medium is the presentation medium, e.g. graphic or textual presentation. The main advantage of this classification framework is its simplicity. This framework is right in our target when trying to classify the languages. But in order to evaluate embedded systems languages, we need supplementary dimensions of criteria.

The authors of [74] first identified four main classes of computation models defined on the vectorial cross product of concurrency (control-driven, data-driven) and synchronization (single-thread, distributed). Then they defined three high-level criteria to compare embedded systems specification languages: the expressive power, the analytical power and the cost of use. The expressive power determines the level of efforts invested when describing a given behavior. The analytical power measures the level of analysis, transformation and verification facilities offered by the language. The cost of use is composed of aspects like the clarity of the models, the quality of the related existing tools, etc. Even though these criteria are very realistic for the evaluation and comparison of embedded system languages, this taxonomy is very abstract and obviously limited regarding the characteristics of AES. Although it allows to consider important AES modeling features such as the conception of timing and concurrency as first class criteria for the evaluation of embedded systems specification languages, the components-based character of AES is not fungible in this taxonomy.

A look into a far different research community lets us discover some frameworks for the classification and comparison of architecture description languages (ADL) [88,100,120] that can efficiently enhance the above mentioned taxonomies [52, 66, 74] with regard to the AES modeling requirements. In fact, in order to classify or compare AES modeling languages, we must also consider automotive-specific concerns, going from the modeling of the system requirements to the description of hardware platforms and the topologies of the infrastructures, including the mapping aspects, the support of different abstraction layers, the support of the design of product lines and the support of non-software (e.g. mechanical) functions and components as well as the environment in which automobiles are used.

6.1.3 Classification criteria

The modeling solutions that are used in the AES design can be distinguished following their originating **specialization**, i.e. the fields of activity for which the solution has been developed, e.g. general purpose, automotive-specific solution, etc. Independently of its specialization, a modeling solution is conceived with focus on a particular **domain of application** or to address some problems that are specific to a given abstraction or conceptual level. For example some modeling techniques are optimized for abstract descriptions while others are more effective for more detailed, fine-grained descriptions. Also, a technique may be optimized to specify only the interactions between the system's modules, but not the computation performed in the modules while another one is designed only to specify the causality and the constraints of the interactions

without detailing the interactions themselves. We retain 5 **domains of application** to classify AES modeling solutions:

1. Requirements modeling,
2. architectures modeling,
3. computations modeling,
4. communications modeling,
5. constraints and non-functional requirements modeling.

The most modeling solutions cover a **scope** of several domains of application. However, for each domain, the modeling techniques differ in the modeling style, the expressiveness, the granularity and the cost of use.

The **modeling style** indicates the style of writing the models when using a modeling technique, e.g. architecture-oriented models may use object- or component-based techniques while behavior descriptions may vary between algorithmic descriptions, differential equations, state- or activity-based models, etc. A classification based on the modeling style can be used to localize the most adequate modeling techniques according to the nature of the system under construction. Components-based techniques seem to provide the best performance for the design and the partitioning of complex heterogeneous embedded systems (see table 6.1).

The **expressiveness** of a modeling technique determines its appropriateness and its usefulness when capturing the characteristics of a specific system. A modeling technique that is not expressive enough to specify a particular item is evidently unsuitable. On the other side, a modeling technique in which the item of interest cannot be described succinctly is also problematic. The expressiveness of a modeling technique is evaluated based on the suitability of the concepts it supports regarding the nature of the information that is to be captured. The suitability is determined by the ease to describe the system and the clarity that can be achieved with a technique. The components of the expressiveness include for example the ability to model the system structures, the support for the modeling of non-software components, the ability to model the computations and the communications, the handling of time and data, the ability to describe concurrency, synchronization and non-functional requirements, etc. Table 6.3 shows a comparison of the expressiveness of the most popular modeling solutions that are used in the AES domain. The criteria measuring the expressiveness will be discussed in detail in section 6.1.4 together with other relevant aspects of the modeling style, the granularity and the cost of use.

The **granularity** determines the dimension of the objects described. In other words, it is the (mean) size of manageable information contained in the elements of the models. Note that even though the dimension as defined here is a relative notion, one should be aware that when we consider a level where the modules are functions realizing complex services, it is clear that simple instructions like assignments are of finer granularity. The size of the objects it manipulates has great influence on the accuracy that a modeling technique can provide. The granularity is measured on the resolution and the level of precision that are achievable with a modeling technique. A classification based on the granularity of the notation of modeling techniques can be used to compare their adequacy according to the level of detail required for the task that needs to be performed with the model. In table 6.1, it can be observed that coarse granular solutions are efficient for high-level abstract modeling while fine granular ones are more adequate for detail and low-level modeling.

AES development is a "team-sport" in which different actors coming from different technical domains act in synergy across different OEMs and suppliers, with different points of interest. A modeling technique must be easy to learn and use, intuitive, capable to capture and visualize domain-specific items, but related to standards and at the best leaning onto formal notations. These features determine the **cost of use** of a modeling technique. The cost of use may include further components like the support of CAD tools, e.g. for edition, syntax check, etc., the executability, the synthesizability, the interoperability with other modeling tools and the visualization

medium. A classification based on the cost of use enables to compare the economics of competing modeling techniques. The cost of use of the 5 languages of interest in this work is given by the "Relation with Standards", the support of different "Abstraction levels", the "Variance and configuration handling" in table 6.2 and by the possibilities for the "Execution" and the "Synthesis" in table 6.3. One will notice that the success of a modeling solution is tightly related with its cost of use. This begins with the affinity with standards. UML and SDL[8] are standards. SysML[12], EAST ADL[40] and AUTOSAR[13] specifications are proposed UML profiles. The efforts invested actually to enable and to promote the executability and the synthesis of these languages testify the importance of the cost of use of a language.

However, even the modeling languages that address the same domain of application and that are deemed adequate for the same conceptual level would differently support the partitioning. The following section defines a framework to capture such differences. Its usefulness is illustrated by a comparison (in table 6.2) and an evaluation of the level of support (in table 6.3) of some high-level modeling languages.

6.1.4 Criteria for evaluating the level of support

The four **dimensions** of the **domains of application** mentioned in section 6.1.3, i.e. the modeling style, the expressiveness, the granularity and the cost of use might be sufficient to classify the AES candidate modeling solutions, but they are still very abstract to enable an evaluation of the level of support that may be provided by a modeling solution. The following taxonomy defines the criteria that indicate the value of a given modeling solution with regard to the partitioning. Depending on the goal of the evaluation, e.g. finding the most adequate, the most useful solution or the one with the best support, a particular combination of these criteria will give the orientation to chose the most appropriate solution.

Modularity: Independently of the domain of application, the modeling style and the expressiveness, each AES modeling solution should provide some modularization features. The modularity support measures the ability to model the structure of the system. The modularity is independent of the granularity but it is an important characteristic of the expressiveness and the modeling style of components-based languages. Clear structuring is provided when the system components are clearly identifiable as detachable building blocks with clear boundaries and interfaces. Fuzzy structuring in contrast denotes the difficulty to identify the components and their boundaries. Clear structuring is a feature characteristic of high-level models. We evaluate the modularity of AES modeling solutions based on the features provided to specify the system modules and the connections between the modules. This includes the substance and the interfaces of the system modules and their connections.

- **The substance:** A component is normally designed to fulfill a given functionality/service. A component may be an atomic structure or a composition of several other components. The substance of a component defines its role and its composition. Both the achievement of a component and its content must be modeled.

- **The interfaces:** The interface of a component depends on the way it is encapsulated. Modeling the interface of a component includes the definition of its points of interaction, what it consumes, what it produces, the constraints on these items and the commitments that are necessary to access them, i.e. the type of information that can be consumed and the protocols allowed to be used for information exchange. Like in [104], some techniques encapsulate the components using wrappers and virtual interfaces that adapt the communication semantics of the components to the needs of its accessors. Other techniques use special connection components like in [103] where an object-oriented approach is presented with an elegant coordinator concept in which the communication of a complex (composite) function is controlled by a coordinator. With this method, the coordinator of a component acts like its communication intelligence. Indeed the coordinator is an intrinsic part of the component. Thus the component's behavior and its communication are always intertwined, making it particularly difficult to separate them and thus to reuse the

component since any instantiation of such a component will require to adapt either the accessing components in the destination model or the coordinator, that means the component itself. In the worst case, both must be redesigned. To separate the communication from the behavior, the most methods propose (in- and output) ports. These methods differ in the power they give to ports and the precision of ports descriptions. Some ports are able to transform data, thus holding complex functionalities. Ports can receive directions, types, etc., that simplify the analysis and synthesis. The modularity of UML2.0, SDL, SysML, EAST ADL and AUTOSAR is compared in table 6.3. All these languages use different semantics for the modules and the substance of the modules, but the ideas of elementary modules and composition of modules are recurrent. Except SDL (because of its OO-orientation), even though the semantics for the interfaces of the modules are different, these languages provide two sorts of interfaces: The receiver and the provider interfaces. Some use ports to structure the interfaces (UML, SysML and EAST ADL) while others (e.g. AUTOSAR) tightly associate the ports with the interfaces.

Resolution of the components: The resolution of a component refers to the granularity of the leaves in its hierarchical structure. A leaf component can be as large as the entire system or as small as a logical operation, an arithmetical operation or a simple assignment.

Computation modeling: A partitioning process will cluster the functions depending on their cost (e.g. computation time), their size, and further attributes like those considering the environment they need to run efficiently (e.g. type of hardware, shareable signals, etc.). Therefore, detailed internal behaviors of components are first-class information for the partitioning, that must be precisely specified. The computation description facilities of a modeling solution are characterized by the type of description used to specify the computations and the provided level of detail. This encompasses the modeling style, the granularity and the expressiveness of these models. State- and activity-oriented behavior descriptions, Petri nets, data and control flow graphs are effective for high-level specifications with coarse granular components but can difficultly provide the accuracy that is necessary to extract the attributes needed for the partitioning, while algorithmic descriptions and differential equations are more detailed, thus more compliant with the requirements for the partitioning. High-level languages solutions for the modeling of computations are generally based on FSMs, use case, activity and state diagrams as documented in table 6.3.

Communication and data modeling: In a communication, a particular type of information is exchanged, e.g. services, messages, operations, data, signals, etc. The type of the information exchanged and the protocol governing the communication strongly constrain the partitioning. AES communication may be synchronous or asynchronous, realized by direct information passing or shared memory, in P2P or multi-cast schemes. At different levels of abstraction, the substance of a communication may be specified in terms of services or operations invocations, messages or data block passing, signals or bits flows, etc., and the communication primitives may variate between call/request, send/receive, read/write, set/reset, load/save, etc. The evaluation of the capability of a modeling technique to model the communication is based on the types of communication supported (cf. expressiveness and cost of use), the tools used to capture the communication (cf. modeling style and cost of use) and the resolution (cf. granularity) of the information exchanged.

Time modeling: Timing information modeling is crucial for embedded systems. The ability of a modeling technique to model time is evaluated through its conceptualization of the notion of time and the resolution of time expression. Time can be expressed through ordering of the activities in the processing (i.e. the order in which things happen induces a notion of time), or as absolute values measured by a clock, this at different resolutions. A modeling technique that can achieve high resolution in modeling timing behaviors is suitable for the partitioning.

Concurrency and synchronization: Embedded systems behave inherently concurrently. Concurrency has two forms: parallelism and interleaving. Parallel processes run at the same time. They may need to communicate and synchronize, for example to publish their beginnings and ends. In interleaving, several processes must compete for resources. Also, in this case, it is necessary to coordinate their interaction. This implies some intelligent synchronization mechanisms

such as schedulers, message queues (buffers), rendez-vous (for message passing), semaphores or read/write blocking in the case of shared memory to assure smooth communication. The evaluation of the ability of a modeling solution to model concurrency and synchronization is based on the concurrency schemes and the synchronization mechanisms that are supported as well as the quality of the concepts that are proposed to capture them.

Relation to standards: The relation to standards measures the distance between a given modeling solution and the nearest standards. The relation to standards determines the intuitiveness of a solution, the facility to learn and to communicate it and the possibility to integrate it with other solutions.

Executability and synthesizability: The executability of a modeling solution refers to the existence of a tool that can be used to simulate the behavior of a system described with this solution. Synthesizable means that there exists a tool that can translate a specified behavior into a machine code or a netlist level model from which properties like memory consumption, hardware size, execution time, etc. can be directly measured. Low-level modeling techniques generally have efficient compilers or synthesis tools that allow rapid prototyping. Some sophisticated high-level models may be executable, particularly when based on formal definitions. Executable and synthesizable modeling solutions have advantageous cost of use.

Abstraction levels: The support of abstraction levels measures the ability to support different, AES domain-established methodological and conceptual abstraction levels.

Support for variance handling: The support for variance handling depends on the quality of the features provided to support the design of product lines. This includes the modeling of varying elements and the description of the configuration information.

6.2 AES modeling languages

Besides the modeling techniques, modeling languages are needed to express the contents of the models. AES modeling solutions generally incorporate each a language. In the reality, the modeling languages have become such a prominence that the whole modeling solution is generally reduced to the modeling language. The spectrum of AES-usable modeling languages is very wide, going from general-purpose programming languages and hardware description languages to architecture description languages and other more specialized languages such as UML, SDL, SysML, EAST ADL, AUTOSAR.

6.2.1 General-purpose languages

Programming languages: General-purpose programming languages, e.g. *C, Assembler, C++, Ada, Pascal, Fortran, Java, etc.* are widely used to specify embedded systems. They are optimized for fine-granular design, provide executable models and possess proved stable compilers. But as typical low-level specification languages, they are very closed to the implementation and they provide poor abstraction capabilities that make them inefficient to specify complex, large-scale systems, i.e. increase in the complexity of the system results in loss of visibility due to the explosive escalation of the size of the model. Components isolation, extraction and reuse are not particularly easy since general-purpose programming languages do not support explicit encapsulation mechanisms. Instead, modularization is implicitly achieved by subprograms definitions, procedures calls and methods invocations. Moreover, since they basically use sequential models of computation, they also drawback in supporting concurrency as well as time modeling. General-purpose programming languages can be suitable for the implementation of embedded systems, but they are definitely not practicable for high-level specifications of complex systems.

HDLs (Hardware Description Languages): During the last decade, a range of modeling languages have been proposed for the specification of mixed software/hardware systems. These languages constitute the family of so-called Hardware Description Languages (HDL), widely used

in the area of HW/SW co-design[1]. HDLs were developed to raise the abstraction level in the design of integrated circuits above the gates level and capture specific embedded systems properties that are not well-addressed with general-purpose programming languages. By today, HDLs have gained maturity. They have mnemonics which are similar to that of programming languages and they are supported by robust simulators and synthesis tools. Some well-known generic HDLs are *VHDL*, *Verilog*, *System Verilog*, *System C* and *Esterel*[6,25]. The electronic design industry is currently successfully using these solutions to design processors, ASICs, FPGAs, memories and other electronic devices, but like the programming languages, HDLs lack complexity management capabilities and high-level abstraction mechanisms. Even if they can better capture concurrency and timing information than programming languages, they still are too closed to the implementation, too fine-granular and thus inadequate for the high-level design of complex systems.

ADLs (Architecture Description Languages): The development of ADLs was motivated by the need of modeling notations providing the constructs for the description of system architectures and their semantics. ADLs consider systems like sets of components related by connectors. They commonly treat components, connectors and their configurations as first-class citizen architectural elements, allowing in this way a basis to achieve modularity, scalability, composition and reuse. Unfortunately, behaviors and time modeling are not in the focus of ADLs. A wide range of ADLs have been developed both within specific domains[28] and as general-purpose architecture modeling languages (e.g. *Acme*[57], *Rapide*[99], *UniCon*[110], *Aesop*, *MetaH*[121], *SADL*[101], *Lileanna*[113], *Modechart*[72], etc.) ADLs usually have formally defined graphical notational syntax that ameliorate their readability. Sophisticated ADLs also provide textual description forms that enable the creation of executable specifications and thus open the door to automatic analysis and synthesis. Because of their abstraction capabilities, they are well-suited for complexity management and provide efficient features to support high-level system design.

UML (Unified Modeling Language): UML is a standard defined by the OMG (www.uml.org). UML is originally a software modeling language based on the OO technology, widely used in the design of business software systems. Since the OMG adopted some real-time and embedded systems optimized concepts like events, actions, resources, schedules and timing, UML is becoming popular in the field of embedded systems software design. In order to address the architectures modeling needs, the new version UML 2.0 introduced a component concept with ports and interfaces, where components communicate through connectors. The separation of components and connectors (i.e. through interfaces) and the explicit treatment of connectors as first-class model elements make UML 2.0 be an ADL. Beside the structure modeling tools (i.e components, classes, compositions, subsystems and deployment diagrams) the UML proposes a wide range of semantics for behavior and interaction modeling that include use cases, activities, states, transitions, communications, sequences, timing and interactions overview diagrams. A component diagram models the system components and their interconnections. Each component owns provided and required interfaces that specify its communication. Ports can be (optionally) used to organize the components interfaces. The UML 2.0 activity diagrams allow the modeling of both the control and the data flows. They also provide semantics for modeling concurrent processes and synchronization. Although UML 2.0 may provide a modeling power that is suitable to capture the behaviors of AES software, it lacks support for AES domain-specific concepts such as requirements specification, product lines, configurations, transitions and hardware resources. However, the UML can be easily extended to address these issues, but at the risk to sheer out of the standard. The properties of the components and the flows on the links are very superficially modeled, i.e. by note boxes used as post-its. Many tools exist for editing, executing and analyzing UML models, but code generation as well as synthesis is still embryonic. The UML is not the ideal modeling language for the low-level design.

[1]Commonly used in the computer science to designate the concurrent design of the software and the hardware parts of a system

SDL (Specification and Description Language): SDL is a general-purpose system description language originally defined for the modeling and simulation of distributed telecommunication systems. SDL is today widely used in the field of embedded systems design. SDL is standardized in the ITU (International Telecommunication Union) recommendation Z.100 [8]. SDL provides four main constructs for system structuring: system, block, process and procedure. A system in SDL is composed of a set of concurrent processes. SDL processes communicate by means of signals. Processes are structured in blocks, that are hierarchical model elements. The system is the outermost block. It builds the higher level in the hierarchy. The inter-block connections are called channels and the inter-process connections routes. Signal paths are thus composed of routes and channels. SDL behavior specifications are based on communicating extended FSMs (EFSM). Each process is implemented by an hierarchical EFSM. Each elementary EFSM is implemented in a procedure. In addition to its ability to model architectures, behaviors and communications, SDL has been extended with real-time concepts. For system engineering, SDL is usually used in combination with MSC, ASN.1 and TTCN (ITU standards Z.105, Z.107). MSC (Message Sequence Charts) is an ITU standard (Z.120) for the description of the interactions between system components. ASN.1 is a language providing powerful means to formally define data types and their structures. TTCN (Testing and Test Control Notation) is a language for the specification of test cases, test suites and test configurations. A UML profile for SDL is in preparation (standard Z.109). Because of its formal semantic, a wide range of tools of performance exist for editing, analyzing, simulating SDL models including automatic code generation, synthesis and performance analysis. SDL provides good message and time modeling features and an appreciable support for synchronization, but draws back in supporting the scheduling of signals and processes as well as broadcast communications. These issues can however be implemented by the user as done in [23]. SDL could be a good solution for modeling AES, but it is OO-based, possesses no compiler and thus provides no support for direct synthesis (i.e. models must be translated into another language, e.g. C, VHDL) and like UML it lacks solutions for AES domain-specific concepts.

SysML (Systems Modeling Language): SysML is a general-purpose modeling language for systems engineering, developed under the commission of the OMG (www.sysml.org) as a customization of UML for system engineering needs. It intends to support the design of heterogeneous systems that may include software and hardware, people, data, etc. in a unified, integrated single modeling tool[12]. SysML adds a requirements diagram to the structure, allocation and behavior diagrams existing in UML. The requirements diagram is defined to bridge the requirements management phase to the subsequent design steps. It supports the description of the requirements, their relationships and constraints, and their relation to the other models, i.e. the specification, analysis and design models. System modules are modeled with SysML blocks. The system structure can be modeled by means of block definition diagrams, package diagrams and internal block diagrams. A block definition includes its structure, its properties, the provided and requested services, i.e. the operations, and the constraints on it. The block definition diagram defines the features of the blocks and the relationships (e.g. associations, generalizations, dependencies) between them. The internal block diagram describes the substance of a block in terms of properties, i.e. data values, parts, references, and internal connectors. The parametric diagrams (other new diagrams not existing in UML) specify the properties, constraints on system properties, e.g. performance, reliability, safety, cost, etc. and their relationships. Parametric diagrams can be used to support the engineering and trade-off analysis, thus the partitioning. Blocks behaviors and interactions can be modeled through state machine diagrams, use case diagrams, sequence diagrams and activity diagrams whereas local and global clocks enable the modeling of timing information. SysML blocks interface through their ports. The services, operations, signals, data, etc. that are provided or expected via a port are specified on the port by means of provider or user interfaces. The ports are associated with the connectors that materialize the communication between them. Two types of ports can be used: Standard ports to specify invocations of services on blocks and flow ports that enable flow of data between blocks. Standard ports are essentially bi-directional. Flow ports

can transmit all kinds of items (e.g. data, energy, material) in a given direction (in, out or inout) between a block and its environment. They can be typed by the type of information that flow through them. Depending on whether a flow port is bound to a property of the owning block or to a parameter of its behavior, it is characterized as behavioral or non-behavioral port. Atomic flow ports (i.e. transmitting only one type of item) can be organized into complex ports transmitting a set of items of several types. A complex port is typed by the list of the items it can transmit. Through the definition of standard and flow ports, SysML provides efficient means to model synchronous, asynchronous as well as P2P and broadcast communications at all conceptional levels of the design. Through the successful extension of UML concepts, SysML is an effective system engineering language that exhibits high capability when used for AES design. The definition of blocks instead of UML 2.0 components and the reenforcement of ports definitions allow the modeling of heterogeneous components, hardware devices, data, energy supply and mixed ports. Furthermore, the separation of block definition diagrams and internal block diagrams introduces implicitly the definition of different views and the separation of conceptual levels of abstraction. The former describes the architecture of the system as needed at higher design levels in the AES design while the latter, as they are conceived to describe the system in terms of processes, properties and constraints, are more suitable for use in the implementation views.

Although SysML provides strong modeling power that can be sufficient to describe the most artifacts that are present in the AES high-level design, they may not match the AES architect's ideas as faithfully as desired. General-purpose languages do not explicitly support automotive domain-specific concerns. In this industry field where the collaboration between OEMs and their suppliers is crucial, where the basic functions are so identical that they can be shared across the constructors, where a product is generally a variant of another one, the need of a clear basis for the communication, the reuse of solutions and the improvement of CAD is obvious. A common modeling approach is thus necessary to facilitate the communication between the stakeholders. This are the principal motivations for the definition of AES domain-specific languages.

6.2.2 Automotive domain-specific modeling languages

Since it focuses primarily on the AES domain concepts, an AES domain-specific language must help to represent specialized automotive domain artifacts directly, helpfully and more easily. In the following, we summarize the main characteristics of the most popular AES domain-specific languages.

EAST ADL: EAST ADL is the ADL defined in the EAST-EEA (Embedded Architectures and Software Technologies-Embedded Electronic Architecture) project (www.east-eea.net) of the ITEA (Information Technology for European Advancement) to describe the embedded electronic functionality of vehicles. EAST ADL intends to provide the means to describe automotive electronic functional architectures from the high-level requirements down to the implementation views. The language is based on UML 2.0, but it also provides concepts to gather automotive domain-specific concerns including the handling of variants, the modeling of hardware resources and the collaboration of OEMs with suppliers. A UML profile for EAST ADL is in preparation. EAST ADL defines seven architectural views that are organized in five conceptual levels[40]. At the vehicle level, the vehicle is described in terms of the functionalities implementing the vehicle features from the users perspective. The Vehicle Feature Model provides the means to specify vehicle electronic architecture product lines. The preliminary decompositions of Vehicle Feature Models are done in the analysis level. Here, the vehicle is described in a Functional Analysis Architecture (FAA). The behavior of the system functions as well as the information that may be necessary for engineering operations (clustering, mapping) can be specified. More concrete specifications can be done within the design level in the Functional Design Architecture (FDA). In the FDA, the system is structured into components following efficiency, legacy, reuse, COTS availability, etc. goals. The low-level

software architectures are specified within the implementation level by means of Function Instance Models (FIM) also called Logical Architectures (LA). This level may contain the code corresponding to the FDA. The last conceptual level, the operational level is concerned with the Allocation Models, the technical architectures, also called Platform Models, the hardware architectures and the Environment Models. The Platform Model contains the basic software and the middlewares available on the infrastructure. The hardware architecture describes the physical elements of the system infrastructure, and the Allocation Model describes the mapping of the FIM software components onto the elements of the platform model while the Environment Model describes the behaviors of the vehicle with respect to its environment and its non-electronic components.

In the FAA, the system functions are described by means of two sorts of modules: The Analysis Functions and the Functional Devices. The Analysis Functions are used to specify the functional components of the system while the Functional Devices are used to abstract the hardware functions, e.g. hardware device managers, sensors and actuators. The environment of the vehicle electronics, e.g. the vehicle dynamics, the road or the driver can be modeled with the Environment Functions. These are hierarchical structures interacting with each other through the connector elements relating their ports. EAST ADL modules interaction points are called Function Ports. Similar to the SysML ports, Function Ports are typed by the interfaces that are associated with them. An interface specifies the type of information that may flow through the associated port. Corresponding to the direction of the flow they are specifying, EAST ADL ports can be defined as required- vs. provided-, or in- vs. out-port as the case may be. All the interfaces associated with a port must have the same direction, the direction of the port. EAST ADL makes a clear difference between signal and operation ports. Signal ports are used to specify the points where signals are sent and received while the operation ports are the interaction points that communicate through operation calls or service invocations. This conception of modularization is similar with the concepts of blocks, class and components used in SysML, UML 2.0 and other ADLs. Another type of port, the System Port, is used to model the access of the application software to the middleware services. This concept allows the abstraction of the application software from the infrastructure (comparable with the VFB of the AUTOSAR) Within the FDA, modules are called composite and elementary software functions with the same ports and connection denominations like in the FAA. The FDA also provides "Precedence" associations to specify the precedence order between the functions. This is important to extract the scheduling constraints of the system functions. In the LA, the modules are function instances and logical clusters. The exchanged data are modeled as signal instances. The hardware modeling elements are hardware architectures, ECUs, Processors, Memories, Pins, Channels, Sensors and Actuators. The EAST ADL platform software model elements are OS, Hardware Abstraction Functions, IPC Exchangers, Middleware-Composite and Middleware-Elementary Software Functions, System Ports, Middleware Local and Middleware Remote Ports. The Allocation Model specifies the OS Tasks, the Communication Buffers, the Frames and the related configuration information.

EAST ADL does not provide any specific behavioral model. It currently considers UML 2.0 state charts and interaction diagrams as its "Native Behavior" models. At this level, every possible high-level behavioral and interaction language, e.g. MSC, SDL, Esterel, etc. may be used. EAST ADL itself does not support synthesis. But the EAST-EEA encourages the connection of low-level behavior models, i.e. detailed descriptions defined in external tools (e.g. Simulink, Statemate), with EAST ADL structural elements. Unfortunately, the description of the EAST ADL behavioral semantics (see [40]) is reduced to the three following fuzzy statements: read input, execute and deliver output. Concretely, that means that to integrate such external modules in an EAST ADL model (for example for simulation), it is necessary that the external module implements the particular interface of each corresponding EAST ADL element. This includes the matching of inputs and outputs ports with the corresponding EAST ADL entities with respect to the data types and the communication properties. Since the EAST ADL interfaces are not yet standardized, this may necessitate complicated integration mechanisms and expensive glue codes. However, the modularization, the ports, the interfaces, the connections and the modeling of their properties

make EAST ADL very compliant to the partitioning needs when regarding the first level of the partitioning as described in section 1.3.

AUTOSAR (AUTomotive Open System ARchitecture): AUTOSAR is a partnership of automotive manufacturers and suppliers (www.autosar.org) currently working for an automotive industry-wide standard vehicle development platform. The declared goals of AUTOSAR include the need to provide the flexibility that will enable product scalability, optimization of the system costs, improvement of the reuse of solutions including reuse across OEMs and suppliers. In order to achieve these goals, the AUTOSAR has defined a methodological process for the system development and a meta model that defines the language for describing automotive E/E systems, i.e. their SW and HW. The AUTOSAR system definition is based on a modular architecture and standardized APIs between system components and between the abstraction layers. The AUTOSAR system description semantics uses components and class diagrams. Stereotype mechanisms and OCL (Object Constraint Language) are provided to define specific semantics and constraints. The AUTOSAR also defines the requirements for the interoperability of authoring tools. A UML profile for AUTOSAR is in preparation [13]. The AUTOSAR methodology defines the technical proceedings for the common steps of the system development process, going form the system-level configuration to the generation of ECU executables including the partitioning of the system. The AUTOSAR Virtual Functional Bus (VFB), i.e. the framework for interconnecting the components of the entire vehicle application, is a powerful mean to enable flexible mapping of the application software on the infrastructure, thus to support the partitioning. In order to be compliant with the AUTOSAR VFB, each function must be encapsulated in an AUTOSAR Software-Component (SW-C). AUTOSAR SW-Cs are system modules with interfaces that are described within the AUTOSAR specification. Standardization of the interfaces within AUTOSAR is a key option to support scalability and transferability of functions across ECUs of different vehicle platforms. In AUTOSAR, the interfaces are classified in three groups: AUTOSAR Interfaces, Standardized AUTOSAR Interfaces and Standardized Interfaces. An AUTOSAR Interface refers to a collection of ports. It defines all the information that may flow through the component. Its use allows to route the information flow through a network in contrast to the Standardized Interfaces that can only be used by components that are located on the same ECU. Standardized Interface is the classifier for those APIs for which a concrete standard exists while a Standardized AUTOSAR Interface identifies an AUTOSAR Interface that is standardized within AUTOSAR. AUTOSAR SW-Cs may contain functions of the application software as well as actuators and sensors software, logical representations of Complex Device Drivers, of the ECU abstraction or the AUTOSAR services. The Complex Device Drivers, the ECU abstraction and the AUTOSAR services are parts of the basic software. Whereas the latter are interfaced through Standardized AUTOSAR Interfaces, the interfaces of the former are ECU-specific. In addition to the standardized interfaces, the AUTOSAR specification defines the concept of "Private Interfaces" for non AUTOSAR-conform software components. However, in order to assure seamless integration and exchangeability of system modules, each component must provide a specific, precise defined functionality through a completely defined AUTOSAR interface.

The communication of AUTOSAR components is modeled by the means of connectors relating their ports. An AUTOSAR port is associated with an interface (called Port Interface) that defines the services or the data that are provided on or requested by the port. A port in AUTOSAR is either a "PPort" (Provide Port) providing information, e.g. Server Port or Sender Port, or a "RPort" (Require Port) requiring information, e.g. Client Port, Receiver Port. A Port Interface can either be a Client-Server Interface (defining the operations or services that can be accessed) or a Sender-Receiver Interface, which allows the usage of data-oriented communication mechanisms. AUTOSAR supports the modeling of both synchronous (blocked client-server) communication and the asynchronous (non-blocked client-server, broadcast sender-receiver) communication. An AUTOSAR SW-C is required to be an atomic block, what means that it cannot be split or distributed on several ECUs. However, when modeling a system within AUTOSAR, its components can be hierarchically grouped in "Compositions" (of components) and then, "Assembly Connectors" are

used to interconnect the components within a composition, i.e. the components of the same hierarchical level, and "Delegation Connectors" are used to delegate connectors from inner ports to delegated ports [13], i.e. to connect ports of components from different hierarchical levels.

Unlike to EAST ADL, AUTOSAR does not explicitly define any conceptual level for the system design. However, the AUTOSAR design process implicitly considers different levels of abstraction that can be summarized in: The design of the functionalities, the identification of the functional units, of (composition) SW-Cs, their decomposition into atomic SW-Cs and then into runnable entities, and in another dimension, the design of tasks and processes. The proposed solution to model components and communication behaviors in the context of AUTOSAR are the UML activity, state machines and interaction diagrams. At the lower level the substance of the atomic SW-Cs is modeled in terms of "Runnable Entities" and "RTE Events". RTE Events (Run Time Environment Events) are generated by the RTE together with the basic software (e.g. OS) to trigger SW-Cs. However, the synthesis of AUTOSAR models will need low-level capable behavior design tools. Apropos, in contrast to EAST ADL, AUTOSAR defines the general use cases and the requirements for the association of AUTOSAR model elements with low-level behavioral modeling tools such as Simulink. The semantics for modeling AUTOSAR systems and ECUs include the typical hardware concepts such as Hardware Elements, Hardware Ports, Signals, and the Hardware Communication.

6.3 Evaluation and classification of AES modeling languages

The mapping is the principal duty of the system architect, that decides which function will run on which device. This operation needs clearly framed functional components with their memory consumptions or their computation times, so that the partitioning can take the intaking capabilities of the different devices into account. Furthermore, the communication paths as well as the timing behaviors and the quantity of the communication between these components are needed to measure their affinities, i.e. the closeness between them, that guide a partitioning. Following table 6.1 we can separate the studied languages into two classes: Those that are more adequate for the high-level modeling and those that are more adequate in the low-level modeling. The first class contains UML, SysML, EAST ADL, AUTOSAR and other ADLs, well-suited for the needs of the high-level design. These languages provide powerful architectural modeling features enabling components detachability, but they lack synthesizability. The second class is populated with the programing languages, the HDLs and all kinds of languages that are similar to those used in Matlab, Simulink, Statemate, ASCET-MD, etc. These languages with high resolution, execution and synthesis tools easily fulfill the requirements related with the executability and the synthesizability required for partitioning-compliant languages, but they are unfortunately too fine-grained and only provide fuzzy structuring capabilities, thus being less efficient in the high-level design. Nevertheless, these two groups of languages are complementary in a system-level design scheme.

However, as we are dealing with a domain where the partitioning shall be done on high-level models, the observed weaknesses of high-level models represent a handicap regarding the input for the partitioning. A way to fill these drawbacks is to extend ADLs with synthesizable languages. Two questions arise here: How should the languages be combined? and What are the best combinations? These questions are not in the focus of this work, but rather, following the discrepancy between the two families of languages, we are interested in investigating the level of support that can be provided to the system architect by the high-level languages as they are found to be the most adequate for the high level of the design. This is done in tables 6.2 and 6.3. In addition to the concepts related with the components orientation, the behavior and the interaction description tools borrowed from the UML 2.0, the automotive domain-specific languages (i.e. EAST ADL and AUTOSAR) and SysML commonly address the domain issues like variants handling, configurations management, hardware platforms modeling, support for non-software components and definition of methodological abstraction levels (table 6.2). This allows them to perform better than the general-purpose languages in modeling AES at the high level. Furthermore, the evaluation of EAST ADL,

	Programming Languages	**HDLs**	**ADLs and affiliates**
Domain of application	computations	computations	architectures
Modeling style	algorithmic	algorithmic	components-based and OO
Modularity and Encapsulation	fuzzy modularity, composition by subprograms, procedures or methods; encapsulation necessitates add-on frames implementations	fuzzy modularity, composition by subprograms, procedures or methods; encapsulation necessitates add-on frames implementations	clear modularity, composition through components-based concepts; clear encapsulation by explicit ports and interfaces
Granularity	fine granularity; high time and data resolution	fine granularity; high time and data resolution	coarse granularity; low time and data resolution
Communication	subprograms calls, methods invocations, shared variables	subprograms calls, methods invocations, shared variables	service invocations, message, signal passing
Concurrency/ Synchronization	from add-ons; weak support	included but weak support	not in focus
Execution/ Synthesis	executable; robust compilers	executable; trusted synthesis tools	difficult execution; synthesis not in focus

Table 6.1: Programming languages, HDLs and ADLs are complementary

AUTOSAR and SysML (cf. table 6.3) shows that they are good basis for an efficient AES modeling solution. Particularly, the standardization of AUTOSAR interfaces allows the designer to switch a function from a device to another one, enabling high-level partitioning. Indeed, the semantics of ports and interfaces are not precise enough to allow a CAD-supported partitioning (table 6.3). However, if enhanced with some features allowing for example clear tracing of the inter-components communication paths, they can be very useful, at least for the mapping.

6.4 Conclusion

High resolution, clear encapsulation, execution and synthesis tools are needed in both the high- and the low-level design while clear modularity is essential in the higher levels. When the design follows a top-down strategy, none of the above languages can be expressive enough to be used efficiently for all purposes along the design process, since each of them offers in reality only a limited set of features. Otherwise, we are not aware of the existence of such an all-rounder general-purpose modeling language. Thus at each step of the development process, the most adequate language should be selected depending on the actual conceptual layer, the level of abstraction and the intended use of the model. The design of AES begins with high levels of abstraction for which the modeling languages like SysML, EAST ADL and AUTOSAR are adequate. Even if the syntax is different from one language to another one, these languages are based on the idea of components-based systems, i.e. they conceive a system as a set of components communicating through interfaces and ports, providing by these means high modularity capability needed for the mapping.

All these languages claim sufficient orientation to the implementation, but they still remain very abstract and lack synthesis and execution tools. As domain-dedicated languages, EAST ADL and AUTOSAR provide the most convenient features and the best precision needed to model AES, but they remain very insufficient to support the partitioning of the system. Firstly, because

they are not synthesizable. Secondly, the semantics of ports, interfaces and connectors are fuzzy. A promising solution to the first drawback is to combine these languages with low-resolution languages such as programming languages, HDLs, etc. But this will not be the ultimate solution for supporting the partitioning of system specifications at the high level. However, if these languages are enhanced with the missing capabilities, i.e. precise computations and communication modeling tools, accurate time and data handling, etc. so that the QoS of the model elements can be extracted and analyzed, then they will represent more appreciable solutions to build partitioning-compliant models for the AES design. The next chapter presents a modeling solution that enhances these languages with some semantical features the aim of which is to enable the screening of the communication paths, the traceability of the inter-components communication data and the particular relationships between the model elements that constraint the partitioning so that an automatic tool can easily extract the properties of the interfacing of the model elements that are necessary for the partitioning.

	UML2	**SDL**	**SysML**	**EAST ADL**	**AUTOSAR**
Specialization	General-purpose system modeling	General-purpose system modeling	General-purpose system engineering for mix SW/HW systems	Automotive domain-specific for embedded SW design	Automotive domain-specific for ECUs configuration and integration
Scope of domains of application	System architectures, behaviors and interactions modeling	Communication	Requirements, architectures, behaviors, interactions and implementation	Requirements, architectures, behaviors, interactions and implementation	Architectures, behaviors, interactions and implementation
Modeling style	Components-based	OO	Components-based	Components-based	Components-based
Relations with standards	OMG standard	ITU standard	Based on UML2.0; Proposed UML profile	Based on UML2.0; Proposed UML profile	Based on UML2.0; Proposed UML profile
Abstraction levels	Not in focus but easy to conceive	Not in focus but easy to conceive	Not in focus but easy to conceive	5 conceptual levels	Not in focus
Support for non-SW components	Stereotype mechanisms; Extension and adaptation of the notion of component	Not in focus	Stereotype mechanisms plus explicit semantics for hardware device and hardware ports	Explicit semantics for hardware device, hardware ports and connections, cf. environment function	Explicit semantics for system model, ECU model, hardware elements, etc., cf. sensor/actuator SW-C
Transitions modeling	Object type definition and unique identifiers	OO concepts of inheritance; Types and subtypes definitions	Object, class and component types definition; Instantiation mechanisms	Explicit binding charts	Unique identification for SW-Cs
Variance handling and configuration	Not in focus	Not in focus	Not in focus	Explicit concepts of varying/configurable elements	Explicit concepts; Central motivation

Table 6.2: Level of support of AES modeling languages with regard to the cost of use

	UML2	SDL	SysML	EAST ADL	AUTOSAR
Structure	Components, classes, subcomponents	System, blocks, processes and procedure	Blocks, parts, packages	Components, composites, clusters	AUTOSAR SW-Cs, compositions
Substance	Atomic components	Elementary processes	Internal blocks	Elementary functions; Clusters	Runnable Entities
Interfaces	Provided and required interfaces	OO interfaces	Provided and user interfaces	Provided and required interfaces, function port and signal port interfaces	AUTOSAR interfaces, standardized AUTOSAR interfaces, Standardized and Private interfaces
Ports	In- and Output ports; Ports are optional	Not explicit	Standard and Flow ports	Functions vs. Systems ports; Provided vs Required, In vs. Out; Signal vs. Operation ports	Provided and Receive ports
Connections	Explicit connectors between components as communication channels	Channels of communication	Explicit connectors for service invocation and Signal transfers between ports	Explicit functions and signal connectors between ports	Sender-Receiver and Client-Server ports
Semantics of ports	Used to structure the interface; Can be stereotyped to contain behavior	Methods	Used to structure the interface; Can be stereotyped to contain behavior	1 port contains n message passing interfaces	Port is interface; Message passing interfaces vs Service invocation interfaces
Data handling	Flexible data type definition	Abstract data type (ADT) and ASN1 data models	Flexible data type definition	Flexible data type definition	Flexible data type definition
Time conception	Order and low resolution duration	Order and low resolution duration	Order and low resolution duration	Order and low resolution duration	Order and low resolution duration
Architectures modeling tools	Components and objects; In components, class, composition, subsystem and deployment diagrams	System, blocks, processes and procedure hierarchies	Blocks and parts hierarchies; internal block and package diagrams	Components diagrams; Analysis, environment, functions, functional devices, elementary and composite functions, etc.	AUTOSAR SW-Cs networks
Computation modeling tools	Use case, activity, state, transition diagrams	Processes, hierarchical extended FSMs	Use case, activity, state and transition diagrams	UML2 state and interaction diagrams	Activity, state charts and interaction diagrams
Communication modeling tools	Sequence, communication, interaction overview, timing diagrams	Parameterized signal passing through routes and channels	Sequence, communication, interaction overview, timing diagrams	Interaction diagrams	Service invocations, signals passing over connectors
Execution/ Synthesis	Executable behavior models, but no synthesis	Executable specifications and wide extensions for CAD tool-supported synthesis	Same like UML	Executable behavior models; Possible mappings to synthesizable languages are in focus	Same like UML, but contains methodologies for mappings to synthesizable languages

Table 6.3: Level of support of AES modeling languages for the partitioning

Chapter 7

Inputs for the partitioning

The overall goal of the partitioning is to assign the elements of the functional specification of the system to the components of the hardware platform, i.e. assign the logical data storage components to the memory components, the behavioral components to the processors and the communication channels to the buses in a manner that the design constraints, such as the functionality, the performance, the cost and the flexibility are fulfilled. The input of the partitioning thus consists of the functional specifications, i.e. the models describing the system's functionality, the platform configuration, i.e. the hardware devices, their compositions and the network infrastructure that is used to interconnect the devices, and the design constraints. In this chapter, we define the inputs for the partitioning of AES. Based on the preceding evaluation of the level of support of the AES modeling languages for the partitioning, we define a system functional modeling solution that is fit for a CAD-supported partitioning. Then we give an insight on the possible AES hardware platforms. Using these inputs, we formally define the partitioning problem.

7.1 Required inputs for the partitioning

The overall goal of the partitioning of an AES is to find an optimal global architecture of the system, i.e. a judicious definition of the composition of the hardware platform of the system, its topology and an intelligent mapping of the functionality of the system on the platform. Within our design process (cf. chapter 1, this objective is to be achieved by reducing the number of frames used to send the messages on the buses in the case of frames-oriented communication protocols. To do that, the mapping must assign the most heavily communicating components of the functional specification to the same device. Thus, the mapping needs a specification of the functional architecture of the system. This specification must include the definition of the structure, i.e. the functional components of the system and their relationships, and the information that allow to measure their communication. The mapping is heavily constrained by the allocation. The mapping must result in executable partitions, i.e. each device functionality must be schedulable and respect the intake capacity of the corresponding device. Furthermore, each device functionality must use the hardware that is installed in the corresponding device optimally, with respect to the defined room that must be reserved for the possible future extensions of the device functionality. Consequently, the mapping needs a precise description of the hardware platform. This description must cover all the devices, the inter-device communication systems, the internal components of each device and the related constraints. For its part, the allocation depends on the working load of the system. We can only determine the number and the equipments of the devices for a given working load if we can measure the execution time, the size of the software code, the magnitude of the communication of the components of the system's functional specification, etc. This information can only be provided if the partitioning is given a consequent description of the behavior and the functioning of the system. Finally, the mapping needs a specification of:

- the logical architecture of the system functionality

- the behavior and the functioning of the system
- the target hardware platform and
- the related constraints

However, these specifications must match the feasibility requirements of the mapping. Matching the requirements of a computer-supplied mapping operation includes the capacity to provide the needed information with the right accuracy and the support of an adequate representation of the necessary artifacts in a format that allows the extraction of the considered decision patterns by the computer-based mapping operation. The latter includes the semantics, the abstraction level, the granularity and the resolution of the input specifications.

7.2 Specifying the system's functionalities

7.2.1 Relevant modeling concepts

Several modeling solutions are used in the automotive domain, e.g. [12], [8], [16], [40], [13]. The most of them are based on the components-based paradigms that provide the abstractions and the complexity management facilities needed in the system-level design of automotive E/E systems [39]. With a components-based modeling technique, AES functional specifications are modeled in the form of separable building blocks, where each building block represents a logical component of the system. A component communicates through its ports and the communication is realized by sharing data or passing messages over logical connectors. A port is associated with one or several interfaces that specify the information that may flow through it.

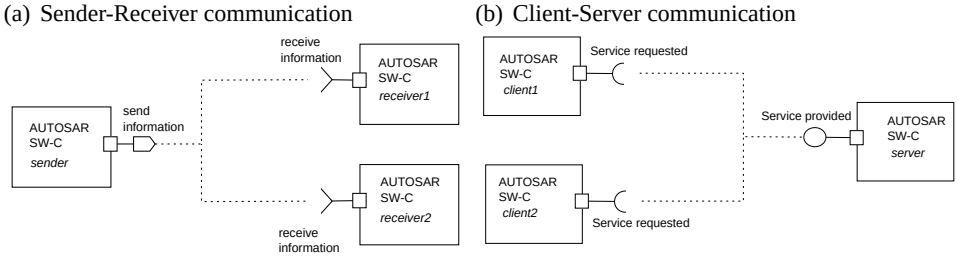
As the above modeling concepts allow to clearly identify the boundaries of each model element, they enable to move each component of the system and assign it individually to a given device. Furthermore, due to the concepts of ports and interfaces, the communication data can be properly specified, at least statically, since all these solutions are lacking the appropriate concepts to describe the behavior of the system. Instead of fine-granular and high-resolution modeling solutions that are desirable for the allocation, they propose high-level modeling tools like state machines or communication, interaction, sequence, data flow diagrams, etc. to specify the behavior of the system. Consequently, because of the low resolution of the proposed behavioral modeling tools, it is not possible to produce a system specification that can be used for a detailed system allocation, and much less for the deployment. In fact, the allocation and the deployment rely on the QoS attributes of the model elements. These attributes include the runtime resource requirements of the functional components, their non-functional requirements, e.g. reusability, upgrading, cost of maintenance, lifetime of operation, and the constraints. The runtime resource requirements of a functional component include the code size, the execution time, the communication load, etc. In order to specify the system behavior so that these attributes can be extracted, we need more detail specifications. These can be provided only for a single component if we do not want to face an order of explosion of the size of the specification that will lead to the loss of visibility and reduce the navigation within the specification, if even this component is not too complex. This is a further reason why the mapping must precede the deployment. After the mapping, we can produce manageable behavioral models for each device and use them for the detail allocation and the deployment.

Although this strategy gives rice to much more loops of allocation-mapping-allocation-deployment-evaluation-allocation-mapping-... than with a straightforward process, we adopt it because of the incapacity of the actual state-of-the-art in the modeling of automotive systems to provide better specifications at the high level of the design. But, even if the runtime resource requirements of the functional components cannot be predicted with the high-level modeling solutions, we can appropriately use the syntax of communication diagrams, sequence diagrams and timing diagrams to specify the communication of the logical components of the system at the level of abstraction

that is inherent to the system-level design of such a complex system, with a precision that can be sufficient to determine the quantity of data exchanged between two components, the frequency of the data exchange, the timing organization of the communication and the constraints on the communication. This solution provides sufficient means to measure the closeness between the system's logical components that is essential to decide if two components must be assigned to the same device or not. Thus, the modeling solutions proposed by the common automotive domain-specific modeling languages can support the mapping, but with a certain degree of accuracy that depends on the granularity of the specifications. Only, with a components-based modeling solution, the granularity of a component is really flexible. A component can be a small piece of behavior, an elementary operation or the whole system. Fine-granular specifications allow fine mappings. With coarse-granular specifications, the components must first be refined into atomic components that can be assigned to a device only entirely or not. This problem has been well-identified within the AUTOSAR.

In an AUTOSAR model, each component is an atomic software component that must be mapped unbroken to a target device. Each AUTOSAR software component communicates via provide and receive ports. A component may also own service ports to interact with AUTOSAR services. Service ports might be bidirectional. Each port is associated with an interface, the port interface. A port interface can be either a sender-receiver interface or a client-server interface. The former is the solution to specify the distribution of data by an asynchronous non-blocking communication method while the latter is designed to be used for client-server communication. As the AUTOSAR promises to standardize the services and the port interfaces of automotive software components, once the corresponding libraries will be available, each unconstrained AUTOSAR component will be mappable on every not forbidden device. This is very encouraging for each partitioner, since the partitioning only makes sense if a certain degree of freedom is allowed for the mapping. In fact, the quality of the partitioning depends on the degree of freedom that the system architect enjoys concerning the targets of the mapping for each component. Note that, in the design of automotive E/E systems, it might happen, e.g. for safety and cost reasons, that some functions become constrained to run on the same device while some others are constrained to run on different devices. The most evidents of these restrictions have also been observed within the AUTOSAR. An AUTOSAR software component is either an application software component, a sensor, an actuator software component, a complex device driver (CDD), an ECU abstraction or an AUTOSAR service. Sensor and actuator software components are AUTOSAR software components that are designed to implement the functionality of a sensor or of an actuator. Thus, the range of their target during the mapping is reduced to the defined sensors or actuators. Furthermore, the substance of an AUTOSAR software component is made of runnable entities that can be scheduled and executed independently from the rest of the component. These runnables are in fact the entities that are deployed.

Figure 7.1: AUTOSAR communication patterns



7.2.2 The FN: The modeling solution for the functional specification

In the end, the AUTOSAR and the similar standard modeling languages cited above provide the most features needed for the mapping. But, although these languages are different regarding

the semantics that they give to the model elements, they are collectively lacking the semantical precision that would support a computer-based mapping. For example, extracting the data that is exchanged on the arms of a branched connector necessitates more screening and analysis effort than on unbranched connectors. We thus added the following semantical precisions to the proved components concepts (i.e. atomic components, ports, interfaces and connectors) and the QoS modeling capabilities provided by the above languages to define a more intuitive modeling solution that meets the requirements of our CAD-supported partitioning: A component is either a functional behavior or a data component, i.e. a data repository, a variable or a memory. A data component is given the semantics of a behavior providing read and write functionalities on the contained data. Like in AUTOSAR, each component is an atomic component. It owns a set of ports through which it communicates with its environment. The set of ports of a component is the components interface. Each port is associated with exactly one port interface. A port interface is solely a container for the information exchanged via the associated port. Such an interface must not implement an API for itself, but it rather describes the information exchanged through the associated port (i.e. data or service). The communication between the components is materialized by means of connectors. A connector is simply a materialization of a connection between two components. It aims at specifying a communication path. This modeling solution underlies the following rules:

- The range of a data object is limited within the connector on which it is transported. If a piece of data is found on different connectors, that means that it is steadily ignored by all the intermediate components. This is important to achieve the tracing of the communication data since with this property the emitter and the receiver of each data object can be directly identified by a simple observation of the data object.

- Each connector has only one originating and one destination component. The profits of this property are clearly illustrated in the example shown in figure 7.3 where the communication paths within the functional specification of the system under design are clearly identifiable.

- Ports are exclusively unidirectional (also see figure 7.3). Unlike the standard languages, we do not allow bidirectional connectors or ports. The direction of the communication is as significant for the issue of the partitioning as the amount of data exchanged. Often, the data exchanged between two ports is different from one direction to the other. This cannot be usefully modeled on a bunched connector as done in AUTOSAR model for example. Furthermore, with bidirectional ports, we deal with hyper graphs, making the partitioning problem harder.

- Only ports of opposite directions can be connected, namely in the direction going from an output port to an input port.

- All the information contained in a port interface must have the same direction. Only input (i.e. required) information can be modeled on an input port and only output (i.e. provided) information can be modeled on an output port or a provider interface.

- A connector relates exactly two ports. Only the ports of different components can be connected. Consequently, a connector can be identified by its originating and its destination port. The number of connectors connecting two components is an important closeness metric since it measures the width of the connection that is an important indicator of the relative importance of a connection.

- Between a pair of ports, there can exist one connector at maximum.

- The set of information flowing on a connector is the intersection of the interfaces of the originating port and the destination port. We define this as the interface of the connector.

As this model is a network of logical components, we call it FN for "Functional Network". Formally, a FN model is a weighted graph, each node of which is either an atomic functional component or an atomic data component and the edges are the connectors described above. We can now define the FN as the following quintuple:

Definition 7.1 (FN:)**.** Each FN model can be formally defined with a quintuple $\langle F, R, P, I, C \rangle$ in which:

- F (Functions) is the set of all the behavioral components in the model, i.e. $F = \{F_1, F_2, ..., F_f\}$ where each F_i represents a functional component; $i, f \in \mathbb{N}$
- R (Repositories) is the set of all data components in the model, i.e. $R = \{R_1, R_2, ..., R_r\}$ where each R_i represents a data component; $i, r \in \mathbb{N}$
- P (Ports) is the set of input and output ports, i.e. $P = \{P_1, P_2, ..., P_p\}$ $i, p \in \mathbb{N}$ with $P = IPorts \oplus OPorts$ (i.e. Input ports $\oplus$ Output ports)
- I (Interfaces) is the set of all the port interfaces in the model, i.e. $I = \{I_1, I_2, ..., I_p\}$ where each I_i represents the interface of the port i; $i, p \in \mathbb{N}$
- C (Connectors) is the set of all connectors in the model, i.e. $C = \{C_1, C_2, ..., C_l\}$ where each C_i represents a connector; $i, l \in \mathbb{N}$
- Each component F_i or R_i is defined by its internal behavior beh and its interface Int, i.e. each component is completely defined by a tuple $< beh, Int >$ with $Int \subseteq P$ and beh is defined by the runnables
- Each port P_i is defined by its behavior beh and its interface Int, i.e. $P_i =< beh, Int >$ with $P_i.Int \in I$
- For each connector C_i, $\exists src \in OPort$, $dst \in IPort$ and Int so that $C_i =< src, dst, Int >$ where $C_i.src$ is the port source of the connector C_i, $C_i.dst$ is the port destination of the connector C_i and $C_i.Int$ is the set of the data that might flow on C_i; $C_i.Int = C_i.src.Int \cap C_i.dst.Int$

Figure 7.2 illustrates the relationships between the FN and the standards.

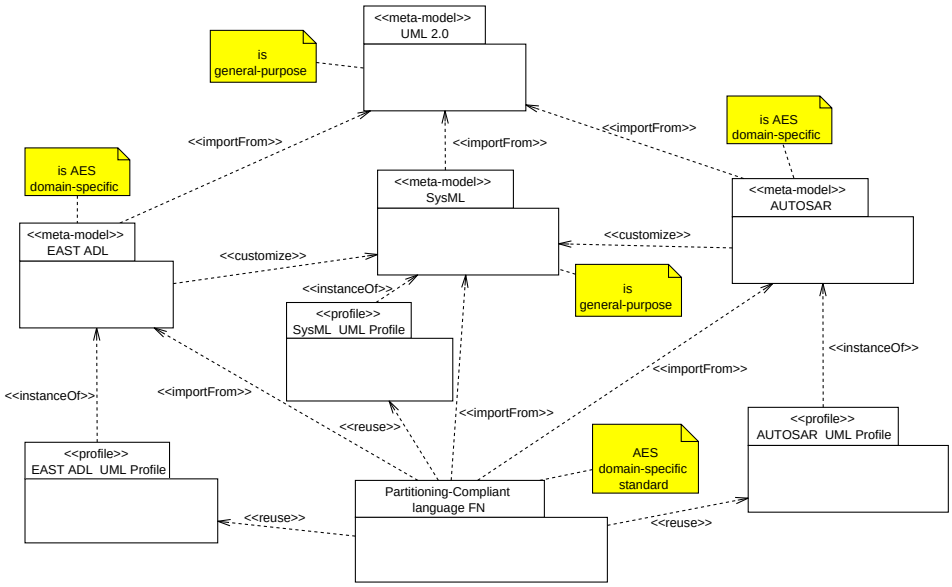


Figure 7.2: The FN inherits the common concepts from the standards

Figure 7.3 shows a graphical representation of a part of the specification of the ACC (active cruise control) following the FN modeling style. The functional components are atomic entities.

The dashed frames represent the data objects that are accessible on the associated output port or that are demanded by the associated input port. These are the ports' interfaces. The data objects contained in each output port interface are all output data (from the point of view of the associated components) while the data objects contained in the input port interfaces are all input data, i.e. incoming data from the point of view of the component that is associated with the port. Such a model meets the humans' natural feeling. With the FN modeling format, the capturing of the functionality of the system under design is as intuitive as the understanding of the resulting specification. The interface of the connector relating two ports is the intersection of the interfaces of these ports. For example, the interface of the connector between the components *Max speed setting* and *Speed control overall* contains solely the data object *speed_selection*, that is the only data object that is communicated through this connector. The remaining data objects are communicated through different connectors, thus with different components. The interfaces of connectors will be further discussed and illustrated in the following chapter (see section 8.3).

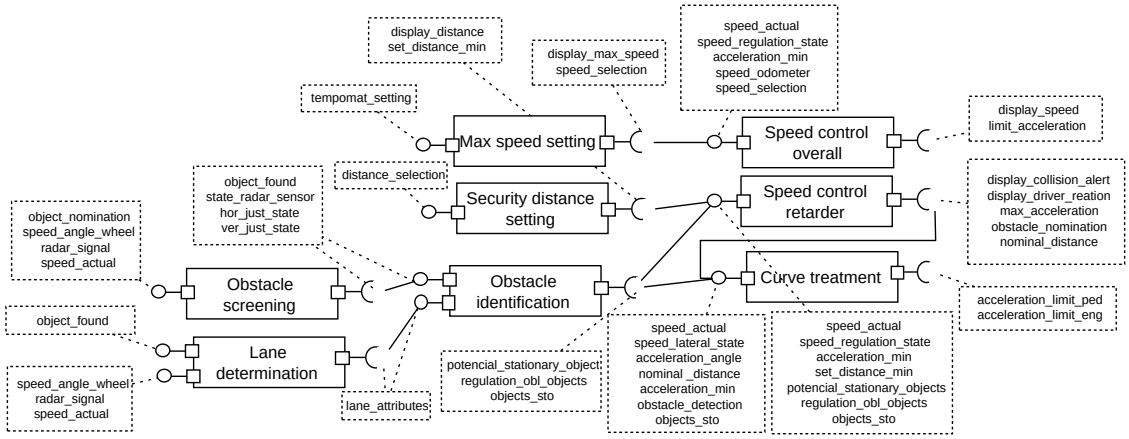


Figure 7.3: A partial graphical representation of the ACC's functionality following the FN format

7.3 Specifying the hardware platform

7.3.1 Relevant modeling concepts

Automotive hardware platforms are networks of communicating multiprocessor systems (i.e. ECUs, sensors and actuators). The choice of a communication protocol follows the topology of the platform and the design constraints. The latter include for example the demand for cost-efficiency, the real-time communication requirements and the requirements for fault-tolerance, robustness, etc. The functional and data components assigned to a device can be implemented on a variety of hardware architectures. Depending on the design options, the device hardware can be a mono- or a multi-elements platform of processing units, memories and on-board communication systems. For example, as micro-controllers generally incorporate a wide variety of hardware modules, e.g. ADCs, I/O devices, timers, UART, SPI and I2C interfaces, RAMs, CAN transceiver, etc., they sometimes represent a good choice to implement a mono-component platform.

The AUTOSAR has defined a generic "AUTOSAR ECU architecture" [22] containing the application software, the basic software and the hardware platform (see figure 7.4). The functionality, that is deployed on an ECU is modeled in the application layer. For each ECU, the corresponding RTE is tailored from the VFB so that only the interfaces and the services that are needed on this particular ECU will be effectively implemented. The RTE is the communication system relating the application layer with the basic software. The basic software implements the infrastructure services (e.g. hardware drivers, OS, ECU abstractions, etc.) and the communication protocols that are needed on the ECU. The basic software layer has a components-based architecture, but it can be

organized in four abstraction layers like in figure 7.5. An example of hardware architecture is given in the ECU-hardware layer. The platform consists of a microcontroller, an ASIC implementing for example a CAN controller, and an external non-volatile memory (flash RAM), interfacing through a SPI or an I2C bus.

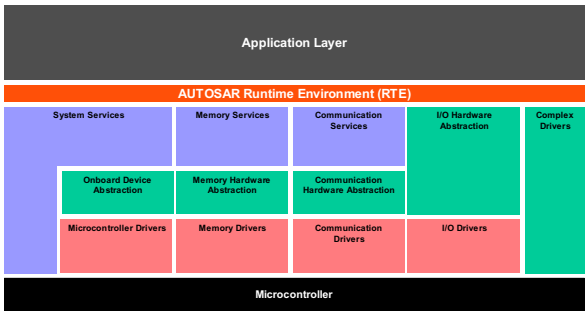


Figure 7.4: The AUTOSAR ECU architecture (source: Autosar web content V2.0.1)

The ECU-hardware or Microcontroller layer contains the real hardware platform of the ECU. Such a platform may contain all types of processors (e.g. microcontroller, ASIC, ASIP, FPGA, DSP, ADC, etc.), memories (e.g. ROM, RAM, Flash, etc.) and communication buses.

The Microcontroller Abstraction Layer (MCAL) is the software model of the hardware platform. It contains the drivers for the internal[1] hardware devices and the memory-mapped external devices (e.g. external flash), and other functions with direct access to the hardware like the test functions for the RAM or the core processor. The MCAL makes the upper layers of the ECU-software independent from the hardware. It provides standardized interfaces between the hardware and the other components of the basic software that need access to the hardware.

The ECU abstraction layer is designed to make the higher software layers independent from the ECU hardware layout. The ECU abstraction layer aims at hiding the location[2] of the ECU hardware devices and their inter-connections. It contains the handlers[3] and the drivers for some external devices, such as external EEPROMs, external watchdogs, etc.

The Services layer builds the larges interface of the basic software to the RTE. It contains the high-level services of the basic software, including the OS services, the vehicle network communication and management services, the memory services, the diagnosis services and the ECU state management services.

The Complex Device Driver (CDD) is a container for the basic software components that are not defined within the AUTOSAR. This includes for example the drivers for hardware devices that are not supported or not foreseen by AUTOSAR such as a driver for the controller of a new communication system, a complex sensor or actuator control module, firmwares and proprietary drivers, etc.

[1] External device is used in this context to identify a hardware device that is located on the ECU but outside the microcontroller. In opposition, a device that is located on the microcontroller is called internal device

[2] Internal devices are located on the microcontroller chip (on-chip) while external devices are on the ECU board (on-board)

[3] A handler is a specific function that controls and synchronizes the access to one or more drivers, i.e. it might perform buffering, queuing, arbitration or multiplexing

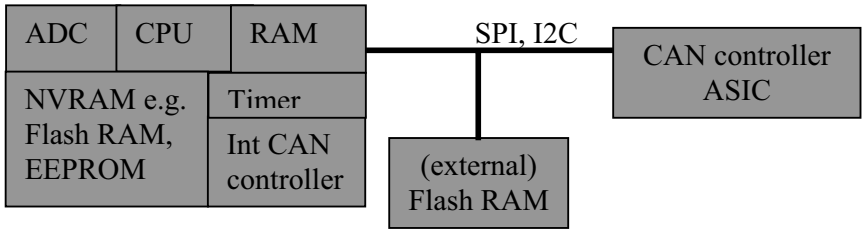
Service Layer -Basic services for the application and basic SW modules e.g. OS, AUTOSAR services, diagnosis service, NW management, etc.	**System services** -Provide basic services for application and SW modules, e.g. RTOS, error manager, library functions.	**Memory services= NVRAM manager** -Abstraction of memory properties and location -Provide data management services e.g. saving, loading, verification, etc. -Provide uniform interface for non-volatile data access	**Communication services** -Provide uniform interface for vehicle NW comm, e.g. for CAN, LIN and FlexRay -Hide protocol and message properties from the application	
ECU Abstraction Layer -SW modules for the abstraction of the location of the ECU-HW devices and the HW layout Location=on chip or onboard	**Onboard device abstraction** -Abstraction of the special ECU onboard self-contained devices -Contains drivers for specific, non usual ECU onboard devices e.g. external watchdog, system basic chip, etc.	**Memory-HW abstraction** -Hide the location of the memories -Offers memory-specific interfaces to access the memory drivers e.g. EEPROM interface, RAM interface -Offers common mechanisms to access internal and external memories	**Communication-HW abstraction** -Hide the location of the ECU-communication controllers and the ECU layout -Offers bus-specific interfaces to access the comm. drivers e.g. LIN, CAN, MOST interface, SPI, I2C interface -Provides uniform bus access mechanisms	**I/O HW abstraction** -Hide the location of the peripheral I/O devices -Hide the ECU-HW layout -Offers common I/O signal interface for accessing the I/O devices
MCAL - Drivers for the HW devices and functions with direct access to HW devices e.g. test functions for RAM, core processor, etc.	**mC drivers** -For internal (on-chip) peripherals e.g. ADC, clocks,etc. -For functions with direct mC access e.g. RAM test, core test	**Memory drivers** -For internal (on-chip) memories e.g. int. Flash, int. EEPROM -For ext. (onboard) memories e.g. ext. flash, external EEPROM	**Communication drivers** -Drivers for ECU onboard comm e.g. SPI, I2C -Drivers for vehicle comm e.g. CAN, FlexRay, LIN	**I/O drivers** - For analog and digital I/O e.g. ADC, PWM, DIO
ECU HW, Microcontroller Layer -HW Platform i.e. buses, microcontrollers, memories, ASICs, etc.				

Figure 7.5: Overview of the AUTOSAR layered ECU architecture

7.3.2 The HN: The hardware platform

Each ECU-hardware is made of processing elements (e.g. processors, DSPs, microcontrollers, ASIC, etc.), memories (volatile and non-volatile) and on-board connections. Most automotive ECUs contain a GPP and several ASICs that implement for example CAN or LIN controllers, PWM modules, etc. DSPs might be used in addition to run computation-intensive functions, such as signals processing algorithms, ADCs, interpolation routines or data conversion functions. The ECU-hardware platform will be a platform of microcontrollers, DSPs, ASICs, etc. and external memories, all related via a bus. Figure 7.6 shows such a platform. The components of this platform communicate through a bus implementing for example SPI or I2C. With this configuration, we gather the principal elements needed to implement the ECU-functionality. Each hardware device contains a certain number of each component shown in this example. However, in reality, some devices might necessitate, let's say no ASIC or no DSP while some others might need several pieces of some of these components or only one piece of some of them. In short, every configuration is possible.

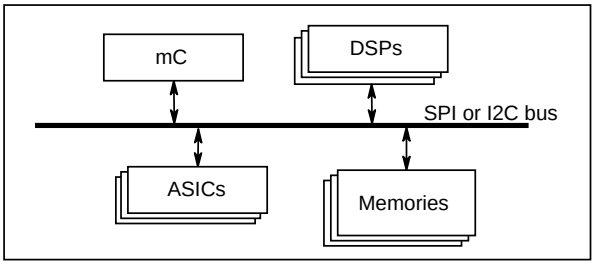


Figure 7.6: Hardware infrastructure of a device

Definition 7.2 (HN:)**.** We formally define the hardware platform of an automobile E/E system with a tuple $\langle D, B\rangle$ representing the hardware resource network HN as follows:

- D (Devices) is the set of all the devices in the system, i.e. $D = \{D_1, D_2, ..., D_k\}$ with $k \in \mathbb{N}^*$ where each $D_i, 1 \leq i \leq k$ represents either an ECU, a sensor or an actuator
- B (Buses) is the set of all the buses in the hardware platform, i.e. $B = \{B_1, B_2, ..., B_q\}$ with $q \in \mathbb{N}^*$ where each $B_j, 1 \leq j \leq q$ represents a bus such as a CAN bus, a LIN bus, etc.
- Each device is a collection of processors, memories and their interconnection systems, i.e. $\forall D_i \in D, D_i = \langle P_i, M_i, OB_i\rangle$ with:
- P_i (Processors) is the set of all the processing elements (custom as well as standard processors) in the device D_i, i.e. $P_i = S_i \cup A_i$, where S_i represents the set of standard processors and A_i the set of custom processors of the device D_i $(1 \leq i \leq k)$,
- M_i (Memories) is the set of all the memories in the device D_i $(1 \leq i \leq k)$, and
- OB_i (On-board buses) is the set all the on-board buses in the device D_i $(1 \leq i \leq k)$.

7.4 The partitioning

7.4.1 Formal definition of the partitioning problem

The partitioning aims at assigning the functional components of the system to the hardware devices and the connector interfaces to the frames with the goal to optimize the inter-device communication and achieve a profitable resource management with respect to the functioning and strategic constraints of the design. The resource usage is optimized when the processors and

memories in the system are not underutilized. The inter-device communication is optimized when the number of messages flowing on the inter-device communication buses is minimized. Based on the formal definitions of the inputs described in sections 7.2.2 and 7.3.2, we can now define the partitioning problem as follows:

Definition 7.3 (The partitioning)**.** Given a FN functional specification $A = \langle F, R, P, I, C \rangle$ and a hardware platform $H = \langle D, B \rangle = \langle \{ \langle P_i, M_i, OB_i \rangle , 1 \leq i \leq k \} , B \rangle$, the goal of the partitioning is to:

- assign the elements of F to the elements of $\bigcup_{1 \leq i \leq k} P_i$
- assign the elements of R to the elements of $\bigcup_{1 \leq i \leq k} M_i$ and
- assign the elements of $\bigcup_{1 \leq i \leq k} C.Int$ to the elements of $\bigcup_{1 \leq i \leq k} OB_i \cup B$

so that the constraints and the objectives of the design are satisfied.

In other words, a partitioning is an operation that assigns the behaviors of A to the processors, distributes the contents of the connector interfaces to either the intra- or the inter-device communication buses and launches the data components of A into the memories of the hardware platform, in a manner that the given constraints are satisfied and

$$assignment(\bigcup_{1 \leq i \leq k} P_i \oplus \bigcup_{1 \leq i \leq k} M_i \oplus \bigcup_{1 \leq i \leq k} OB_i \oplus B) = A.$$

Each output of the partitioning is called a partition of A on H. The partitioning thus consists of finding the best partition of A on a given platform H that satisfies the design constraints. As already mentioned, the assignment process goes through two steps (figure 7.7). The first step, i.e. the mapping, assigns the elements of A to the elements of D and B. The second step, i.e. the deployment, deploys the assignment of each device D_i to P_i, M_i and OB_i $\forall 1 \leq i \leq k$. These are the processors, the memories and the intra-communication buses of the devices D_i:

$partitioning == [allocation][mapping] :: deployment$ where

$mapping : A \stackrel{assign}{\longrightarrow} \langle D, B \rangle$ and

$deployment : assignment(D_i) \stackrel{bind}{\longrightarrow} \langle P_i, M_i, OB_i \rangle \forall 1 \leq i \leq k$

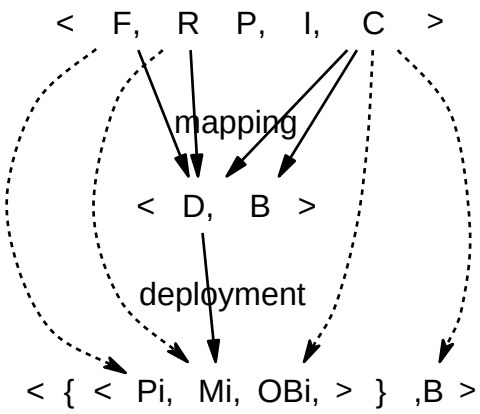


Figure 7.7: Formal definition of the partitioning of automotive E/E systems

7.4.2 Relevant attributes for the elements of the input models

The partitioning is guided by the resource needs of the functional specification and the capacity of the platform. The input specifications must be enhanced with partitioning-relevant information such as the resource requirements of the components of the functional model and their connections, and the amount of resource that is available on the platform. In our input models, this information is specified in the form of attributes. The attributes of the elements of the functional specification (i.e. components, connectors, interfaces, ports) that are relevant for the partitioning are mainly those concerning the consumption of memory, of execution time and of communication bandwidth. The resource consumption of a functional component includes the execution time and the amount of memory needed to store the compiled code (e.g. the object file), the data that is necessary for the component's behavior and the data that is exchanged with its environment. Connectors and interfaces consume the communication bandwidth or (as variables) the memory. The resources that are available on the infrastructure platform, i.e. hardware devices (ECU, actuator or sensor) and buses, are the amount of computation power, memory and communication bandwidth that can be used by the application software on this platform. The hardware resources can be divided into static and dynamic resources. Static resources, e.g. ROM, are assigned for occupation once and kept unchanged. Dynamic resources, for example processors or RAMs can be occupied dynamically by the different consumers that need to share them. In the following, we introduce some partitioning-relevant attributes of the elements of the input models:

Functional components: The attributes describing a functional component that are relevant for the partition include:

- The execution time: The duration (in seconds) of the execution of the code of the component on a given CPU. This is the time required between the starting of the execution of the component and its finishing. Alternatively, the execution time of a data component is the duration of a read or write operation on the memory on which the data is stored. Each data component may thus also have several execution time values depending on the type of memory on which it is stored.
- The execution rate: The period (in seconds) of execution of a behavior. Alternatively, the period of execution of read and write operations on a data component.
- The software code size: The amount (in bytes) of memory required to store the code of the functional component when implemented in software. Alternatively the amount of memory needed to store the data component.
- The hardware size: The amount of hardware material (in number of gates, transistors or logic blocks) that would be used to implement the software component in hardware.
- The basic software: The list of compatible (versions) of basic software modules that are needed by the components.
- The hardware element: The processor type with which the actual implementation is compatible. Alternatively the type of memory that is appropriate to store the data component.

Interfaces: An interface is a container for the data flowing through the associated port. Like in AUTOSAR, the interfaces in FN models describe the static structure of the data. The interface data itself consists either of data elements or operations. Data elements are described by means of data types. Operations are described through their signatures, i.e. the arguments that are communicated between the client and the server. Each argument of a signature has a direction (in, out or inout) and also a data type. For reasons of syntactical clarity, we subsume the information that is exchanged between the nodes of FN models under the common aspect of data objects. The logic of this abstraction is shown in figure 7.8.

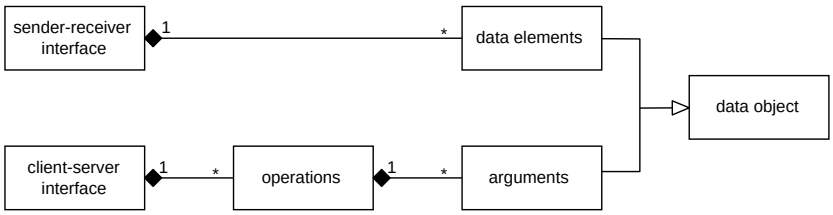


Figure 7.8: Data object is a generalization of operations and data elements

Data objects: The relevant attributes of data objects include:

- The direction: In, out or in-out direction.
- The resolution: The number of bits that is necessary to store the data object. At the implementation level, the resolution of a data object may variate depending on the medium transporting it (e.g. CAN or SPI) or the machine on which it is implemented (e.g. big-endian, little-endian), but at the logical level, the value of the attribute resolution is not supposed to change. For example, if the size of the structure in which a data element is stored is 12 bits, the resolution of this data element will be considered as equal to 12 bits and every possible conversion of the data element will be ignored.
- The access frequency: This attribute indicates the frequency at which the data object is accessed, i.e. the number of times the data is read during a given period. For example during an execution time of the source component.

Ports: The attributes describing a port include:

- The resynchronization time: The time allowed for the resynchronization of the data values after the current data is lost, e.g. after an ECU reset.
- The timeout: The time (in seconds) between the activation of the port and its time out.

Connectors: Relevant attributes of the connectors include:

- The latency: The maximum time allowed for the transfer of a data element from the source component to the destination.
- The jitter: The maximum allowed jitter as a measure of the variance of transport time.

Processors: Examples of the attributes of a processor include:

- The processing power: The speed of computation (in mips, i.e. machine instructions per seconds) of the CPU for standard processors.
- The architecture: The composition and the structure of the processor, the computing paths, the elementary operations, etc.

Memories: Memories can be classified in two categories: Volatile memories and Non-volatile memories. Volatile memories require power to maintain the stored information, e.g. RAMs. Non-volatile memories can retain the stored information even when the power supply is off. Examples of non-volatile memories include ROMs, flash memories and NVRAMs (Non-Volatile RAMs, e.g. flash RAMs). Examples of attributes of volatile memories include:

- The segment size: The size (in Byte) of one memory segment.

- The access type: Read only or also written memory.
- The access time: The mean time for a read, write or erase operation.
- The location: Internal (on chip), external memory (on-board) or stand-alone memory (device).

  In addition to these attributes, non-volatile memories can be characterized by:

- The data retention time: The mean time during which a data is valid within a memory segment.
- The endurance: The number of guaranteed data changes on the memory.

Buses: Examples of the attributes of hardware connections include:

- The bandwidth: The bit rate, i.e. the amount (e.g. in bps, i.e. bits or bytes per seconds) of data that can be transmitted throughout the bus in a fixed amount of time.
- The jitter: The time deviation from the ideal timing of data transmission. The jitter on a bus can follow a random (statistical dispersion) or a deterministic behavior.

7.5 Conclusion

The standard languages provide powerful features to model AES, but because the semantics of ports, interfaces and connectors are fuzzy, they remain very insufficient to support the partitioning. For the functional specification, we have defined a modeling solution that is based on the most common AES specification languages like AUTOSAR, EAST ADL and SysML, so that we keep very close with standards. We enhanced the basic concepts provided by these languages in order to obtain a modeling solution that is fitter regarding the requirements of a CAD-supported partitioning. In addition to the partitioning-friendly features provided by AUTOSAR, EAST ADL and SysML, i.e. atomic detachable components, QoS modeling, etc., the FN provides a clear screening of the communication paths, very useful for the partitioning. We also formally defined the modeling concepts for the specification of AES hardware platforms that allowed us to define the partitioning problem more precisely and formally. However, even at this point in the progression of our work, we have not definitely answered the question relative to the ability of the input models to support a CAD-based partitioning, i.e. Are these models sufficient for the partitioning? If not, how to make them more partitioning-friendly? These questions will be answered in the following chapter.

Chapter 8

The synthesis Model

In the actual state-of-the-art, AES designers input a functional specification in the form of a network of software components (like a FN model) and a HW platform specification in the form of interconnected devices (e.g. a HN model). This models might be explicit enough to describe the system, but they are not optimized to support the partitioning of the system specification. We need synthesizable models that can enable the analysis of the data flow and highlight the closeness between the elements of the specification. In this chapter, we define the synthesis model, i.e. the modeling format that will be used to support the partitioning of AES functional specifications, and the transformation rules that shall govern the translation of an input FN model into this modeling format.

8.1 Definition of the synthesis model

8.1.1 Requirements for the synthesis model

The FN allows clear screening of the communication paths and the tracing of the data flowing upon each connector. But, in order to solve the partitioning problem, we need to transform each FN model into a formal representation on which we can use efficient and reliable mathematical tools to perform data flow analysis and quantify the relationships between the elements of the functional specification of the system. This will be achieved through the definition of an intermediate model that we call the synthesis model. The usefulness of a synthesis model is given by its ability to support the intended design task, in the present case, the partitioning. This includes the ability to reflect the system architecture as given in the FN input model, the ability to specify the information that is needed for the partitioning and the ability to enable rapid estimation of the partitioning metrics. Reflecting the system architecture requires that the intermediate model must be at least at the same level of granularity with the input FN model. A FN model is made of atomic software components like those described in AUTOSAR. The synthesis model must allow to estimate and compare the closeness between different pairs of components. Enabling rapid metrics estimations requires that as much information as possible is known before the partitioning begins, e.g. the access frequencies, the timing behaviors of the inter-components communications, the quantity of data exchanged, the relationships between these data objects and between the components themselves, etc.

Depending on the type of representation used, the formal representations that meet the requirements for the synthesis model can be roughly classified in two groups: Those based on FSMs and Petri nets and those based on graphs. In contrast to graph-based representations that consider a unique system state, FSMs [123] and Petri nets [35, 119] are powerful in modeling and verifying the dynamics of a system. But, FSMs- and Petri nets-based representations are obviously not the best representation forms for systems for which the architecture is important. The main kinds of architecture-oriented forms of FSMs used in the design of embedded systems include the FSM with data paths (FSMD) [53] and the FSM with Coprocessors (FSMC) [75]. But even these special forms cannot reproduce the system's architecture in a useful way. Moreover, they considerably

suffer from the state explosion problem. In the end, as our input models are not aware of any state information, FSMs and Petri-nets are certainly not the solution for our synthesis model.

In contrast, because they more easily reflect the architecture of the system, graph-based representations have been largely used to solve partitioning problems. The most usual graph-based models include data flow graphs (DFG)[38,95], control flow graphs (CFG), data control flow graphs (DCFG)[69] and task graphs. A DFG is a directed acyclic graph in which the nodes represent the system components and the edges represent the data flow. DFGs are well-featured to describe the data dependencies. A CFG is a directed graph of behavioral components in which the edges represent the sequencing of the operations, i.e. the execution order of the nodes. CFGs are well-suited to model control-oriented systems, but they provide restricted facilities for the data flow analysis. CDFGs extend the DFG with control nodes. They provide good models for data flow oriented applications for which the control information is important. Task graphs are similar with DFGs in their structure. But, in opposition to DFGs, special types of task graphs may be cyclic or undirected [26, 95, 122]. Following the configuration of the input models and the requirements for the synthesis model, such a special form of a task graph is probably the solution to our problem.

Various special task graph-based modeling formats have been used for similar problems. In [89], a data flow machine that is built of networked rings is used to analyze the data flow in a network of processors and memories. In [115], a directed task graph, called access graph, is used to model the accesses (i.e. data exchange) between the functional components of the system, while a similar, but undirected graph, called communication graph is used in [21] to model the communication between a set of tasks. All these solutions yield static models that however effectively reproduce the system's structure and the data dependencies.

8.1.2 The synthesis model

The synthesis model is intended to specify the components of the system, their communication and all relevant relationships between them. We chose to represent each synthesis model with a task graph (V, E, Ω, S). Each node $v_i \in V$ represents a behavioral or a data component of the corresponding (FN) input model. If each connector of FN is replaced by an edge in the synthesis model, this will be a multigraph, i.e. a graph with multiple edges between two nodes. Synthesizing the communication between two nodes that are related by multiple edges is much more complicated than when each two collaborating nodes are connected by a single edge. However, reducing multiple connectors to a single one requires that we must ignore the directions of the original connectors. This leads to undirected edges. Undirected edges keep the synthesis model safe from the specification of the direction of the communication and consequently enable the unification of the connectors regardless of their individual directions. Therefore, with undirected edges, each edge $e_{ij} = (v_i, v_j) = (v_j, v_i) \in E$ materializes the communication between the FN components represented by v_i and v_j. Such an edge thus models exclusively the fact that two nodes exchange data in a direction that is ignored by the edge itself or in the two directions. Thus, in the context of the synthesis model, "edges" and "connectors" are semantically identical with the adoption of undirected edges, i.e. the semantics of an edge in the synthesis model is reduced to: "These connected nodes represent two FN components that exchange data in some way".

Evidently, transforming multiple and oppositely directed connectors into a single undirected link gives rise to two problems: Firstly, we need a convenient interpretation of the connections with multiple connectors that will allow to properly capture the data shared between the connected nodes on a single edge. Secondly, as the edges are undirected, we have to specify the direction of the communication somewhere else. The direction and the timing of the communication are essential for the mapping in a frames-oriented communication network, since a device is concerned only with the packaging of the data objects that it sends, but not with the objects that it receives. The packaging and the sending of the tokens are submitted to the constraints concerning the dates of their occurrences and the dates at which they are sent. In order to bundle several differently directed connectors into a single edge, it is necessary to separate the data objects from

the connectors. To do that, we model the data that is exchanged between the nodes of the synthesis model by means of tokens. Ω is the set of the tokens flowing around the graph. A token $T_{ij}^k \in \Omega$ represents the data object k that is exchanged between two nodes v_i and v_j. A token is unbounded in the dimension and is not supposed to contain any additional information such as token delimitation information (i.e. begin of token, end of token), etc. Independently of the connector through which a data object is exchanged in FN, the corresponding token T_{ij}^k is associated with the edge e_{ij} that connects the nodes v_i and v_j. Thus, the set of the tokens associated with an edge models the intensity of the communication between the two nodes. Now, as the edges are undirected, we model the direction of the communication in the tokens. The direction of a token defines the sense of the data transfer on the corresponding edge, i.e. a token has a source and a destination node, both related by the associated edge. So, due to the direction of the tokens, we can distinguish the data sent by a node to another node within the total data exchange between two nodes. Remember that we defined a connection in FN as the interfacing between two components, i.e. the bundle or the set of all connectors connecting two components (see chapter 6).

This definition of the synthesis model leads to the following straightforward transformation of FN models into synthesis models:

- Each component of a FN model is transformed into a node in the corresponding synthesis model.
- Each connection of a FN model is transformed into a single edge in the corresponding synthesis model.
- Each data object exchanged between two components of a FN model is transformed into a token in the corresponding synthesis model.

For illustration, suppose the case of a client-server communication in which a client component v_i produces a token T_{ij}^k and sends it to a component v_j (destination of T_{ij}^k). v_j performs the necessary job with the received data and responds to the client using two tokens, lets say T_{ji}^m and T_{ji}^n. In the corresponding (FN) model, this transaction is modeled by two connectors: One going from the client to the server with the token T_{ij}^k and the other one going in the opposite direction with the tokens T_{ji}^m and T_{ji}^n. But in the synthesis model, the two operations, i.e. the sending and the response, will be modeled by means of a single undirected edge $e_{i,j}$ associated with the three tokens T_{ij}^k, T_{ji}^m and T_{ji}^n. However, due to the direction of the tokens, we can distinguish that token T_{ij}^k is transferred from v_i to v_j while T_{ji}^m and T_{ji}^n flow in the opposite direction. Like the nodes and the edges, each token has a system-wide identifier. This allows us to smoothly identify the tokens among the whole system specification. In fact, the identification of a token is directly related with the corresponding data object and the associated edge, i.e. if the same data element is transferred repeatedly over a given edge, then it will be identified as the same token.

Several mechanisms can be used on this basic model to describe the data-passing procedure. For example, a node can send data by placing it on the dedicated edge, i.e. the token is addressed exclusively to the node connected at the other end of this edge, or the sender can just put the token on its output where it will be collected by the destination node. These two mechanisms are fundamentally different concerning the resulting behavior of the system. The first one processes a peer-to-peer communication while the second one, if not enhanced with restrictive routing rules, is merely adapted to realize broadcast communication, i.e. each component that is related with the sender can access the data that is on the sender's output port. Note that it is also conceivable that the sender node pushes the data to the destination and so synchronous and asynchronous communication schemata can be designed. Defining such mechanisms would introduce a dynamical dimension in the specification of the communication in the synthesis model. But, as the model is not intended to support the simulation, the dynamics of the data exchange and the routing mechanisms will not be discussed in this paper. However, we agree that a token is created as soon

as the corresponding data object is emitted. We then say that the token is available. A token can be sent to the destination nodes only if it is available. The date at which a token is sent is not necessarily the date at which it is made available. A token is available solely means that the token can be sent. A token is available at the date of its creation. But it can be sent later, depending on its freshness requirements. So, while very hasty tokens must be sent as soon as they are available, the sending of less hasty tokens can be delayed. These concepts are introduced to support the scheduling of the communication and the control of the occupation of the communication bus.

In addition to the technical factors of performance and cost optimization such as the communication and the resource usage optimization, the design of an E/E system typically underlies a full range of constraints and strategic concerns relative to the commercial, the technological and the organizational conditions of the design, as well as the procurement and the production issues, etc. Consequently, some components of the functional model might be specified to run on the same device while others are required to run on different devices. The latter is the case when for example it is known that the implementations of two components are not EMC-compatible. The case of the components that must run on different devices include the components that are known to have a high potential to share special hardware resources, those that must preferably be built together by a given sub-contractor that has proved his competency in their specific production or those between which the communication is so constrained that it is not acceptable to take the risk of separating them. It is also conceivable that it may be profitable to group or to separate the components following for example the similarities on their required level of safety, their level of mission or business-criticality, the productivity capability of the source of procurement, the conditions of production, etc. For illustration, components to be implemented in-house might be separated from those to be implemented by sub-contractors and those to be implemented by COTS while components to be implemented by programmer X can be separated from those to be implemented under the responsibility of Y. Further strategic relationships between the components might include the potential level of reuse (i.e. typically reusable components vs. components with low probability to be reused), the rhythm of changing (i.e. frequently changing components vs. rarely changing components), the target of the procured service, the activation time (e.g. components that are active during driving vs. components that are activated during parking), components implementing system internal services vs. components implementing system external services, etc.

These relationships between the components typically have heavy consequences on the partitioning and must be specified in the synthesis model. We do this by means of *needs* and *excludes* relations. Two nodes v_i and v_j are in a *needs* relationship, i.e. $needs(v_i,v_j)$, if they must be implemented on the same device. Two nodes v_i and v_j are in an *excludes* relationship, i.e. $excludes(v_i,v_j)$, if it is forbidden to implement them on the same device. Note that the *needs* and *excludes* relationships are also defined between the tokens. In relation with our taxonomy, all needs-related tokens build a message. Note also that indefinitely similar relationships can be defined on this model. S is the set of these kinds of relations that exist between the nodes and between the tokens of a synthesis model.

In the remainder of this work, we will call this modeling format the "Components Data Flow Machine ($CDFM$)". The $CDFM$ enables the synthesis of the communication by allowing the scanning of the data exchange and the tracing of the communication paths. However, it does not yet contain the information that is needed to guide the partitioning. In addition to the modeling of the components and the inter-components communication, a system partitioner needs the information that will help to decide the destination device of each component and also enable to investigate the cost and the performance of the resulting partition. The *CDFM* provides these information by means of the attributes that capture the runtime characteristics of the model elements.

8.2 Annotations for the synthesis model

8.2.1 Concurrency, sequencing

The information concerning the concurrency between the tasks of a system and their sequencing is certainly important for the partitioning. But, this is not actually explicitly specified in the FN models and thus must not be supported by the synthesis model. Nevertheless, as the connectors specify the data flow, they induce a certain level of data dependency information in the FN models that is replicated in $CDFM$ models, saving their affinity with the FN models. Anyway, we do not have concurrency and sequencing information at this level of the design where we are dealing with software components the implementation of which might give rise to multiple tasks that can be scheduled in various processes depending on whether they need to run concurrently or sequentially. Thus, unless the components of a FN model consist each of a single task, we can neither establish concurrency or sequencing relationships between them, nor can they be reasonably scheduled. However, as we firstly intend to map the components of the input FN models on the devices of the E/E system, we do not need concurrency and sequencing information between them. Concurrency and sequencing information is more useful for the deployment and the scheduling of the "runnables" within a device.

8.2.2 Annotations for the nodes

The performance and the cost of a node are determined by the duration of its execution, the frequentness of its execution, the size of the resulting software code or the size of the hardware that should be needed to implement the component. While the size of the software code is easy to determine, the execution time of a component depends on the behavior of its runnables. As long as a component is not implemented, estimating its execution time is very difficult and even if possible, it might be useless to do that since the component will probably not be executed as a non-interrupted sequence of instructions. Actually, the most components are designed to encapsulate a functionality that is well-known, thus the software code is known or can be estimated, but as they mostly result in several tasks at the runtime, it remains difficult to estimate their execution time. Estimated values are given as requirements for the implementation. Assuming that a particular hardware unit or a family of hardware units have been identified to implement or to store each component so that the memory needs for the code size and for the stacks or heaps for its runnables are known (or can be estimated), the attributes of the nodes of the synthesis model include:

- The **software size (swSize)**: The software size of a node is the total amount of memory required to store the code and the data of the corresponding component when implemented in software. The software code size of a component depends on its implementation. An efficient implementation of a software component may results in the consumption of less memory space. As we consider an optimistic resource reservation strategy, we can take for granted that there is sufficient storing capacity on the platform to store the code of the nodes.

- The **hardware size (hwSize)**: The hardware size of a node is the total amount of hardware components that would be used to implement the functionality of the corresponding component.

- The **execution rate (eR)**: The execution rate of a node is the maximum of the execution rates of the runnables of the corresponding component. The execution rate of a runnable is the number of times that it is executed during an activation time of the system. This is also known as the frequency. If a runnable is periodically executed, its execution rate is easy to determine. For non-periodic runnables, we define the execution rate on the basis of the mean interval between two successive executions. This assumption holds in the most cases since a large majority of tasks in embedded systems are periodic.

- The **priority (prio)**: The priority of a node is the priority order of the most prioritized runnable of the corresponding component. The priority order of a runnable is the priority that it enjoys in the competition for the occupation of a given processor as assigned during the system design.
- The **execution time (eT)**: If available, the execution time of a node is the "sum" of the execution times of the runnables of the corresponding component. The execution time of a runnable is the time between the end of its triggering and the end of its execution. The execution time of a runnable depends on the processor on which it is executed and the context in which it runs. A runnable may thus be characterized by several execution time values, i.e. one execution time for each CPU on which it might run.

8.2.3 Annotations for the edges

The set of tokens associated with an edge is its weight. The determination of the weight of a token is explained in section 8.3.1. An edge may underly some constraints such as the allowed maximum transmission errors (i.e. the jitter) or the allowed maximum latency (i.e. transmission time). For illustration, the attributes of the edges of a $CDFG$ model include:

- The **weight (T)**: The weight of an edge is the set of tokens that flow over the edge during an activation period of the system.
- The **access frequency (accFreq)**: The access frequency of an edge is the access frequency of the most accessed connector within the corresponding connection. The access frequency of a connector is the number of times that the connector is accessed during an activation period of the system.
- The **constraints (cons)**: The constraints on an edge are given by all the constraints on all the connectors of the corresponding FN connection. The constraints on the edges might include the latency, the reliability, the security, the safety, etc. Care must be taken to have consistent sets of constraints.

8.2.4 Annotations for the tokens

Concerning the tokens, the most relevant information for the partitioning include their dimension, their dates of occurrence, their freshness requirements, their direction and their priority, i.e. :

- The **direction (dir)**: The direction of a token defines the sense in which it is transferred. Like the direction of the corresponding data object, the direction of a token is given by the source node and the destination node of this token.
- The **resolution (res)**: The resolution (or the dimension) of a token is the number of bits that is needed to encode the corresponding data object.
- The **frequency (freq)**: The frequency of a token is the frequency at which the corresponding data object is emitted.
- The **priority (prio)**: The priority of a token is the priority level that the corresponding data object enjoys in the occupation of a given communication channel.
- The **date of occurrence (occur)**: The date of occurrence of a token is the time at which it is available. The tokens will be available in the same order with their dates of occurrence. The date of occurrence of a token is different from the date at which it is effectively sent (i.e. the **sending date**). The sending date of a token depends on the scheduling of the communication.

- The **freshness (fresh)**: The freshness requirements of a token determine the latest date at which it must be sent.

- The **constraints (cons)**: The data objects, thus the tokens, may underly some constraints concerning for example their freshness, their safety, etc.

8.2.5 Formal definition of the synthesis model

Given a FN model $A = \langle F, R, P, I, C \rangle$ of the functionalities of an E/E system with the components $M = \{M_1, M_2, ..., M_k\} = F \cup R$, the corresponding synthesis model is a graph $G = (V, E, \Omega, S)$, where $V = M$ is the set of nodes, E is the set of edges $e_{ij} = (v_i, v_j) = (v_j, v_i)$, Ω is the set of the tokens and S is the set of the relationships induced by the constraints and the strategic concerns of the design over the set of the nodes and the set of the tokens, i.e.

***** for each $M_i \in M$ there is a corresponding node $v_i \in V$,

***** for each data object exchanged between two components M_i and M_j there is a corresponding token T_{ij}^k or $T_{ji}^k \in \Omega$,

***** each relation between two components (resp. two data objects) also exists between the corresponding nodes (resp. the corresponding tokens), and:

- Each node $v_i =< swSize, hwSize, eR, prio, eT >$ where

  - $v_i.swSize$ (resp. $v_i.hwSize$) is the software (resp. the hardware) size of v_i
  - $v_i.eR$ is the execution rate of v_i
  - $v_i.prio$ is the priority of v_i
  - $v_i.eT$ is the execution time of v_i

- Each edge $e_{ij} = e_{ji} =< T, accFreq, cons >$ where

  - $e_{ij}.T = T_{ij} \cup T_{ji}$ is the weight of the edge e_{ij}, i.e. of the edge e_{ji}, where $T_{ij} = \left\{ T_{ij}^k, k \in \mathbb{N} \right\}$ is given by the set of the tokens transferred from node v_i to node v_j and $T_{ji} = \left\{ T_{ji}^k, k \in \mathbb{N} \right\}$ is given by the set of the tokens transferred from node v_j to node v_i. Note that $e_{ij}.T = e_{ji}.T$ for all $i, j \in \mathbb{N}$ but $T_{ij} \neq T_{ji}$ for each given pair of nodes i, j
  - $e_{ij}.accFreq$ is the access frequency of the edge e_{ij}, i.e. of e_{ji}
  - $e_{ij}.cons$ is the set of constraints on the edge e_{ij}, i.e. on e_{ji}

- Each token $T_{ij}^k =< dir, res, freq, prio, occur, fresh, cons >$ where

  - $T_{ij}^k.dir$ is the direction in which T_{ij}^k flows. Note that the direction is already given by the foot notation ij of the token
  - $T_{ij}^k.res$ is the resolution of the token T_{ij}^k
  - $T_{ij}^k.freq$ is the emission rate of the token T_{ij}^k
  - $T_{ij}^k.prio$ is the priority of the token T_{ij}^k
  - $T_{ij}^k.occur$ is the date of occurrence of the token T_{ij}^k
  - $T_{ij}^k.fresh$ are the freshness requirements on the token T_{ij}^k
  - $T_{ij}^k.cons$ is the set of the constraints and requirements on the token T_{ij}^k

8.3 Applications

8.3.1 The weight of an edge in a $CDFM$ model

We illustrate the calculation of the weight of an edge in a *CDFM* model with the following simple example. Suppose that two FN components are connected with a sender-receiver port interface, i.e. an interface through which they can exchange data elements, and a client-server interface, i.e. an interface through which operation calls can be initiated. Every 10 seconds, the first component A_1 sends for example the actual distance covered since the very first start of the system as read from the odometer to the second component A_2. This is done through the sender-receiver interface. Then with a time interval of 1 minute, A_1 triggers A_2 periodically so that it calculates the total mileage, i.e. the total number of kilometers that have been covered by the vehicle during the beginning of the actual trip. The mileage is communicated to A_1 that displays it to inform the driver. This is done through the client-server interface. These two communication interfaces are modeled in FN with three connectors (see figure 8.1). In fact, following the semantics of the FN, as A_1 must receive the result of the mileage computation done by A_2, the client-server interface will be modeled by two connectors, one from A_1 to A_2 and the other one from A_2 to A_1.

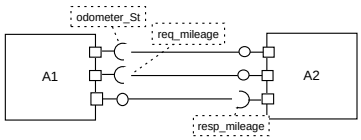


Figure 8.1: The FN graphical representation of the mileage inquiry

The weight of the edge e_{12} of the corresponding $CDFM$ model is the set of tokens exchanged between v_1 and v_2.

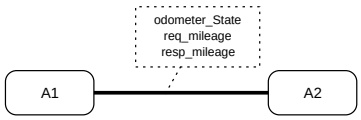


Figure 8.2: The graphical CDFM representation of the mileage inquiry

We now define the metric that we use to determine the weight of the edges of $CDFM$ models: Given a FN model A and its corresponding $CDFM$ model G, consider the operator $width_{ij}$ that defines the set of connectors of A that are represented by the edge e_{ij} in G. These are the connectors that relate A_i with A_j. Assume that the operator $srcConnectors(A_i)$ returns the set of connectors for which A_i is the source and the operator $dstConnectors(A_i)$ returns the set of connectors for which A_i is the destination, i.e.:

$$\begin{aligned} srcConnectors(A_i) &= \{c \in C | c.src \in A_i.Int\} \\ dstConnectors(A_i) &= \{c \in C | c.dst \in A_i.Int\} \\ width_{ij} &= srcConnectors(A_i) \cap dstConnectors(A_j) \end{aligned}$$

Given two nodes v_i and v_j of G, the weight of e_{ij} (i.e. the set of tokens transferred over the edge e_{ij}) is:

$$e_{ij}.T = T_{ij} \cup T_{ji} = \bigcup_{c \in width_{i,j}} c.Int \tag{8.1}$$

Now, assuming that the resolution of the mileage is 16 bits, the parameters sent for the operation call are altogether 72 bits (i.e. operation description, input values and addresses) and the return value is 32 bits, we have:

$|width_{i,j}| = 3$ *connectors*
$T_{12} = \{$*the* 16_*bits_token*, *the* 72_*bits_token*$\}$
$T_{21} = \{$*the* 32_*bits_token*$\}$ *and*
$e_{12}.T = \{$*the* 16_*bits_token*, *the* 72_*bit_token*, *the* 32_*bits_token*$\}$

Given a period of time in which the system is activated, the frequencies of the tokens determine the number of appearances of each token on the associated edge. For example if we consider that the sender-receiver connector is accessed 5 times during an activation period of A_1 and the client-server connector is accessed 2 times during the same period, then:

$T_{12} = 5$ *times the* 16_*bits_token* $+$ 2 *times the* 72_*bits_token*

and

$T_{21} = 2$ *times the* 32_*bits_token*

The date of occurrence of the tokens determines the order in which the tokens appear on an edge, i.e. each token is emitted at a given date that determines its position in the sequence of the tokens associated with a given edge. Thus, the tokens associated with an edge can be ordered in a n-vector in ascending order following their date of occurrence where n is the total number of tokens transferred between two nodes during an execution period of the system.

8.3.2 Model transformation

The following example illustrates the transformation of FN models into $CDFM$ models. Figure 8.3 shows the $CDFM$ model corresponding to the FN model of the ACC (active cruise control) functionality presented in figure 7.3.

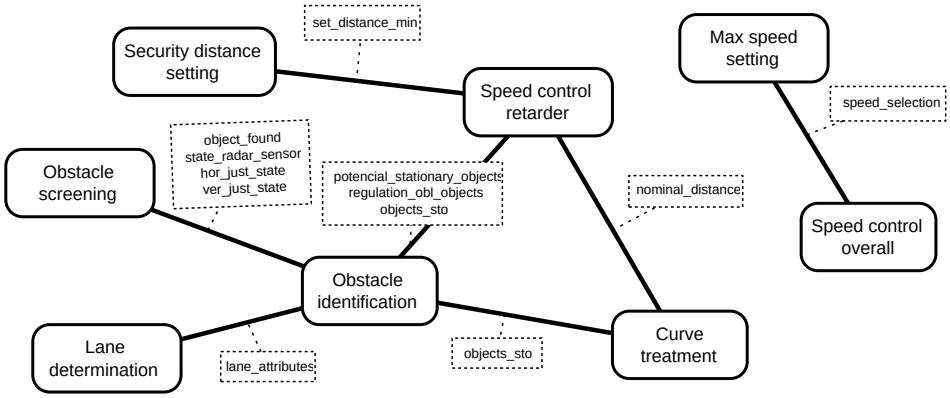


Figure 8.3: Model transformation: The $CDFM$ representation of the FN model of figure 7.3

8.4 Conclusion

The $CDFM$ provides a powerful modeling format for the design of automobile E/E systems architectures. These intermediate models are defined to allow the analysis of E/E system specifications, particularly the synthesis of the data flow for the partitioning. $CDFM$ models are flexible as they can be enhanced without limitation and they do not state any restriction on the granularity of their elements (i.e. nodes and tokens). $CDFM$ models are obtained from a simple and straightforward transformation of FN models. They provide a graph formulation of the system's functional specification that enables the application of usual graph partitioning algorithms for the mapping as well as for the deployment.

Chapter 9

The partitioning: State-of-the-art

In this chapter we at first give a more concrete formulation of the partitioning problem from which we deduce the requirements for the partitioning algorithms that are adequate for our problem. Then the state-of-the-art in the partitioning is outlined. This includes a presentation of the different partitioning methods and the most popular classes of partitioning algorithms. We finally specify the algorithmic process of the partitioning.

9.1 The partitioning problem

9.1.1 Frames of the problem

We consider that the input functional model specifies the functionalities of a given AES functional domain, e.g. power train, comfort, multimedia and telematics, etc. that will be implemented on a network that does not integrate a gateway, e.g. a CAN, a MOST or a FlexRay bus. As the most AES inter-device communication protocols are message-oriented, we consider that the target platform architecture is made of the most popular of them, a CAN bus, on which the devices (ECUs, sensors, actuators) are connected. The partitioning will be done by grouping together-belonging, i.e. related components of the input functional specification, following a set of closeness patterns in order to build the functional clusters that will be mapped on the devices. A feasible partitioning will result in clusters that respect the capacity of the devices and use no more than the available stock of messages to communicate. This means that although extreme load-imbalance must be avoided, the partitioning is not reduced to a load-balancing problem, but is rather an optimization problem, i.e. the communication overhead must be minimized, the real-time communication of the system must be optimized, the resource utilization and the dependability of the system must be maximized by avoiding computing and communication faults. However, following the actual trends in the automobile market, although the goal of this work is to avoid underutilization of the resources in order to prevent the overhead of costs due to unnecessary computing power and communication bandwidth, it is well-advised to have some reserve resources in the system where new functionalities can be implemented.

To solve this problem, the design space has to be explored with a strategy which at least converges toward a solution close to one which yields the optimal solution. Actually, AES architectures are designed based on the designer's inspiration. AES system designers are highly-experienced engineers that mostly act as system integrators, i.e. they order devices from components suppliers and plug them on the rest of the system. As they actually do this successfully, one can argue that automatic partitioning tools might not perform competitively with humans. This assertion is perhaps reasonable when the input set of components and the partitioning patterns are reduced and easy to trace and to manage. But like most real-life problems, the design of AES architectures involves the manipulation of very large sets of items with a large range of partitioning criteria that are not always easy for a human intelligence to manage optimally. The level of expertise needed by humans is very expensive as long as system architects are rather artists than engineers. A system

architect needs to invest a very long time to acquire this level of expertise and then it is very adventurous to replace him. Furthermore, the solution space is often so large that humans will never get time to scan it in order to pick out the best solution. With an automatic partitioning, we provide a powerful engineering tool for the design of AES architectures, helping this activity to make the transition from its actual status of art to an engineering discipline.

In this work, we are concerned with the mapping. A mapping and definitely a partitioning can be static or dynamic. In static mapping, the elements of the functional specification are assigned to the platform elements prior to the execution and are not changed until the execution finishes. Dynamic mapping is needed when the functional specification changes dynamically during the running time and needs to be redistributed. Based on the input functional specification and the resource platform, static mappings can be classified following two taxonomies: The first classification is based on the specialization of the functional specification and/or of the platform. Following this classification, we can distinguish the mapping of specialized components on specialized platforms, specialized components on arbitrary platforms, arbitrary components on specialized platforms and arbitrary components on arbitrary platforms. The second classification is based on the uniformity of the attributes of the elements of the functional specification (e.g. the attributes of the components and the weights of the connections) and the characteristics of the elements of the hardware platform. Following this classification, we can distinguish the mapping of uniform functional specifications on uniform platforms, non-uniform functional specifications on uniform platforms, uniform functional specifications on non-uniform platforms and non-uniform functional models on non-uniform platforms.

In our case, the partitioning problem is how to map $CDFM$ models onto multi-device architectures of heterogeneous multi-processor hardware platform devices. Considering the $CDFM$, this problem is a graph partitioning problem. However, even though this work is focused on a specialized domain, i.e. the automotive domain, the problem we address here is a static mapping of a non-uniform graph of arbitrary components on an arbitrary non-uniform multi-device platform architecture, since the mapping is done once during the system conception and remains unchanged for the whole life of the resulting system. Furthermore, a particular architecture might be used for years on a wide range of product series. The functional components contained in the nodes of the $CFDM$ models do not underlie neither a specialization nor a granularity constraint. They are just required to be atomic components. The communication data described in the tokens of the $CDFM$ models might be of every possible nature, i.e. information, computation value, control directive, power supply, etc. The target platform is also unconstrained on the type and the capacity of its elements.

9.1.2 Requirements for the partitioning algorithm

Because of the wide range of its application fields, the partitioning of graphs have been widely investigated. The most active research communities applying graph partitioning for the last decade include the microelectronic design (e.g. for SW/HW co-design), the system analysis domain (e.g. for pattern- and similarity-based classification), the network engineering and the business management (to solve optimization problems). These communities have different terminologies, different patterns for the partitioning and make different compromises for the partitioning process. But commonly, the aim of the partitioning is to achieve a clustering of a given set of objects following some similarity patterns. As the overall goal of a partitioning algorithm is to find a global optimal solution while performing as little computation as possible, it is necessary to identify the most suitable clustering techniques when solving a partitioning problem. A vast collection of clustering algorithms is available in the literature. Near general-purpose partitioning algorithms, a full range of specific-purpose algorithms have been designed. General-purpose algorithms propose generic solutions while specific-purpose algorithms propose more effective solutions, each however optimized for a specific problem. Unfortunately, we are not aware of an algorithm for the partitioning of automotive systems functional specifications. Moreover, because clustering algorithms often

contain implicit assumptions about the cluster shapes, the measurement of the similarities and the grouping criteria, there is no clustering algorithm that is universally applicable on all input structures or with all possible clustering patterns. Thus, there is no competent perspective enabling someone to select one of the existing algorithms that is suitable for the problem at hand.

The usefulness of a partitioning algorithm is determined by its ability to match with the particular problem under resolution and its complexity (in time and eventually in space). Matching the problem includes the support of the format of the input specification with the involved artifacts (i.e. abstraction level, granularity), the capacity to accommodate the considered clustering patterns and the organization of the clustering objects as well as the ability to produce good partitions. The latter involves the manner in which the clusters are formed and the sensitivity of the clustering technique to changes that do not affect the intrinsic structure of the input objects, e.g. changes in the order of the input objects, or on the formulation of the clustering criteria. For our purpose, a partitioning algorithm must preserve the functional structure of the system (i.e. conservation of the connectivity within the input specification), allow user interaction (i.e. the system architect must be able to impose the composition of the clusters at every time in the partitioning process) and afford the flexibility (i.e. the ability to rapidly integrate the changes occurring on the functional model, the platform or the design constraints). To achieve good results, such an algorithm shall certainly enjoy a global view on the system specification that enables the comparison of the closeness between surrounding objects. As our partitioning is done for a static mapping of functional specifications on quite similar platforms of a wide range of vehicle series, the time and the space complexity of the algorithm is not very critical. However, the algorithm must be able to find the best result in the best possible time.

9.2 Partitioning methods

9.2.1 Exact and heuristic methods

A partitioning algorithm might make use of either exact or heuristic methods (figure 9.1). There are two broad approaches for exact methods. The first approach follows a systematic enumeration of all solutions. This approach is not practicable when the solution space is very large like in the present case. The second approach makes use of linear programming methods. Linear programing is an important modeling tool in operations research. It has been widely used in business management to minimize the cost of production systems, in transportation to optimize the traffic flow, in telecommunication for routing problems, in microelectronics to optimize the cost of embedded HW/SW systems, in the embedded software engineering for scheduling and allocation problems and in a lot of other engineering and management fields. If there exist some points in the solution space where the objective function has the largest value, linear programming methods guarantee that at least one of them will be found. The description of a problem through linear programs consists of a linear function $C^T x$, e.g. the objective function, that is to be maximized, the problem constraints in the form $Ax \leq B$ and a set of non-negative variables $x \geq 0$, where x is a vector of variables (representing for example the components of the cost function), C (representing for example the weights of the components of the cost function) and B (the constraints) are vectors of coefficients and A is a matrix of coefficients, e.g. a distance matrix. A linear program is usually expressed in matrix form and then becomes: Maximize $C^T x$ subject to $Ax \leq B$ and $x \geq 0$. The most usual form of the linear programming method known in the partitioning of embedded systems is based on Integer Linear Programs (ILP), e.g. [105]. ILPs can be exactly solved by branch-and-bound algorithms. But the problems with ILPs include the difficulty to solve the equations, the cost in time (exponential time-complexity) of ILP solutions and the difficulty to model non-linear constraints. Because of the complexity of the problem we are solving, even if we could find an exact solution, linear programming would not be practicable due to the expensive computation time it would need (i.e. exponential runtime), the difficulty to model the constraints as mathematical propositions and the loss of fidelity with the problem definition that will be

induced by the unavoidable assumptions. As the partitioning problems are generally NP-Complete [34, 54, 106], heuristic algorithms are the best way to find good solutions in a reasonable amount of time.

Heuristics attempt at finding optimal or nearly optimal solutions whereas no guarantee is given on the optimality of the solution found. However, approximative methods, i.e. that find solutions whose proximity to the optimum is provable, can be applied to complete heuristic methods. Furthermore, problem-specific heuristics can use information about the problem both to accelerate the partitioning and to approach the optimal solution. Following the approach used, we can distinguish two groups of heuristics: Those using constructive techniques and those using partitional techniques (e.g. iterative improvement techniques). Constructive partitioning techniques are based on clustering techniques. Given a set of objects and a partitioning problem on these objects, constructive partitioning techniques process by stepwise grouping these objects in order to achieve a complete partition while partitional techniques determine all the clusters at once. Partitional techniques usually process by relocating (iteratively) the objects across the clusters to optimize an objective function defined either locally (on a subset of the objects) or globally (over all of the objects). Partitional methods have advantages in applications involving large sets of objects for which the construction of a dendrogram, i.e. a tree showing a sequence of clusterings where each clustering is a partition of the set of objects, is computationally prohibitive. But in practice, partitional algorithms are generally used to refine an existing partition. As they do this iteratively, they are also called iterative improvement algorithms. Nevertheless, some hybrid algorithms exploiting the positive features of both the constructive and the partitional methods have been presented [102] but the results are obviously very poor, particularly concerning the time complexity of the algorithms [73]. Most partitioning problems are solved using a combination of constructive and iterative algorithms whereas constructive algorithms are used to generate the start partition that will be refined iteratively. We also proceed in this way, i.e. we first construct a partition and then we use an iterative improvement algorithm to ameliorate the quality of this partition.

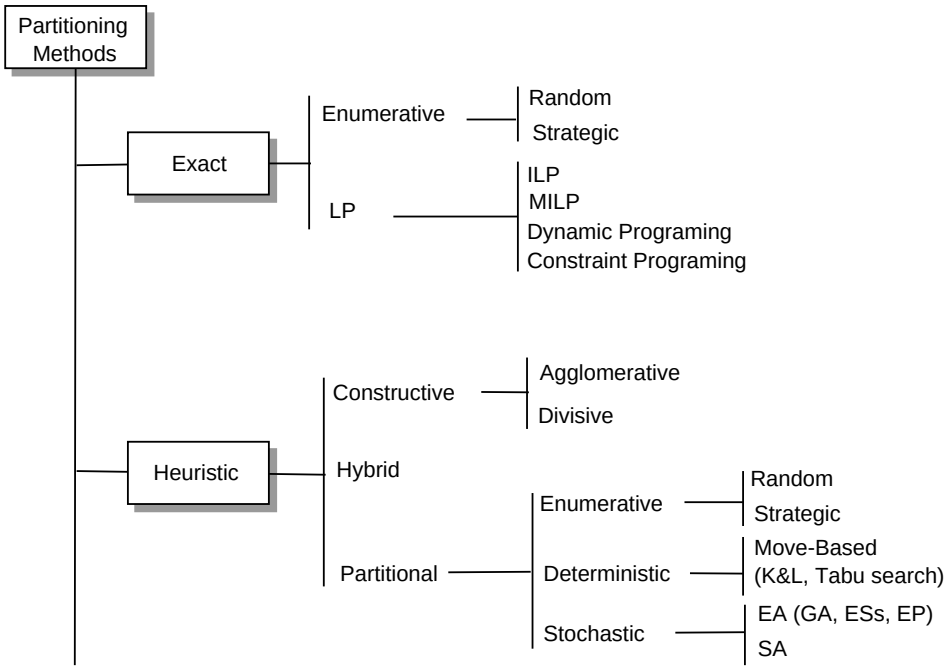


Figure 9.1: Taxonomy of the partitioning methods

9.2.2 Constructive partitioning techniques

The construction of a partition can be done by assigning randomly each object to a part of the partition (random assignment) or by clustering the input objects using closeness (or distance) measures. Because random assignment methods are easy to implement and economical in computation time, they are used essentially to build the start partitions for improvement algorithms, i.e. partitions that will be refined. But, as the performance of the refinement procedures depend on the quality of the start partition, inputting randomly constructed partitions can significantly damage the quality of the end-partition as well as the performance of the partitioning procedure itself. In fact, random assignment methods are characterized by the unpredictable quality of their solutions. Furthermore, they can completely destroy the connectivity of the system specification. Because of these shortcomings and the fact that they do not guarantee any quality for the result, random assignment methods will not be considered in this work.

Given a system of closeness patterns on a set of objects, clustering algorithms can classify these objects following the values of their closeness measures to each other. Depending on the configuration of the problem to be solved through a clustering algorithm, we can process hierarchically either with an agglomerative (bottom-up) or with a divisive (top-down) clustering. Divisive partitioning algorithms follow a top-down splitting method. They start with all the objects in a single cluster and successively split this cluster into smaller ones as desired or until each object falls in one cluster. Divisive methods have been rarely applied to solve real-life partitioning problems. A divisive clustering algorithm is developed in [127]. It exploits the minimal spanning tree (MST) of the input set, i.e. the MST is constructed and the clusters are generated by deleting the edges with the largest lengths. Even though the time and space complexities of divisive algorithms are typically lower than those of agglomerative algorithms, a typical divisive algorithm works well only on object sets containing non-isotropic clusters, i.e. containing well-separated, chain-like or concentric clusters, whereas an agglomerative algorithm equally accommodates with object sets having isotropic distribution, i.e. with uniformly or orderless distributed objects. Consequently, divisive partitioning algorithms are inadequate for our problem since AES functional models generally have heterogeneous unordered distributed components.

Agglomerative clustering algorithms follow the opposite strategy of the divisive algorithms, i.e. they follow a bottom-up merging strategy. Agglomerative clustering algorithms build clusters by progressively merging the previously established clusters. They start with each object being a separate cluster itself, and successively merge them according to a closeness measurement until each object is in a cluster or any other desired configuration is reached. The steps of an agglomerative clustering algorithm can be roughly summarized as follows:

1. Consider each object as a cluster
2. REPEAT UNTIL a termination criterion is met
3. Merge the two closest clusters
4. Determine the new closeness values

Such an algorithm has been used in [116]. A problem accompanying the use of agglomerative clustering methods is the definition of the termination criterion. Defining the condition to stop the clustering in a way that the solution is optimal is a delicate issue that can easily turn to be a drawback for the algorithm. In fact, a combinatorial search of the set of all possible solutions to discover an optimum value of a termination criterion is clearly computationally extremely prohibitive and thus unattractive. Generally, although this issue is extremely relevant for the quality of the partition, its definition is generally left up to the user. Other weaknesses of agglomerative clustering methods include their poor time (at least $O\left(n^2 \log n\right)$) and their space complexity $O\left(n^2\right)$, where n is the total number of input objects. A good solution to these problems is provided if the number of the output clusters is known. If this is given, the problem resumes to a N-way clustering. The functioning of N-way clustering algorithms can be roughly outlined as follows:

1. Choose (or given) the number N of output clusters
2. Generate N clusters and determine their centers
3. REPEAT UNTIL each object is assigned to at least one of the N clusters
4. Assign each object to the nearest cluster
5. Compute the new closeness values between the clusters and the remaining objects

N-way partitioning algorithms are popular because they are easy to implement, and their time complexity can be reduced to $O(n)$ where n is the number of objects. In the definition of our problem, we avoid to suppose that we know the number of devices that will be installed in the system to execute the functionalities that are specified in a given FN model. Setting the number of devices a priori can lead either to computation surcharges in the devices with negative consequences on the inter-device communication or to underutilized devices. To stay close with reality, it will be rather assumed that we know the capacity of the available devices and we can determine the amount of resources that is reserved for future enhancements of the system's functionality. The objective is then to use the least possible number of devices so that the system functioning and the reserve resource (i.e. distributed on the devices) are guaranteed and the frames usage is minimal.

If the number of output clusters is not given within the definition of the problem, a clustering approach that is based on the quality threshold (QT) can be used as demonstrated in [98]. The QT approach can be roughly summarized as follows:

1. Choose the maximum size and a candidate object for a cluster
2. Build the cluster by including the closest object, the next closest object, and so on, until the size of the cluster meets the threshold
3. Save the candidate cluster and repeat with the remaining objects

The main advantages of these algorithms include the ability to divide the set of the objects into a large number of partitions without augmenting the runtime. This corresponds to the objective of the partitioning of AES as we stated above. Obviously QT algorithms are adequate for solving our problem. We have developed a partitioning algorithm based on the QT scheme to build a partition of $CDFM$ models. Unfortunately, the quality of the partitions built with QT algorithms is not always good. For illustration, when partitioning the functional specification of a system, the last parts tend to be disconnected. QT algorithms also tend to be sensitive to the choice of the beginning cluster but since they are quite fast, one can run the algorithm with different starting points and simply choose the best result. However, these problems are due to the fact that QT algorithms, like all the preceding agglomerative algorithms, have a local view on the system. A clustering method that will take a global view on the closeness landscape of all the components of the system will obviously produce better results. An important class of clustering techniques that take advantage from a global view on the set of clustering objects use spectral methods.

The spectral approach for solving the partitioning problem uses global information about the set of objects that must be clustered by analyzing the spectrum of a matrix representing the relationships between these objects before making assignment decisions. Spectral partitioning methods are based on the use of eigenvectors of a graph adjacency, laplacian, or any other matrix representation which captures the properties of the graph to construct a geometric representation of its nodes (e.g., linear ordering) which are subsequently (heuristically) partitioned. The general flow of a spectral partitioner is as follows:

1. Compute $A =$ Adjacency matrix and $D =$ Degree matrix of the graph G
2. Compute the second smallest eigenvalue $\lambda(L)$ of $L = D - A$, where L is the Laplacian of G
3. Compute x, the real valued eigenvector associated with $\lambda(L)$

4. Map x into a heuristic partition of G

Spectral partitioning methods might potentially give a qualitatively better result than standard agglomerative clustering algorithms [20] which rely on local information (like the one we used before). Further advantages of spectral partitioning methods over agglomerative methods include their stability and their scalability. The stability of spectral partitioning methods is provided by their stability, i.e. their performance is predictable, as there is no need of resorting to multiple solutions derived from random starting points. The scalability is provided as the quality of the solutions found with spectral partitioning methods is maintained as the problem size increases. This is especially important for large graphs as sub-optimality of iterative improvement-based heuristics can affect the partitioning adversely. Spectral methods have been proved to be extremely useful for computing good minimum and ratio cuts of graph bi-partitioning problems [64]. Other successful applications of spectral methods include the geometrical as well as the temporal placement of computation modules on hardware layouts of FPGAs [29]. But these problems are very different from the one we are solving here:

- Firstly, because of the real formulation of the partitioning problem. While most of the known approaches have applied spectral methods on two-way partitioning problems, we are dealing with a multi-cluster partitioning problem. The multi-cluster partitioning, as needed for example for the placement of modules on FPGAs, is solved by means of hierarchical bi-partitioning with the objective of obtaining sized-balanced parts. Load balancing is not in our objective.

- Secondly, because of the input model, i.e. while the direction of the communication between two nodes of the graph is of high relevance for us, these problems totally mask the direction in which the data is communicated on an edge.

- Thirdly, because of the different natures of the objective functions, i.e. number of messages vs. mostly the number of edges in the cut set. In most applications, the objective is to minimize the number of cut edges combined with a given cluster size balance requirement or when the graph is weighted, the problem is formulated as to minimize the sum of the weights of the edges in the cut.

We aim in contrast at minimizing a particular packing of the objects in the cut whereas the clusters might be of very different sizes. However, spectral partitioning algorithms also can underlie some drawbacks the heaviest of which is e.g. their excessive computation needs in terms of runtime. Nevertheless, it is our interest to make a profit from the advantages of spectral methods if there is a relationship between the spectral properties and the partitioning properties of $CDFM$ models. Hoping to obtain high quality, stable and scalable partitioning solutions, we have investigated the ability of spectral methods for computing good solutions in our context and then we applied spectral methods for the clustering of $CDFM$ models as another constructive partitioning algorithm.

In conclusion, clustering algorithms are supposed to optimize an objective function, but this is very difficult for certain reasons including the fact that they do not support back-tracking, i.e. they generally never undo an assignment decision. The reason for this is obviously the overhead in computation cost that will occur on the algorithm. In some cases, the objective function cannot be precisely apprehended by a clustering algorithm. Iterative improvement algorithms are then used to improve the quality of the partitions built with clustering algorithms.

9.2.3 Iterative improvement techniques

Iterative refinement algorithms process by successive transformations of an existing partition. They begin with a possible solution, i.e. a complete partition. This initial solution is iteratively modified and evaluated with the objective function whereas each step of the partitioning produces a valid partition. An iterative improvement partitioning algorithm can be lightly formulated like a procedure which inputs a partition P and returns a partition P' such that: $Cost(P') \geq Cost(P)$. To find

the next best solution, iterative improvement partitioning algorithms perform different variants of neighborhood search. The neighborhood of a solution s is a set of solutions that can be reached from s by a simple operation (e.g. *move*). A neighborhood search procedure operates by recursively comparing the actual solution with the surrounding solutions and then sets the best solution as the actual solution. The general flow of a neighborhood search procedure of an iterative improvement algorithm can be summarized as follows:

1. Start with Current-Solution = Initial-Solution
2. Until Current-Solution = Goal-Solution OR there is no change in Current-Solution
3. Use the evaluation function to assign a value to each neighbor of the current solution
4. If one of the neighbors has a better value than the current Current-Solution then set this neighbor to be the new Current-Solution

Iterative improvement partitioning algorithms can be classified following the search technique that they use. Some use deterministic while others use non-deterministic search techniques. Deterministic search techniques guarantee an optimal partition by performing the exhaustive enumeration of the solution space. A deterministic search technique is very critical for the performance of the related algorithm. A non-deterministic search technique generates a near-optimal partition but reasonably quickly. Some non-deterministic search techniques can guarantee asymptotic convergence to the optimal partition. Very popular examples of iterative improvement techniques based on non-deterministic search techniques include the Kernighan & Lin (K&L) algorithm, the simulated annealing (SA) and the evolutionary algorithms (EA). Evolutionary approaches and simulated annealing use stochastic search while the K&L uses more goal-oriented, strategic search methods. In the strict meaning of the term, only K&L processes an iterative improvement. SA and EA temporarily accept solutions that are not better than the actual solution in order to escape from local minima. This feature is called hill-climbing.

Evolutionary approaches follow the Darwin theory of natural evolution based on "the survival of the fittest" to find the globally optimal partition. They operate by applying evolutionary operators on a given population of candidate solutions that are coded as chromosomes. A fitness function determines the capability of the chromosomes to survive into the next generation. Each evolutionary operator transforms one or more input chromosomes (called parents) into one or more output chromosomes (called offspring). The most commonly used evolutionary operators are the selection, the recombination and the mutation. An evolutionary algorithm proceeds as follows:

1. Choose a random population of solutions, i.e. of complete partitions
2. Assign a fitness value to each solution
3. REPEAT UNTIL some termination condition is satisfied
4. Use the evolutionary operator to generate the population of the next generation of solutions
5. Evaluate the fitness values of these solutions

Between the algorithms making use of evolutionary techniques, genetic algorithms (GA) have been the most used in clustering problems [59, 70, 71]. In GAs, the space of all the possible solutions (chromosomes) is mapped onto a set of finite strings over a finite alphabet and hence each solution is represented by an array or a string of values like in genetics. Generally the solutions are represented by binary digit-coded strings. The algorithm starts with an initial population of solutions that will evolve over generations. A probabilistic selection operator is used to identify the fittest solutions, i.e. those that will be promoted for the next generation. The selection probability (i.e. the fitness of a solution) is set proportional to the selection representation scheme, e.g. the dimension of the space in the roulette wheel. In GAs, the most used recombination operator is the

crossover. A crossover inputs a pair of chromosomes (parents), exchanges genes between them and generates a new pair of chromosomes (called children, descendants or offspring). A mutation operator that enables small random perturbations on a given single solution is used in a way that makes sure that all the parts of the search space will be explored. Doing so, the recombination (i.e. the crossover) and the mutation operators provide hill-climbing behavior to GAs, since they can produce new solutions that are completely different from the current ones. Thus in contrast with the clustering methods that perform a localized search, i.e. the solution obtained at the next iteration of the procedure is in the vicinity of the current solution, GAs perform a globalized search for solutions. Some partitioning solutions using GAs have been presented in the embedded system domain, e.g. [18,111] for min-cut partitioning and [31] for ratio-cut bisection problems. Designing the selection scheme, the crossover and the mutation operators of a GA is critical for its quality. Random selection schemes can dramatically disconnect the graph, e.g. the roulette wheel that is commonly used in GAs to select the parent solutions. Thereto, it is obvious that GAs are extremely sensitive to the selection of the parameters of the algorithms (size of the population, crossover and mutation probabilities, etc.). Furthermore, as GAs do not support the direct representation of the attributes of the solutions as real-valued vectors, the real values of these attributes will be discretized by the commonly used binary coding and this may disrupt the search space because of the non-linear structure of binary codes.

Simulated annealing [87] is a well-known optimization technique that emulates the physical annealing process. It is used as an alternative to greedy approaches which are easily trapped in local optima or to recover from solutions which correspond to local optima of the cost function. Given the initial solution, a SA algorithm randomly picks a neighbor solution and moves to it for the next iteration if it is better than the current solution. If not, the SA algorithm can accept the new solution for the next iteration with a probability $e^{\frac{-\Delta}{T}}$ (i.e. the Bolzmann acceptance rule). Otherwise it will retain the current solution and continue the search. The probability of acceptance is governed by the parameter T called temperature (in analogy with the annealing in metals) which is varied to guide the optimization, i.e. the exploration of the solution space. For SA to work, the starting temperature (for the first iteration), the final temperature, the cooling schedule as well as the cost function must be defined. Δ is the cost of the new solution minus the cost of the current solution. The cooling schedule states the interval by which T varies. The general scheme of SA is as follows:

1. Select an initial partition P_0 and select the values for the control parameters, i.e. the initial and the final temperatures T_0 and T_f
2. REPEAT UNTIL $T_0 \leq T_f$ or the optimal solution is found
3. REPEAT this step for a fixed number of iterations
4. Select randomly a neighbor P_1 of P_0 and compute Δ; If $\Delta \leq 0$ then replace P_0 with P_1 with a temperature-dependent probability $e^{\frac{-\Delta}{T}}$; Else replace P_0 with P_1
5. Reduce the value of T_0, i.e. $T_0 = cT_0$, where c is a predetermined constant (sometimes the number of iterations of the preceding step)

There are some applications of SA in the embedded system design, particularly for HW/SW partitioning [42, 43]. In theories [65, 91], SA can always find the global optimal solution [17]. But the definition of the control parameters of the algorithm, i.e. the start and the final values for the temperature and the temperature (cooling) schedule, is very critical for the performance of SA. To have the chance to reach the optimal solution, the temperature must be decreased very slowly from an iteration to the following one and the start and final temperatures must be optimally chosen. Like GAs, SAs are very expensive and make use of random choices of the candidate solutions. It is reported in [78] that these approaches can be competitive on uniform and geometric random graphs. But multiple runs of K&L may be preferable for sparse graphs that have a local structure. $CDFM$ models are such graphs.

According to [97], K&L-based improvement heuristics are the main robust methods for graph partitioning that generate satisfactory results. Presented by Kernighan and Lin [86] to solve graph bi-partitioning problems, the K&L algorithm was extended in [49] to run in linear time and has been continuously improved to enhance the performance both in terms of the quality of the results and in terms of its complexity. The K&L-algorithm relies on the definition of the cut of a graph bisection as well as the notion of the gain of moving a vertex from one side of the bisection to the other side. For this reason, the K&L algorithm is also called group migration or min-cut algorithm. Given a graph, the cut is defined as the sum of all the edges crossing between the parts. By moving a node from one part into the other, the number of crossing edges is modified and the value of the cut changes. If the gain of moving a vertex is positive, then operating that move will reduce the total cost of the cut of the partition. The K&L-algorithm allows a series of moves which reduce the cut. The best advantage of the K&L heuristic is its simple but yet powerful control strategy, which can overcome many local minima without using excessive moves. However, K&L algorithms are sensitive to the selection of the initial partition and they may converge to a local minimum of the cost function if the initial partition is not properly chosen. In order to maintain the sizes of the parts, two nodes are swapped rather than moving a single node. During one iteration of the K&L algorithm, the nodes that are moved from one side of the bisection are locked on the other side. The cost of all possible swaps of unlocked nodes is computed and the nodes with the best gain (greatest decrease or less increase of the cut) are swapped. If all the nodes are locked, the current partition is set to be the best partition if it improves the cost of the cut. One iteration of the K&L-algorithm is called a pass. After one pass, all the nodes are unlocked and a new pass is executed. The iteration terminates if a pass produces no further improvement on the cut. The basic K&L algorithm can be summarized as follows:

Given an edge-weighted graph $G(V,E)$ and a partition $G = A \cup B$ with $|A| = |B|$, the goal is to minimize the cut set of the partition

1. REPEAT UNTIL there is no more improvement on the cost of the cut

2. Find equal-sized subsets X in A and Y in B, such that swapping X with Y reduces the total cost of edges between A and B

3. If the cost of the new partition is less than the cost of the old one, then consider the new partition as the actual partition and restart

The following definitions are used: Let $E(a)$ be the external cost of the graph's node a and $I(a)$ the internal cost of the node a. $E(a)$ is the part of the total cost of the partition that is due to the fact that a is in the part that actually contains it. $I(a)$ is the cost that would arise if a was moved into the other part. Consider $D(a) = E(a) - I(a)$ the cost difference of a. $D(a)$ is the difference that would occur on the cost of the partition if a is moved.

The gain of switching two components between two parts is:

$Gain(a,b) = D(a) + D(b) - 2 * \omega(a,b)$;

$Gain(a,b)$ measures the improvement in the cost of the partition that would result from swapping a and b.

If a and b are swept, then the new cost will be:

$NewCost = Cost - Gain(a,b)$ and the values of D will become:

$New\_D(a') = D(a') + 2 * \omega(a',a) - 2 * \omega(a',b)$ for all a' in A, $a \neq a'$

$New\_D(b') = D(b') + 2 * \omega(b',b) - 2 * \omega(b',a)$ for all b' in B, $b \neq b'$

A good adaptation of the K&L algorithm, i.e. one with judiciously defined move operation, well-adapted definition of the gain and of the cost difference related with the nodes moving will certainly be the solution to find the best output to our problem.

9.3 Conclusion

The partitioning will be done by building a partition that will be refined. Although QT algorithms do not guarantee the best partition, they are good enough for building the start partition as the number of output clusters is not given. In this context, a QT-based partitioning algorithm is the best solution as it enables the flexible determination of the number of clusters. Otherwise, spectral methods that take advantage from their global view on the system and enable the conservation of the system connectivity can provide better start partitions. But they are computationally too expensive for our problem and furthermore, as spectral methods are not really well-featured for n-way partitioning problems by which the resulting clusters might have very different sizes, the effort of using them will obviously not be profitable for solving our problem.

EA, SA and K&L are the candidate solutions for the improvement of a partition. EA and SA randomly search the solution within the solution space and try to refine it stochastically. This approach supposes that the number of clusters is known beforehand. This is unfortunately not the case in the problem we are solving. However, although the most implementations of the K&L are applied on bi-partitioning problems, we think that the K&L algorithm provides a supportable starting point for developing an improvement algorithm for our problem. This necessitates a good adaptation of the move operations of the K&L to the context of our problem. But the major problem with the K&L algorithm, i.e. its sensitivity to the initial partition, will remain a challenging issue that must also be managed. In fact, the K&L algorithm may converge to a local minimum of the cost function if the initial partition is not properly chosen. It is thus important both to find the best initial partition and to design the adequate K&L algorithm for its improvement. The first of these two issues is the purpose of the next chapter. The improvement algorithm will be defined in chapter 11.

Chapter 10

The partitioning algorithms

In this chapter we present the partitioning algorithms for solving the problem defined in this work. We firstly specify our partitioning strategy. Then, we describe the algorithms for the pre-clustering and the clustering. To make the algorithms understandable, we profoundly discuss the conditions ruling the pre-clustering as well as the definition of the closeness metrics and the choice of the closeness function. The clustering is done by a QT algorithm that supports the relationships that are induced in the input models by the constraints and the strategic orientations of the design.

10.1 The partitioning strategy

10.1.1 A three-step process

Given a *CDFM* model *G*, we solve the partitioning problem in three steps as follows:

1. We first merge the nodes of *G* that *need* each other and those whose separation would obviously not produce a good solution. This is the pre-clustering.
2. Based on the result of the pre-clustering, we build the clusters that represent the logical devices, i.e. that will be implemented as the platform devices.
3. The third step applies an iterative improvement by means of a K&L algorithm on the result of the second step.

Figure 10.1 illustrates the three-step partitioning process.

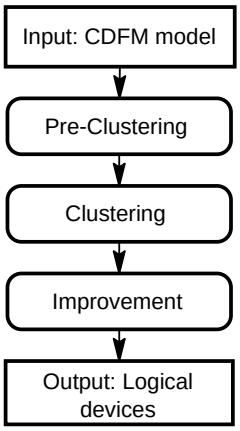


Figure 10.1: The partitioning process

The first step of the partitioning operates a pre-clustering of the nodes of the input model. During this operation, we merge the nodes of the input model that must run on the same device.

To do that, we solve the relationships that induce a dependency between the nodes of the input *CDFM* model. These relationships are specified by means of *needs* and *excludes* relations between the nodes and between the tokens of the input model. As *needs* and *excludes* relations are formally defined, they can be treated automatically as well as manually. The result of this operation is a graph of atomic modules (called super-modules) that will be assigned each unbroken to a device of the system and a set of messages. At the end of this operation, there are no further *needs* relations in the graph of super-modules.

The second step of the partitioning is a constraints-oriented clustering. We operate a bottom-up clustering on the result of the first step of the partitioning. In this operation, we progressively merge the modules that are related according to the strength of their relation. The result of this step of the partitioning is a graph of clusters, each representing a logical device of the system. As we assume that the number of devices that will be implemented on the platform is not pre-defined, we have two possibilities to terminate the clustering: Firstly, the size of each device (upper bound capacity) is known. Secondly, the maximum number of messages allocated to a device is known. These are the most probable constellations of the architectural design of AES.

The third step of the partitioning deals with the improvement of the result of the second step. Two super modules that are assigned to different devices will communicate by passing signals through the bus. A frame can contain several signals produced by several super modules that are assigned to the same device (cf. frames multiplexing in section 11.1.4). At this step of the partitioning process, we assign the signals to the frames, then we optimize progressively these assignments. Here, we can redefine the partitioning problem as follows:

Problem formulation 10.1 (Improvement)**.** : Assuming that the functioning of the system is guaranteed by its actual configuration under the given constraints, what is the gain in terms of frames economy if we change this architecture?

10.1.2 Definitions of terms

We now introduce the following terms that will be used to describe the partitioning.

Definition 10.1 (Configuration)**.** At each time in the partitioning process, the configuration of a *CDFM* model is the actual composition of its nodes and the characteristics of its edges. A configuration is changed by a clustering operation.

Definition 10.2 (Partition)**.** A partition P of a *CDFM* model is a configuration $\{P_1, P_2, \ldots, P_n\}$ of the given *CDFM* model where each cluster P_i represents the functionality of a device. A partition is a graph of clusters.

Definition 10.3 (Part)**.** Each node P_i of a partition P is called a part of P. A part is a cluster of components that represents a logical device. Per analogy with the real system, we may also say part of the system or device.

Definition 10.4 (Clustering)**.** The clustering is the process of building hierarchical modules (i.e. clusters) from the nodes of a *CDFM* model.

Definition 10.5 (Partitioning)**.** The partitioning is the process of building a partition, i.e. the process of grouping the system's components (i.e. the nodes of a *CDFM* model) to build parts.

Definition 10.6 (Membership)**.** A module m_1 is a member of a bigger module m_2 if m_1 is embedded in m_2, i.e. the functionality of m_1 is a part of the functionality of m_2.
The notation m_2+m_1 means that m_2 and m_1 become members of the same cluster.
The notation m_2-m_1 means that m_1 looses its membership in m_2.
A token T_{ij}^k is a member of a message M_i^q if the significance of M_i^q looses its plenitude when T_{ij}^k is extracted from M_i^q.

Definition 10.7 (Sub-module)**.** A module m_1 is a sub-module of a module m_2 if m_1 is a member of m_2.

10.1.3 The main procedure

The main procedure described in algorithm 1 implements the three-step partitioning process. It inputs a *CDFM* model G and outputs the final configuration of the logical architecture of the system, i.e. the best partition.

Algorithm 1 PROCEDURE Partitioning(G: CDFM)

1: SuperModules:= Pre-Clustering(G); *//* Builds the graph of super-modules by solving needs and excludes relationships*
2: InitialConfig:= Clustering(SuperModules); *//* Partitions the graph of super-modules and returns the initial configuration of the system*
3: FinalConfig:= Improvement(InitialConfig); *//* Improves the initial configuration of the system and returns the final configuration, i.e. the best partition*
4: **return** FinalConfig; *//* Final definition of the logical devices and the frames*

10.2 The pre-clustering

10.2.1 Definition

The procedure "Pre-clustering" must find the pairs of nodes (v_i, v_j) of the input *CDFM* model G that must be merged and build the super-modules with them. We first introduce the following rules to solving the *needs* and the *excludes* relations.

Definition 10.8 (*needs* and *excludes*)**.** For each two nodes v_i and v_j, the notations "v_i needs v_j" and "v_i excludes v_j" mean that there is a needs resp. an excludes relation between v_i and v_j while $needs(v_i, v_j)$ and $excludes(v_i, v_j)$ mean that the two nodes can effectively be clutched resp. must be separated. Thus for each two nodes v_i and v_j of a *CDFM* model,

- IF $needs(v_i, v_j)$ THEN v_i and v_j must run on the same device and thus must be merged to form a super-module.
- IF $excludes(v_i, v_j)$ THEN v_i and v_j may never run on the same device.

All the relations guiding the pre-clustering can be specified in terms of *needs* and *excludes* relations between the nodes of G. The pre-clustering thus finally aims at merging the *needs*-related components. *Excludes* relations will be treated later in the partitioning process, always when necessary. The following properties of the *needs* and *excludes* relationships state the rules that guide the pre-clustering. Note that these properties are identical for both the nodes and the tokens.

The *needs* relation is:

- reflexive, i.e. for each v_i, v_i needs v_i AND $needs(v_i, v_i)$.
- transitive, i.e. IF $needs(v_i, v_j)$ AND $needs(v_j, v_k)$ THEN $needs(v_i, v_k)$.
  Similarly, IF v_i needs v_j AND v_j needs v_k THEN v_i needs v_k.
- not symmetric (per agreement), i.e. IF $needs(v_i, v_j)$ THEN not obligatorily $needs(v_j, v_i)$.
  Similarly, IF v_i needs v_j THEN not obligatory v_j needs v_i

and the *excludes* relation is:

- not reflexive, i.e. for each v_i, $\neg(v_i$ excludes $v_j)$ AND $\neg excludes(v_i, v_i)$.
- not transitive, i.e. IF $excludes(v_i, v_j)$ AND $excludes(v_j, v_k)$ THEN not obligatorily $excludes(v_i, v_k)$.
  Similarly, IF v_i excludes v_j AND v_j excludes v_k THEN not necessarily v_i excludes v_k.

- but symmetric (per definition), i.e. IF *excludes*(v_i, v_j) THEN *excludes*(v_j, v_i).
  Also, IF v_i excludes v_j THEN v_j excludes v_i.

The *needs* and *excludes* relations between the tokens induce some *needs* and *excludes* relationships between the nodes producing these tokens. Given two tokens T_{il}^k and $T_{jm}^n \in \Omega$,

- IF T_{il}^k needs T_{jm}^n THEN v_i needs v_j
- But IF T_{il}^k excludes T_{jm}^n THEN v_i and v_j don't care.

However, when solving the *needs* and *excludes* relations, some conflicts can arise, for example, when a single node is involved in several *needs* and *excludes* relationships or when the relationships between the nodes are conflicting with the relations between the tokens. In the following, we examine these conflicting situations. The *needs* relations are intended to augment the quality of the partition (i.e. performance and cost). Passing over a *needs* relation can reduce the chance to find the optimal system architecture, but it will not be dramatic for the functioning of the system. In contrast, ignoring an *excludes* relation can cause serious faults in the system's functioning. Since our primary objective is to build well-functioning systems, we agree that the *excludes* relationship is stronger than the *needs* relationship, i.e. for each two nodes v_i and v_j of G, given two tokens T_{il}^k and $T_{jm}^n \in \Omega$,

- IF v_i needs v_j AND v_i excludes v_j THEN *excludes*(v_i, v_j).
- IF v_i needs v_j AND v_j excludes v_i THEN *excludes*(v_i, v_j).
- IF a sub-module of v_i excludes a sub-module of v_j THEN *excludes*(v_i, v_j) (and consequently *excludes*(v_j, v_i)).
- IF *needs*(T_{il}^k, T_{jm}^n) THEN v_i needs v_j AND (T_{il}^k and T_{jm}^n) build a message.
- But IF *excludes*(T_{il}^k, T_{jm}^n) THEN v_i and v_j don't care.

Proposition 10.1 (*needs*)**.** *needs*(v_i, v_j) == TRUE if and only if there is at least one sub-module of v_i that needs a sub-module of v_j AND there is no sub-module of v_i that excludes a sub-module of v_j AND no sub-module of v_j excludes a sub-module of v_i.

10.2.2 The pre-clustering algorithm

The procedure "Pre-Clustering" finally runs the following algorithm:

In procedure 2, the instruction *"Actualize the edges connecting the new SuperModules"* determines the connections of the new super modules, i.e. the edges connecting a new super module with any other node and the corresponding weights (see definition 10.11 and figure 10.2). The following definitions specify the process of actualizing the edges of the graph of super modules.

Definition 10.9 (Neighborhood of a node)**.** : Given a node v_i of G, the neighborhood of v_i, called Neighborhood(v_i) is the set of the nodes that are directly related with v_i, i.e. Neighborhood(v_i) = set of the nodes v_j, so that there is an edge e_{ij} in G.

Definition 10.10 (Neighborhood of a super module)**.** : Given a super module v_i with sub-modules v_{i_k}, $k \in \mathbb{N}$, Neighborhood(v_i) = $\bigcup_k$ Neighborhood(v_{i_k})

Definition 10.11 (Weight of a super edge)**.** : If v_i is a super module with sub-modules v_{i_k}, then for each $v_j \in$ Neighborhood(v_i),

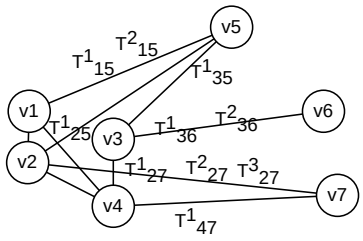
$$T_{ij} = \bigcup_k T_{i_k j} \text{ and } T_{ji} = \bigcup_k T_{j i_k}$$

$$\text{thus } e_{ij}.T = \bigcup_k e_{i_k j}.T$$

Algorithm 2 PROCEDURE Pre-Clustering(G: CDFM)

```
1: nbModules := |SetOfSuperModules|; //* SetOfSuperModules is the set of the nodes of G
2: k:= nbModules + 1; //* temporary index for the new modules
3: for all v_i, v_j in SetOfSuperModules do
4:    if needs(v_i, v_j) then
5:       v_k := v_i + v_j; //* merge them into a new hierarchical module v_k following property 10.1
6:       SetOfSuperModules:= (SetOfSuperModules - v_i - v_j) ∪ v_k;
7:       nbModules:= nbModules - 1;
8:       k:=k+1;
9:       Actualize the edges connecting the new SuperModules; //* Super-edges following definition
         10.11
10:   end if
11: end for
12: Actualize the indexes of the SuperModules; //* Allocate the indexes from 1 to nbModules
13: return SuperModules; //* Graph of super-modules with actualized edges
```

(a) Modules and edges



(b) A super module with the super-edges

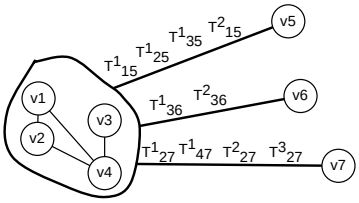


Figure 10.2: The super graph

10.3 The clustering

10.3.1 Closeness metrics

The procedure "Clustering" implements the second step of the partitioning. It clusters the nodes of the graph of super modules in order to build the logical devices. To do this, it merges closely related modules in order to implement them on the same device. Concretely, the problem to solve here is to find a clustering that minimizes the data exchange on the bus, i.e. the inter-clusters communication, and that at the same time meets the design constraints and the other optimality concerns such as maximizing the HW sharing, optimizing the grouping of the components following the strategic requirements like the level of safety, the mission- or the business-criticality, the source of procurement, the production issues, the potential level of reuse, the rhythm of changing, the target of the procured service, etc. For instance, two modules will be closely-related if there is no *excludes* relationship between them and they are related with each other in some way, for example if:

- they can share resources (i.e. HW and frames),
- they have common accesses,
- they communicate with each other,
- they underlie a relation given by the strategic orientation of the design

In order to determine the closeness between two modules, all the involved closeness factors must be quantified and combined with each other. The first step in solving this multi-objectives optimization problem is to formalize the objectives. This step may yield a set of closeness functions.

A closeness function is defined by a closeness metric that can measure and compare the strength of a given relation between two modules. The higher the closeness between two modules, the closer the two modules are related. The closeness metrics must thus reflect the closeness factors and the objectives of the design.

The resource sharing metric: The potential that two modules have to share a given hardware unit is often considered as a closeness factor in the design of embedded systems [116]. However, a satisfactory HW (processing unit, e.g. ASIC) sharing metric cannot be extracted from our input model because of its high resolution. Thereto, as the components of the HW devices of the platform are not allocated, it will be particularly difficult to analyze the potentiality of two components to share a given HW. Furthermore, the hardware sharing is obviously not useful in our case, since we are concerned with the mapping of the modules on the logical devices (i.e. Partitioning 1) rather than with the deployment of the tasks and processes (Partitioning 2). However, if known, the HW sharing can be considered as a strategic decision that is taken before the partitioning, specified in the input model, e.g. in the form of *needs*, and used in the pre-clustering to build the super modules. Therefore, the HW sharing metric will certainly not induce any substantial benefit in the quality of the partitioning (i.e. in the mapping).

The frames sharing metric: The frames sharing metric measures the potential of two modules to share frames. The likelihood to share frames is determined by the number of tokens that are produced in the same range of time by two modules. Modules that produce a high number of tokens within the same range of time are more likely to share frames. Thus, following the frames sharing metric, the higher the number of tokens produced within the same range of time, the closer the producing modules are related. If $v_i.dates$ is the set of ranges of time within which the module v_i produces its tokens, the frames sharing metric can be defined as:

$$FrameSharing(v_i, v_j) = |v_i.dates \cap v_j.dates| \tag{10.1}$$

The common accesses metric: Grouping two modules that have a great number of common accesses would probably improve the quality of the communication by reducing the number of frames needed for the inter-components communication. Assuming that each module is its own neighbor, the common neighborhood of two modules v_i and v_j defines the number of modules that access, i.e. exchange tokens with, both v_i and v_j. The common accesses metric is defined as:

$$CommonAccesses(v_i, v_j) = |Neighborhood(v_i) \cap Neighborhood(v_j)|$$

The communication metric: Grouping two modules that heavily communicate with each other will obviously increase the performance of the communication. The communication metric measures the magnitude of the communication between two modules. This is determined by the amount of data exchanged between the two modules. The simplest and most usual way to measure the communication between two modules is to count the number of bits exchanged between them, i.e.

$Communication(v_i, v_j) = \sum_k T_{ij}^k.res \times T_{ij}^k.freq + \sum_k T_{ji}^k.res \times T_{ji}^k.freq.$

In this case, the modules exchanging the highest number of bits are the closest. But, in our case, two components that are assigned to different devices will communicate by exchanging signals over the bus. The communication metric must measure the impact of the separation of two modules on the occupation of the bus. This depends on the number of frames that are sent on the bus and the length of each of them. Each short token (i.e. $resolution \leq maxframeL$) can fit into a frame. A long token will be fragmented into the corresponding number of signals that can fit each into a frame. Short frames occupy the bus differently, depending on their length. The following

example illustrates the expressiveness of this metric. Suppose four modules v_1, v_2, v_3 and v_4 of a *CDFM* model. Within an activation time of the system, v_1 exchanges a token of 20 bits 10 times with v_2 (interface e_{12}), a token of 100 bits 3 times with v_3 (interface e_{13}) and a token of 112 bits 3 times with v_4 (interface e_{14}). Considering the number of bits exchanged, $Communication(v_1,v_3) = 100 \times 3 = 300$ bits is greater than $Communication(v_1,v_2) = 20 \times 10 = 200$ bits, meaning that v_1 is closer to v_3 than to v_2. But considering the number of signals (assuming that $maxframeL =$ 8 Bytes), e_{12} would produce 10 signals while e_{13} produces $2 \times 3 = 6$ signals, meaning that v_1 is closer to v_2 than to v_3.

These two conclusions are contradictory. But the number of signals produced by an interface clearly better captures the impact of the merging or the separation of two modules on the bus occupation than the number of bits exchanged between them. Now, e_{14} also produces $2 \times 3 = 6$ signals. But these signals are longer than the signals produced by the interface e_{13}, what means that the medium would be more occupied by e_{14} than by e_{13} and consequently v_1 is closer to v_4 than to v_3, although both interfaces produce each 6 signals. Thus, the closeness of two modules does not depend only on the number of signals produced by their interface, but also on the length of the signals. However, the length of the signals makes the difference only when two interfaces produce the same number of signals. To get a feeling of the impact caused by the length of a signal on the load of the medium, suppose that the bit-time on a given medium is t_b. The bit-time is the latency resulting from the propagation of a bit on the medium, i.e. the diffusion time of one bit. In a CAN network, the bit-time depends on the operating rate of the network and the length of the bus. Note that the bit-time is different from the transmission time of a bit. The latter is made up by the delay caused by the medium driver, the time spent in the CAN-controller and the CAN-transceiver of the receiver device, and the diffusion time of the bit on the specific actual medium. Since each frame contains the user data that is of variable length and a relatively constant number (i.e. 47) of framing bits, we will consider in the following only the delay induced by the user data for each frame. In a serial transmission method, the time that is necessary for the diffusion of a signal s of length $signalL$ (the framing bits are not included) will be approximately:

$$t(s) = signalL * t_b$$

and the total medium occupation time induced by the throughput of an interface e_{ij} is equal to:

$$t(e_{ij}) = \sum_{s \in e_{ij}} t(s) * s.freq$$

Thus, the higher $t(e_{ij})$, the closer the modules i and j. This formulation of the closeness results again in the number of bits exchanged between two modules. However, it only makes sense if it is applied on two comparable interfaces that produce the same number of signals.

In conclusion, the number of signals is by far the most efficient communication metric in a *CDFM* model. The higher the number of signals produced by an interface, the closer the two modules. When two interfaces produce the same number of signals, the communication load induced by each of them can be used to make the difference. The combination of these two metrics allows to fairly compare the magnitude of the communication between the modules of a *CDFM* model. The communication metric can thus be defined as follows:

$$\begin{aligned} Communication(v_i,v_j) &= nbSignals(e_{ij}) \\ &= \sum_k nbSignals(T_{ij}^k) + \sum_k nbSignals(T_{ji}^k) \end{aligned}$$

To compare two interfaces that have the same number of signals,

$$Communication(v_i,v_j) = t(e_{ij})$$

The constraints and strategic relationships metric: The closeness factors between the modules induced by the constraints on the communication and the strategic factors must also be quantified. These factors tend to change constantly and depend on unpredictable events. They cannot be accurately formalized. However, following their importance, these closeness factors can be classified objectively. For example, if a constraint or a strategic factor induces strength relationships between the modules, it can be modeled by the means of needs and excludes relations and then solved during the pre-clustering. Else, the designer will quantify these factors following the level of accuracy of their demand for achievement. In fact, the strategic closeness factors as well as the non-functional constraints can be classified in "must be" and different degrees of "will be nice" closeness levels whereas "must be" relationships can be modeled as needs and excludes relations. The degree of importance of a closeness factor in the "will be nice" class can be easily fixed exactly as the level of priority of a message or a task. In practice, an integer is sufficient to model such a ranking as the level of nicety of these factors can be evaluated so that each becomes the desired weight compared to the other factors. Thus, if a closeness factor is very important, it receives a high closeness value that will prevent the partitioning algorithm to ignore it. Otherwise, it is assigned a closeness value that can make the clustering difficult or easy depending on its level of nicety. We can thus assign a tag to each relation induced by a strategic factor that corresponds to its level of nicety. Each tag is an integer.

$$\begin{aligned} Relationships(v_i, v_j) &= \sum tags(e_{ij}) \\ &= Tag(e_{ij}) \end{aligned}$$

10.3.2 The closeness function

Because the closeness factors are disparate and mostly competing, some trade-offs must be done. Making trade-offs means that there is no unique solution. We thus have to find the best possible closeness function or the closeness functions that are achieved by acceptable trade-offs between the closeness factors and choose the optimal one. In this case, the optimality is not absolute, since the optimal solution refers to a point in the set of the optimal solutions, i.e. a Pareto optimal point. Anyway, deciding which trade-offs are acceptable might be subject to laborious investigations. The most intuitive approaches to make trade-offs include the weighting of the factors, the hierarchical optimization (also multilevel programming) method and the goal programming method. When proceeding by weighting the factors, the closeness functions are positively weighted and a single scalar closeness function is obtained by adding the weighted closeness functions. The weighting coefficients are then varied to yield a set of feasible optima, e.g. the Pareto optimal set. It is up to the user to choose appropriate weights to generate various points in the Pareto set. The weighting strategy is a standard technique for solving multi-objective optimization problems. But, typically for linear functions, the relationship between the weights and the Pareto curve is generally so that a uniform spread of weights rarely produces a uniform spread of points in the Pareto set. Often, all the points found are clustered in certain parts of the Pareto set with no point in the interesting middle part of the set, providing thereby little insight into the shape of the trade-off curve. In general, it is difficult for the user to find the optimal vector of weights. However, the closeness factors of our problem show that we can probably achieve a very simple closeness function with a combination of the closeness metrics.

The hierarchical optimization method aims at finding just one optimal point in opposition to the entire Pareto set. In this method, the closeness factors are firstly ordered following their importance. Each closeness function is then optimized individually, subject to a constraint that does not allow its minimum to exceed a prescribed fraction of the minimum of the previous function. The method proceeds recursively until all the objectives have been optimized on successively smaller sets of possible optimal solutions. This method is also called multilevel programming. Multilevel programming is a useful approach if the hierarchical order among the objectives is important and the designer is not interested in the continuous trade-offs among the functions. However, the

problem becomes very tightly constrained in the lower steps down in the hierarchy and often becomes numerically infeasible, so that the less important factors have no influence on the final result. Our problem easily fits in this method as the closeness factors can be clearly ordered. Furthermore, ignoring the less important closeness factors (resource sharing and common accesses metrics) is not dramatic at all.

The goal programming method aims at minimizing the deviation from the target constraints. In this approach, we optimize one factor while constraining the remaining factors to be less than the given target values. The user chooses weighting factors to rank the goals in order of importance. Finally a single objective function is written as the minimization of the deviations from the above stated goals. This is perhaps the most well-known method for solving multi-objectives optimization problems. However, it is not always easy to choose the appropriate goals and the good weights for the constraints. For these reasons the goal programming method is not a viable solution for our problem.

Consider the following function: $Closeness(v_i, v_j) = \sum_k \alpha_k ClosenessMetric_k(v_i, v_j)$ where the weights α_k correspond to the relative importance of the closeness factors (i.e. resource sharing, frame sharing, common accesses, relationships, constraints, communication). We now examine the weighting of the closeness factors: In a CAN network, the frames are owned by the sender, not by the receiver. Each frame can have several receivers but only one sender. There is therefore no obvious advantage to cluster the modules that have common accesses. In opposition, clustering the modules that can share frames can be beneficial. But before doing this, it is necessary to reduce the inter-device communication load as described in section 11.1.5. The reason of the uselessness of the resource sharing metric have been discussed above. In the end, the communication metric and the strategic relationships are the most useful metrics for the clustering. As the tags are assigned to the relationships in a way that they correspond to the weight they need, we can define the closeness function as follows:

$$\begin{aligned} Closeness(v_i, v_j) &= Communication(v_i, v_j) + Relationships(v_i, v_j) \\ &= nbSignals(e_{ij}) + Tag(e_{ij}) \\ &= \sum_k nbSignals(T_{ij}^k) + \sum_k nbSignals(T_{ji}^k) + Tag(e_{ij}) \end{aligned}$$

10.3.3 The QT clustering algorithm

The clustering deals with the multi-objective optimization problem identified above. The problem is to cluster the modules whose communication, if separated, might highly occupy the bus and to conjointly satisfy the given strategic concerns of the design. The capacities of the available devices are given. Using the above defined closeness function, we can easily extract the closeness values from the input model (i.e. the *CFDM* graph of super modules) and display them for example in a closeness matrix where excludes relations can be represented with a negative value, e.g. -1. The procedure "Clustering" finally runs the algorithm 4. It uses the procedure "InitialConfiguration" described in algorithm 3 to solve the excludes relations.

This clustering procedure assures that whenever two nodes underlie an *excludes* relation, they will not be assigned to the same cluster. At the beginning, all the nodes that can definitely not belong together are set as starting points for different clusters (in algorithm 3). These clusters are then completed before checking if new clusters are necessary (in algorithm 4). The remaining modules are thereafter clustered following an agglomerative clustering technique by which the closest pairs are merged. After the completion of the initial clusters, the working principle of the procedure can be resumed as: Whenever a cluster is initialized, grow it up until the given limit before the construction of the next cluster begins. This is a QT algorithm. At each time, the actual configuration of the partitioning is made of the remaining modules (i.e. super modules) that are not yet assigned to a part and the parts that are not yet completed. A module can be assigned to a part only if it is the closest for which the assignment is feasible. An assignment is

Algorithm 3 PROCEDURE InitialConfiguration(SuperModules)

InitialConfig:= SuperModules; //* *Initial set of parts*
repeat
  Traverse the set of super-modules;
until The first pair (v_i, v_j) with $excludes(v_i, v_j)$ is found;
Set v_i and v_j to be 2 different initial parts P_i and P_j of the partition;
RestOfSuperModules:= SuperModules - $\{v_i, v_j\}$;
InitialConfig:=RestOfSuperModules $\cup \{P_i, P_j\}$;
for all $v_l \in$ RestOfSuperModules **do**
  for all elements v_m that are already in a starting partition **do**
    if $excludes(v_l, v_m)$ **then** //* v_l *excludes all the existing parts*
      Set v_l to be another initial part P_l;
      RestOfSuperModules:= RestOfSuperModules - $\{v_l\}$;
      InitialConfig:= InitialConfig $\cup P_l$;
    end if;
  end for;
end for;
return InitialConfig; //* *The graph of the first initial clusters and the remaining super modules*

Algorithm 4 PROCEDURE Clustering(SuperModules)

Initialize CompletedParts:= $\{\}$; //* *Full clusters*
Run InitialConfiguration(SuperModules); //* *Outputs the initial parts and the remaining super modules*
repeat
  Find the closest pair (v_i, P_j) that is not constrained by an excludes relationship; //* *Best next assignment*
  if P_j can aggregate v_i **then** //* *Content of* P_j + $v_i \leq$ *Capacity of* P_j
    Assign v_i to P_j and delete v_i from RestOfSuperModules;
  else //* *Content of* P_j + $v_i >$ *Capacity of* P_j *but CAUTION! Pj is possibly not full*
    CompletedParts:= CompletedParts $\cup P_j$;
    InitialConfig:= InitialConfig - P_j;
  end if
until InitialConfig== $\{\}$; //* *The initial parts found by "InitialConfig" are all completed*
repeat
  if RestOfSuperModules $\neq \{\}$ **then**
    Choose a **central element** v_i of RestOfSuperModules and set it to be an initial part P_i in InitialConfig;
    Assign the closest module v_j of RestOfSuperModules to P_i, Then the next closest, and so on UNTIL the size of P_i meets its limit;
    CompletedParts:= CompletedParts $\cup P_i$;
  end if
until no remaining RestOfSuperModules;
InitialConfig:= CompletedParts;
return InitialConfig;

feasible if the behavior of the part and the behavior of the system are not compromised by the assignment. Whenever a module is assigned to a part, the closeness matrix is actualized.

The QT clustering procedure terminates successfully if each module is assigned to at least one part. It behaves greedily for both the initialization of the parts and their growing, i.e. no backtracking. This one-shot, one-dead strategy leads perforce to a hard, i.e. exclusive partitioning. However, the procedure can be easily adapted to achieve fuzzy, overlapping and redundant partitioning, for example by enabling the concerned modules to be assigned to the closest, the next closest part and so on, or by correspondingly modifying the assignment process, for example by marking instead of deleting a module from the set of free modules after its assignment to a cluster. Redundant partitioning is needed for example if redundant implementation of some modules is required, e.g. for safety and reliability reasons. Modules that must be partitioned fuzzily may have a degree of membership for each potential target cluster. The degree of membership is for example a number between 0 and 1 indicating the probability that the module will be assigned to the corresponding cluster. Fuzzy partitioning is a realistic partitioning option at the level of product-lines design where the mappings might change from a product variant to another one following some stochastic rules. These features can be easily implemented in an extended version of our algorithm.

10.3.4 Conclusion

We follow a three-step partitioning process beginning with the pre-clustering that solves the needs relationships and merges the needs-related components. Then, we use a powerful closeness function to proceed the clustering of the nodes of the graph that has been produced by the pre-clustering. The closeness of two given nodes of the super-graph is measured by a combination of the magnitude of the communication with the relations induced by the constraints and the strategic relationships between the modules of the system under design. At this step, we use a QT algorithm that is able to solve the excludes relationships. The algorithm designed here assumes that the capacities of the available devices are given. This information is used as the threshold. The clustering algorithm always yields a feasible partition of the input *CDFM* model, i.e. in the sense of the capacities of the allocated devices. But a further important question remains open: Is this the best algorithm or at least is this a good one? This question will be answered in the next chapters.

Chapter 11

Evaluating and improving a partition

The purpose of this chapter is to explain the method by which we improve a partition. Before improving a partition, we must evaluate it. Then we process a perturbation of its configuration and we evaluate the result. This is done a few times. At the end, we take the best solution. To achieve this process, we must define the cost function, i.e. the metric by which the value of a partition will be determined. As the optimizing of the usage of the consumption of communication frames is the primary declared goal of the partitioning process followed in this work, we begin this chapter by introducing the CAN protocol as the example for the frames-oriented communication protocols. Frames-oriented automotive communication protocols include LIN, FlexRay, CAN, MOST, etc. The CAN is the most prominent of this class of communication protocols in the automotive domain. We hereby discuss the frames multiplexing techniques and their relations with the partitioning. This discussion also includes an outline of the requirements for the multiplexing of the frames, the constraints that govern the multiplexing and the rules that guarantee a good partitioning in relation with the multiplexing of frames. Then, as the cost function is defined as a bin packing problem, we revise the state-of-the-art in the resolution of bin packing problems in order to find out if there exists a solution that is adequate for our problem. The FFD that is investigated as the best solution that fits to our purpose is used to design the algorithm implementing the cost function. We finally present our improvement algorithm. It is an adaption of the K&L partitioning approach.

11.1 The CAN: A frame-oriented communication protocol

11.1.1 Organization of a CAN network

CAN (Control Area Network) is a frames-oriented event-triggered communication concept used in automotive systems. The CAN uses a synchronous serial bit transmission method that affords the economizing of the communication bandwidth, i.e. an information is transmitted as a sequence of bits in a frame and when necessary, the synchronization between sender and receiver is done at the beginning of the frame, rather than for each bit as it would be the case in an asynchronous transmission mode. CAN is a multi-master network based on the carrier-sense multiple-access method with collision detection (CSMA/CD). Thus, a CAN network enjoys a flat organization in which all the nodes are equal in rights. Each node can place a frame on the medium as soon as it is ready. When a collision is detected, the node sending the frame with the highest priority wins the competition. The priority of a frame is coded in its identifier (ID). The lower the ID of a frame, the higher its priority, i.e. the frame with the smallest ID value has the highest priority. Collisions are resolved through a bitwise comparison of the IDs of the competing frames. Once a node wins the competition, it takes the control of the medium and finishes to send its frame before releasing. Furthermore, to each frame, a minimal time interval (i.e. the Inhibit-Time) can be assigned between two successive medium accesses (see the CMS-CAN frame Specification) that ensures that each frame will have the possibility to be placed on the medium independently of its priority and no node will be able to confiscate the medium for too long time.

Each CAN frame that is set on the medium is broadcast in the whole network and can be read by every device that is ready. Excepting some particular types of frames such as the network management (NM) and the diagnosis frames for which the receiver's address is introduced in the data part of the frames, the addressees of a CAN frame, i.e. those that are interested in the contents of the frame, are not coded in the frame. A device identifies a frame that is addressed to it only if it knows the ID of this frame. This is, it has been preprogrammed to receive the frame. CAN networks are generally implemented on linear two-wire buses, but the CAN protocol can be implemented as star topologies, on optic fiber or on single-wire buses as actually done for the comfort functions in AES (e.g. see section 1.1).

11.1.2 CAN frames

Following the CAN protocol, four classes of frames, also called telegrams, can be identified in a CAN network:

- The data frames: These are the frames that transport the application data.
- The remote frames: These are the frames used for data request, i.e. they are used by a node to require an information from another node.
- The error frames: These are the frames that can be used to notify a functioning error (e.g. during sending or reading).
- The overload frames: These are the frames that can be used by a node to request an extra delay between two consecutive frames.

The error frames are not used by the application, but rather by the basic software. Equally, the overload frames are not used by the application, but rather by the CAN drivers. The remote frames are formatted like the data frames as shown in figure 11.1 while the error and the overload frames have different formats [46]. The most known CAN implementations (CAN-controllers) do not use the overload and the remote frames. The spacing of successive frames is realized through the placement of "intermission" fields between two frames that impose a respectable inter-frame space. Data requests are achieved with data frames. The data frames are thus the only interesting frames for our purpose.

Depending on their role, there are the following types of data frames in a CAN system:

- Diagnosis frames
- Network management frames
- Data request frames
- Standard frames

The diagnosis frames are received by all the devices in the system including the diagnosis gateways. The network management frames are received by all the networked devices. The data request frames are received only by the devices that can send the requested information. The standard frames are received only by the devices that need the sent information. In this work, we are interested only in the transfer of the data (i.e. tokens) of the application software, thus in the standard data frames that can be used to send or request application-relevant information.

11.1.3 Format of a standard CAN data frame

We can distinguish three versions of the CAN protocol: The low-level CAN (ISO 11519) running up to 125 kbps, the so-called standard CAN (2.0A; ISO 11898:1993) and the extended CAN (2.0B; ISO 11898:1995) both running up to 1Mbps. Following the CAN specification (ISO-WD 16845),

each frame has a unique identifier of 11 bits length in the "Standard format" and 29 bits in the so-called "Extended format". In this work, we consider the standard format (i.e. IDE-bit = 1). The ID of a CAN frame, defined between 0 and 2047, clearly identifies the frame and the sender. As shown in figure 11.1, a standard CAN data frame is made up by 1 bit SOF (Start Of the Frame) + 11 bits Identifier + 1 bit RTR (Remote Transmit Request) + 1 bit IDE (Identifier Extension Bit) + 1 bit that is reserved for scalability + 4 bits for DLC (Data Length Code) + 0..8 bytes for User Data + 15 bits CRC (Cyclic Redundancy Check) + 1 bit DEL (Delimiter) + 1 bit ACK + 1 bit again DEL + 7 bits EOF (End Of Frame) + 3++ bits IFS (Inter Frame Space). The IFS has at least three bits that are part of the frame. All additional bits that can appear in the IFS are not part of the frame, but they are solely considered as separation bits.

SOF	Identifier	RTR	Res.	DLC	User Data	CRC	DEL	ACK	DEL	EOF	IFS
1	11	1	2	4	0..8 bytes	15	1	1	1	7	3++

Figure 11.1: Format of a standard CAN data frame

The CRC contains the information needed to check the safety of the transmission, i.e. the integrity of the frame, that is confirmed by setting the ACK (acknowledge) bit. The CRC has a 1 bit delimiter (DEL). The IDE-bit, the scalability-bit and the DLC-bits build the Control Field of a CAN frame. The IDE-bit indicates the format of the frame, i.e. value 1 for standard and 0 for extended format. The RTR-bit is used to distinguish a data frame (RTR = 1, i.e. dominant) from a remote frame (RTR = 0, i.e. recessive). All the bits used in a CAN frame around the user data (e.g. the 47 heading and queue bits used in the best case, i.e. with the minimum -3 bits- EOF in standard CAN frames) will be called in the remainder of this work "Framing Bits".

11.1.4 Frames multiplexing

In a CAN network, each piece of information exchanged between two functional components of the system that are located on different devices is transported by a frame. After the partitioning, each device is assigned a given set of data frames, i.e. frames IDs, that must be sufficient to transfer the application's data to the other devices. The application's data of a device is the set of the tokens that are emitted by the corresponding logical device. For a device P_i of a partition P, this set is given by the operator $Tokens(P_i)$ that extracts all the tokens for which the source is P_i, i.e.

$$Tokens(P_i) := \left\{ T_{ij}^k, for\ each\ device\ j;\ k \in \mathbb{N} \right\}$$

The name, the ID, the DLC and the contents of the user data part of each frame are statically predefined. In complex systems like automotive systems, the number of tokens exchanged via the network medium is generally more than the number of the frames that are available. For illustration, less than half of the total possible 2048 frames are available in the CAL (CAN Application Layer) DBT-Model, i.e. are free to be used for data transfer, and a large number of these frames is reserved for the system, the basic software services and other predefined messages like the network management and the diagnosis messages. Thus, it is not opportune to invest one frame for each token. Even if we are provided sufficient frames, the bandwidth of the network is not unlimited and the throughput in an AES network must be designed so that even by maximal load, there is still room for the transmission of asynchronous messages, particularly in the case of event-triggered communication systems.

As the number of frames is limited, the designer must optimize their usage. An effective way to optimize the usage of frames is to make use of frames multiplexing [46], i.e. under some conditions several pieces of information can be joined together in a single frame in a way that increases the economy of the frames. A frame to which more than one piece of information is assigned is a

multiplexed frame. A device receiving a multiplexed frame must de-multiplex it to extract the information that is addressed to it. In order to enable the frame de-multiplexing, a part of the user data field (e.g. the first byte of the user data field in CAN) of a multiplexed frame is used to code the sub-identifiers of the sub-frames, i.e. the IDs of the different entities of information contained in the frame. We now introduce the following definitions for the terms that will be used in the context of frame multiplexing. The relationships between these terms are illustrated in figure 11.2.

Definition 11.1 (**Data object**)**.** A data object is a piece of user information exchanged between two functional components.

Definition 11.2 (**Token**)**.** A token is the logical representation of a data object in the context of synthesis models.

Definition 11.3 (**Signal**)**.** A signal is an atomic, i.e. an elementary piece of user information that is sent by a device to another one. Each signal represents a token or a fraction of a token. For example, a token that fits in a frame constitutes one signal. A token that is longer than a frame can be split in several signals. Thus, a multiplexed frame contains several signals.

Definition 11.4 (**Packet**)**.** A packet is a set of signals that are sent together within a frame, i.e. as a block of sequential signals. A multiplexed frame contains exactly one packet of signals, whereas each signal constitutes a sub-frame. The sum of the signals transported within a frame determines the payload of this frame.

Definition 11.5 (**Message**)**.** A message is a meaningful and complete piece of information. A message might be made of one or several tokens. For example, some tokens are constrained to be always shipped together. This relationship is modeled in *CDFM* models by means of *needs* relations. The tokens that *need* each other are bundled within a message. If a token is *needs*-related with several other tokens, then it will be redundantly instantiated in each message containing an associated token. The length of a message *messL* is defined as the sum of the resolutions of the constituting tokens.

Definition 11.6 (**Sub-frame**)**.** A sub-frame is a distinguishable and separable part of a multiplexed frame. This concept aims at providing an ID to each signal within a multiplexed frame. From the point of view of a multiplexed frame, these IDs are sub-IDs. A sub-frame is thus made of one signal and the corresponding sub-ID. Note that the sub-identification of frames is not recursive, i.e. sub-frames do not incorporate sub-IDs, since each sub-frame contains exactly one signal.

Definition 11.7 (**Frame**)**.** A frame is a block of information that is transferred from a sender device to the receivers. A frame is made up of one signal or when multiplexed, it contains several sub-frames. The concept of frame is analog to the concept of slots known in some protocols like FlexRay or MOST.

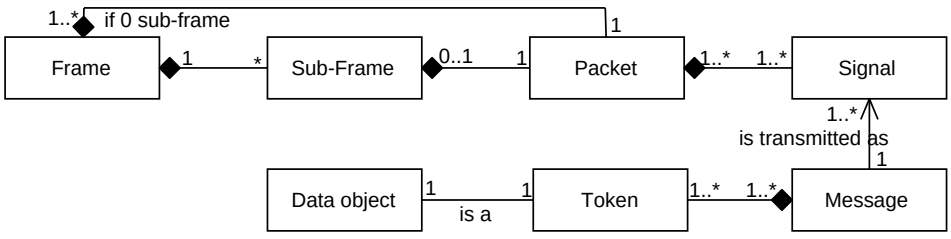


Figure 11.2: Semantical relationships between the concepts for the frames multiplexing

Another multiplexing technique allows the variable occupation of some positions within a multiplexed frame. For example, different signals can rotatively occupy the same position in a given frame. This multiplexing technique is used for example when a periodic signal can share a frame with other signals, but at different sending times of the frame, different signals are available that can occupy the free positions. When this multiplexing technique is used, a switch is used to identify

the signal that actually occupies the varying position in the frame. A switch is a bit field within the user data field of a varying multiplexed frame that indicates the signal that actually occupies the corresponding varying position so that depending on its actual runtime value, a device receiving the frame knows which signal is actually occupying the corresponding position. Both the values of the sub-frames and the switches are assigned statically during the system design.

11.1.5 Relations with the partitioning

The frames multiplexing itself is a signal partitioning problem. Actually, the frames are assigned to the network nodes based only on the experience and the feeling of the designer. The assignment of the signals to the allocated frames is done per hand, based on approximative calculations and puzzles reasoning. Inexperienced designers cannot expect to find optimal solutions in a relative short time. Even experimented designers cannot verify the optimality of their solutions, although there is a need of optimal system architectures and design time saving. Sustaining the frames multiplexing through CAD is obviously necessary to allow AES system architects to explore new system architectures. CAD-based frames multiplexing will promote the evolution in the quality of AES architectures and enable savings in the design time of systems that are based on frames-oriented protocols. Since we make benefits if we optimally occupy the user data field of each frame, we will provide efficient solutions for the frames multiplexing problem if we are able to solve the two following sub-problems:

1. Minimize the number of tokens that must be exchanged between the system components through the network.

2. Optimize the frames multiplexing for these tokens.

The first sub-problem is a graph partitioning problem. In fact, we will minimize the number of tokens exchanged between the logical devices if we assign heavily communicating components to the same device. The goal here is to minimize the inter-device communication. The second sub-problem is a combinatorial optimization problem that is to be solved by finding the best combinations of the signals derived from the inter-device communication tokens that will be assigned to a given set of frames. In fact, the combination of data objects to fill up the frames is a classical packing problem.

11.1.6 Practical considerations for the frames multiplexing

When multiplexing the frames, we must observe some practical considerations concerning the capacity of the frames and the relationships between the tokens. For instance:

- Each frame, i.e. each frame ID, can be attributed to only one sender device.

- The quantity of data that can be packed in a frame is bounded by the length of the user data field of the frame. We call the latter ($maxframeL$). In the standard CAN $maxframeL = 8$ bytes.

- 0 byte signals are not allowed, i.e. $minframeL > 0$.

The length of a token is its resolution. We now identify a token that is shorter or equal to *maxframeL* as a "short" token and a token that is longer than *maxframeL* as a "long" token.

- Each short token can be assigned only entirely to a frame, i.e. a short token cannot be truncated to fill several different frames. For example, if a token T_{ij}^k is 4 bytes long and a frame that could take it already contains 5 bytes of user data, it is not allowed to assign 2 bytes of T_{ij}^k to this frame and the remaining 2 bytes to another frame, but rather the 4 bytes of T_{ij}^k must be assigned together to a frame that provides free space for 4 bytes. In other

words, fragmenting short tokens is not allowed although doing so might enable to fill the gaps in the frames and could drastically enhance the economy of the frames. But, considering the recovery time, i.e. extraction and reconstitution time, and the related computation efforts with regards to the fragmented tokens, the fragmentation of short tokens is clearly not profitable.

- In contrast, long tokens must be fragmented and distributed to several frames. For example if a token is 18 bytes long, it will be assigned to 3 frames, i.e. 2 frames for the first 2 x 7 bytes and the last 4 bytes in a third frame. However, it won't work differently, since a long token cannot fit in a single frame. The case of long tokens appears for example when texts, system initialization information, configuration parameters or program codes are transferred. The fragmentation of tokens is done by a fragmentation protocol that can be easily specified by the designer.

When assigned to a frame, each short token results in one signal whereas long tokens must be transformed into several signals. Remember that a part of the user data field of each frame transporting a fragment of a token is used to encode the fragment's number so that the receiver can easily reconstitute the order of the fragments as it is in the original token. In some CAN implementations, the segment code occupies the first byte of the user data field. This explains why a frame containing a fragment of a token has a capacity of 7 bytes instead of 8 bytes of user data in the above illustrations. Similarly, because of the sub-IDs, the full capacity of a multiplexed frame cannot be occupied by the user data. Let *segmentCode* be the space needed to code this information. The number of signals corresponding to a token T_{ij}^k is:

$$nbSignals(T_{ij}^k) = \begin{cases} 1 & \text{if } T_{ij}^k \text{ is short} \\ \left\lceil \frac{T_{ij}^k.res}{(maxframeL - segmentCode)} \right\rceil & \text{if } T_{ij}^k \text{ is long} \end{cases} \quad (11.1)$$

A message M_i^q (q identifies the message itself) is built with the tokens T_{ij}^k, $k \in \mathbb{N}$ that verify $\forall T_{ip}^s, T_{il}^r \in M_i^q$, $needs(T_{ip}^s, T_{il}^r)$ where i is the device source of the message M_i^q; j, l, p are the destination devices of the tokens and k, s, r identify the tokens.

The number of signals corresponding to a message M_i^q is:

$$nbSignals(M_i^q) = \sum_j \sum_k nbSignals(T_{ij}^k) \quad (11.2)$$

As the devices must compete for the bus in a CAN network, the frame IDs should be allocated to the devices according to the characteristics and the constraints of the signals that must be transferred, e.g. priority, date of occurrence, freshness, etc. Thereto, as we are designing systems that are highly business- and mission-critical, the frames multiplexing will be constrained by the reliability- and the safety-relevant requirements of the system. For instance, each signal underlies real-time and safety constraints. Hard real-time signals must be transmitted as soon as possible, thus they necessitate high priority. Soft real-time signals are less hasty than hard real-time ones and can wait for some payload completing signals for a given time as authorized by their freshness requirement. However, each signal must be transmitted and received safely. Depending on the characteristics of the signals that must be transferred, we can classify them on the basis of the following criteria:

- The **synchrony**: We distinguish event-triggered (i.e. asynchronous, sporadic) and time-triggered (i.e. synchronous, basically periodic) signal. Both periodic and non-periodic signals might underly real-time constraints.

- The **dynamic**: This is the periodicity or the rate at which a signal must be sent. Between the signals that are sent repeatedly, some appear very fast while some others are rare as their instances appear always after a long period of time. Same, some asynchronous signals appear regularly while others are very rare.

Based on the synchrony ans the dynamic of the signals, we consider two classes of signals:

- ASAP signals, i.e. signals that must be sent "As Soon As possible". This class contains both cyclic and event-triggered hard real-time signals. The frames transporting ASAP signals must enjoy high priority so that they can be transferred as soon as possible. ASAP signals can share frames only with signals that are available at the date at which they are sent (i.e. their sending date).
- SOFT signals, i.e. signals with an allowed retardation time. SOFT signals can be retarded for a given time that is defined according to their freshness requirement. The freshness attribute of a signals is given by the freshness requirement of the corresponding token. This class of signals can be subdivided in several sub-classes, each containing the signals that are available within a given range of time so that they can share the same frame as allowed by their resolutions.

In conclusion, a short token can share frames with every other token that is available at its sending time. Long tokens must be segmented and sent in combination with the tokens that are available at their sending time. However, in order to avoid mini frames, the partitioning process must prevent the existence of mini signals that are ASAP, e.g. by avoiding the exchange of mini tokens that are ASAP through the network. These are additional requirements for the partitioning. In the end, the priority of a frame will be assigned depending on the priority of the highest prioritized signal contained in it. The sending time and the waiting time of a frame is given by the sending time of the most hasty signal contained in it.

11.2 The value of a partition

11.2.1 The cost function

The cost of a partition is the magnitude of the communication on the network bus. This is the number of frames needed to realize the inter-cluster communication. Good partitions need fewer frames. Given a partition $P = \{P_1, P_2, \ldots, P_n\}$,

$$Cost(P) = NbFrames(P)$$

$NbFrames(P)$ is the total number of frames that is used by all the parts P_i of the partition P to communicate. Since each frame is owned by a cluster, $NbFrames(P)$ is the sum of the frames used by all the clusters P_i of the partition P. Given the set $Signals(P_i)$ of the signals that are produced by the cluster P_i, we can extract the set $Signals(P_i).occur$ of the dates of occurrence of the signals emitted by P_i and the set $Signals(P_i).fresh$ of their freshness requirements. Note that $Signals(P_i).occur$ and $Signals(P_i).fresh$ are given by the dates of occurrence of the tokens from which $Signals(P_i)$ is derived, i.e. given a signal s derived from a token T_{ij}^k,

$$s.occur \;=\; T_{ij}^k.occur \text{ and } s.fresh \;=\; T_{ij}^k.fresh$$

Thus

$$\begin{aligned} waiting\ time(s) &= waiting\ time(T_{ij}^k) && (11.3)\\ &= s.fresh - s.occur && (11.4) \end{aligned}$$

Given a set of signals, we can group them according to their waiting time so that each group is made of signals that can share frames, i.e. signals that can be sent at the same time. The sending date of an ASAP signal is the first opportunity that is given by the scheduling of the bus occupation to sent it. In contrast, SOFT signals can wait for other signals until the date defined by their freshness attribute as they must not be sent at the first opportunity, i.e. SOFT signals can

be grouped together with the signals that are available within their waiting time. Equally, even though the waiting time of ASAP signals is very reduced (theoretically zero), ASAP signals can share frames with signals that are available at their sending date regardless of whether these signals are ASAP or SOFT. Each frame will thus contain a particular combination of signals that can be either ASAP or SOFT. Concretely, each group of signals will content the signals that are available within a given range of time that begins with the date of occurrence of the oldest between them and finishes with their common sending date. Each of such groups will need a certain number of frames to transfer its signals. Let us identify each such group g of signals with the date d_g at which its signals must have been sent and call it $Signals(P_i).d_g$. Let $NbFrames(P_i).d_g$ be the number of frames that will be used by the group $Signals(P_i).d_g$. $NbFrames(P_i).d_g$ is thus the number of frames that will be used by the cluster P_i at a given date around d_g, more exactly at a give date that hopefully precedes d_g. Note that even if d_g can coincide with the date of occurrence of some signals, it neither represents the date of occurrence of signals nor it states the exact date of emission of frames. d_g rather specifies the time at which the signals that can build a frame must have been sent. The number of frames used by a cluster P_i is consequently the sum of all the frames emitted by P_i at the dates $\{d_g,\ g\ \in \mathscr{P}(Signals(P_i))\}$., i.e. g is a group.

Given a partition P, we can formalize the cost of P as follows:

$$Cost(P) = NbFrames(P) \tag{11.5}$$
$$= \sum_i NbFrames(P_i) \tag{11.6}$$
$$= \sum_i \sum_g NbFrames(P_i).d_g, \tag{11.7}$$

where P is the actual partition, P_i is the i-th part of the partition P, $\{d_g,\ g\ \in \mathscr{P}(Signals(P_i))\}$ is the set of dates at which the signal groups must have been sent.

Given a part P_i, each frame of P_i will contain a particular combination of the elements of the corresponding $Signals(P_i).d_g$. As the objective is to minimize the usage of the frames, $NbFrames(P_i).d_g$ must be determined by a function that optimally combines the signals of $Signals(P_i).d_g$ into the frames. If only one signal is available in a given group $Signals(P_i).d_g$, then the corresponding frame will contain only this signal, else, i.e. several signals are available at the same time, the signals must be packed into the frames in a way that the capacity of the frames is used optimally in order to minimize the number of frames used by P_i, i.e. $NbFrames(P_i)$. The problem of determining $NbFrames(P_i).d_g$ can be specified as follows:

Problem formulation 11.1 ($NbFrames(P_i).d_g$)**.** : Given a finite set $Signals(P_i).d_g$ of signals of different sizes and a supply of frames of fixed capacity *maxFrameL*, compile a packing list of the signals such that the sum of the signals in each frame is equal or less to the frames capacity, a minimal number of frames is used and the given constraints on the contents of the frames are satisfied.

This problem is a bin packing problem.

11.2.2 The cost as a bin packing problem

The purpose of a bin packing problem is to fill large spaces with specified smaller pieces in the most economical way. In the classical bin packing problem, a set or a sequence [37] of objects are to be packed into fixed-size bins (i.e. containers of fixed volume). The supply of the bins is unbounded and the sum of the sizes of the objects in a bin cannot exceed the size of the bin. The problem is to minimize the number of bins used, i.e. to determine the minimum number of identical bins needed to store a finite collection of objects of discrete sizes. Formulated like this, a bin packing problem is an optimization problem, since the solution is to find an optimal partitioning of the objects to fill the bins. Bin packing problems have many applications, e.g. trucks loading, memory management [114], temporal partitioning of reconfigurable processors (FPGA) [29], etc. Even though these

problems seem to be simple, they are NP-hard, i.e. no procedure is able to solve each instance of a bin packing problem in polynomial time [54]. In fact, the decision version of a bin packing problem is known to be NP-complete. That is, given a capacity K and a list L of objects with different sizes and an integer N, the problem of determining if the objects in L can be packed into N or fewer bins of capacity K is NP-complete. A proof is given in [34]. We know that each optimization problem whose decision version is NP-complete is NP-hard. Thus formulated as an optimization problem, the bin packing problem is NP-hard. Detailed proofs of the NP-completeness and NP-hardness of bin packing can be found in [54] and [106]. Thus, when solving a bin packing problem, finding approximately optimal solutions in polynomial time is probably the best we can hope for.

As knapsack problems, bin packing problems are members of the class of combinatorial and discrete optimization problems. In this domain, mathematics usually meet puzzle. The most efficient algorithms for solving bin packing problems use heuristics. The problem we are dealing with is a linear (i.e. one-dimensional) version of the bin packing problem. We are concerned with finding the fewest number of frames needed to hold a set of signals of different sizes. This formulation meets the optimization version of the bin packing problem that is NP-hard. Knowing the constraints of our problem, we need to design the best algorithm that will optimally solve it. As the problem is NP-hard, we mean best algorithm in terms of the quality of the output (i.e. distance from the optimal solution). The quality of the algorithm in terms of its computation complexity and its consumption of memory space is of secondary relevance. In fact, given a bin packing problem, we can calculate the optimal solution (OPT) by computing the total sum of the sizes of the objects E and dividing this number by the capacity of the bins C. Since we are dealing with integers, the number of bins we need must be at least $OPT = \lceil E/C \rceil$. But, the objects to be packed are generally of different sizes so that they cannot be distributed uniformly in the bins. The number of bins used is thus typically greater than the optimal solution.

11.3 Bin packing techniques

11.3.1 The Next Fit, the First Fit and the Best Fit strategies

Most solutions for bin packing problems rely on very natural ideas. The simplest is the Next Fit (NF) strategy. In the NF strategy, a bin is opened and the objects are placed into it in the order in which they appear in the list of objects. If an object will not fit into the open bin, we close this bin permanently and we open a new one in which we continue to pack the remaining objects. NF is very simple and fast. The time required to pack n objects is linear in time, i.e. $O(n)$. But clearly, it is difficult to achieve an optimal usage of bins with NF. Indeed, NF is the best solution in certain settings of the bin packing problem. For example, NF allows for bins to be shipped off quickly because even if some free room remains in a bin, we do not wait around in the hope that an object which will fill this empty space will come out later in the list. We ship the bins as soon as they cannot take the actual object and gain in storing space. Unfortunately, we are not in the situation where there is limited space to place the bins, implying that they must be rapidly closed and shipped, but in opposition we are interested essentially in achieving the best usage of the available space in the bins. NF is not the best strategy for solving our problem.

In contrast to NF, some solutions try to take advantage from the intuitive fact that if the bins are kept open, some objects that can fill the empty spaces will come out from the list later so that they will consume fewer bins. The most famous strategies based on this idea include First Fit (FF), Best Fit (BF) and Worst Fit (WF). Following a FF strategy, we place the next object into the first bin that has room for it (the leftmost thought of as numbered from left to right) and we open a new bin if the object does not fit in any existing bin. The objects are considered for packing in the order 1, 2, 3, ... The bins are labeled as 1, 2, 3, ... FF then packs object i in bin j where j is the least index so that bin j can contain object i. The BF procedure runs through the same steps as FF, except that when object i is to be packed, it finds out the bin which after accommodating object i will have the least amount of space left, i.e. it puts the next object in the

fullest bin that has room for it and opens a new bin if the object does not fit in any existing bin. FF and BF can be implemented to run in $O(nlogn)$ time. In contrast to BF, Worst Fit (WF) puts the next object in the emptiest bin that has room for it and opens a new bin if the object does not fit in any existing bin. WF has the same worst-case performance ratio with NF [76]. But in general, it apparently does not take advantage from the fact that it never closes a bin. [77] Finally, one can consider a further procedure, called Almost Worst Fit (AWF) that puts the next object in the second emptiest bin if that bin has room for it and opens a new bin when the object does not fit in any open bin. Although AWF is just a slight modification of the WF algorithm, [77] reports that it performs clearly better than WF, i.e. just as good as FF and BF. The time necessary to find the minimum number of bins for n objects using either FF, AWF or BF is higher than for NF, i.e. $O(n)$ against $O(nlogn)$. These strategies behave as if the objects to pack arrive surprisingly, one-after-one, like on-line algorithms. This is surely a limitation. The parameters of our problem (i.e. the number and the sizes of the signals) are all known in advance. Thus we can organize the signals in a way that enables more advantageous packing.

11.3.2 Off-line packing strategies

Some algorithms have demonstrated that they guarantee better packings when the on-line restriction is removed. The idea guiding the most known of these algorithms is to apply FF or BF on a sorted list of objects, i.e. they first sort the objects to be packed in the bins in decreasing or in increasing order by the size. The sorting is relatively expensive ($O(n^2)$), but without sorting, the packing can achieve only the bound of $17/10\ OPT+2$ bins [56]. The most popular of these strategies have an intuitive appeal. They pack the bulky objects first and hope that the smaller objects can be used to fill up the gaps. In the First Fit Decreasing (FFD), the objects are sorted in decreasing order of size. The biggest object is the first and the smallest is the last. Then FF is used, i.e. FFD inserts the objects of the sorted list one-by-one into the first bin with sufficient remaining space. Similarly, Best Fit Decreasing (BFD) orders the objects like FFD but uses BF. Doing so, FFD and BFD improve drastically the performance of FF and BF. Depending on the implementation, the FFD and BFD strategies can achieve the bound of $11/9\ \ OPT+1$ [125]. The theorem 2.9 in [34] (p.17) referring to [76] states that: Any algorithm that sorts the items by decreasing size and then applies any fit packing rule is worse than BFD and FFD, more precisely $11/9 \leq R_A \leq 5/4$. Also, any algorithm that applies any fit algorithm after first sorting the objects in increasing order by the size has a ratio $R_A \geq 1.69103$, thus is worse than the decreasing sort.

Thus, between all the algorithms studied till now, FFD and BFD provide the best performance. They also fit at best to our problem. When well implemented, they can require $O(nlogn+bn)$ time to pack n objects, where $b \leq min(n,m)$ and m is the number of bins actually used. Analytical and empirical results suggest that the best of the two heuristics is FFD. In [55] and [126] an efficient version of FFD that uses no more than $71/60\ OPT+1$ bins is presented. However, [76] states that for all lists L with no element smaller than $1/6$ of the bin size, BFD(L) $\leq$ FFD(L) and precises that for such lists, BFD(L) $\leq 6/5\ OPT(L)+1$. But the authors of [55] and [126] also claim that if smaller objects are present in the list, BFD can result in worser packing than FFD. Thereto, according to [76], the worst-case ratios of FFD and BFD are improved as the minimum item size decreases. In the purist sense of optimization theories, this is a notable improvement. But, is this improvement interesting for our particular purpose? The signals we need to pack are of disparate sizes, i.e. we are not aware of any restriction on the size of a signal, except the basic constraint stating that the maximal length of a packet, thus of a signal, must not exceed the size of a frame. At this level of investigation, FFD is the best solution for solving our problem.

11.4 Investigating the cost of a partition

11.4.1 The FFD strategy for the cost estimation

As we need the best packing solution for our problem, i.e. the packing tool that achieves the minimum number of bins, it is interesting to investigate the improvements of FFD. There are basically two approaches to improve the worst-case performance of the FFD and BFD. Firstly, one can imagine a procedure that packs pairs or groups of objects at the same time in combination with FFD or BFD instead of one-by-one as done by classical FFD and BFD. It is also imaginable to pick the objects from the list in a different order than the decreasing one as done so far. In a second approach, going from the intuitive apprehension that the configuration of the list of the objects is a highly influencing factor for the outcome of the packing, we can legitimately imagine to restrict the attention to specific types of input lists. These two approaches have retained the attention of researchers working on packing problems and the literature on the corresponding investigations reveals very useful information.

We first pay attention to the run-time behavior of FFD and BFD. Rememeber that we are working on a static partitioning problem for which the solution will be reused for a long period of time on different products or on several variants of the same product. The most important goal of the partitioning is the optimality of the space usage in the bins. It is thus preferable to spend extra time to search the best packing than to save computing time at the cost of the quality of the packing solution. However, as the partitioning runs the packing algorithm many times, it will still be opportune to take care not to spend too much time in each execution of the algorithm. Concerning the time-complexity of FFD and BFD, an interesting question is to see what can be done better than FFD and BFD with less than $O(nlogn)$ time. We pick an answer once more from [34] that claims that "No algorithm is currently known that does better than $4/3$ bound and runs in linear time on a sequential computer".

We are now interested in investigating what we can gain if we are allowed more than $O(nlogn)$ time. The answer is given in [124] where the existence of an algorithm with asymptotic worst-case ration $11/9 - 10^{-7}$ and $O(n^{10}logn)$ run-time, called Refined FFD is shown. This algorithm provides an improvement of bin utilization that is however not so significant that we are motivated to adopt it. Garey and Johnson [55] proposed an algorithm called Modified FFD (MFFD) that is supposed to improve the classical FFD more substantially than the Refined FFD. The MFFD differs from FFD in the way that it treats the objects with sizes in the interval $[1/6, 1/3]$. When packing these objects, the MFFD currently considers the bins containing a single object of size $> 1/2$ (called A-bins) from right to left, i.e. in order of decreasing gaps. To treat the current A-bin, the algorithm first checks if the two smallest still unpacked objects with size in $[1/6, 1/3]$ will fit together in the A-bin. If so, it places the smallest of such objects in the bin, together with the largest of the remaining objects that will fit with it. If not, this special phase is over and all the remaining unpacked objects are added to the current packing according to FFD. The MFFD has the same $O(nlogn)$ running-time as FFD, and has roughly the same constant of proportionality. Its worst-case performance is somewhat better than the one of FFD, i.e. $71/60$. Also trying to improve FFD, Friesen and Langston [50] proposed an approach to beat the worst-case performance of FFD. Same like the MFFD, their algorithm, the Best Two Fit (B2F), tries to improve the packing of individual bins by considering pairs of objects as well as individual objects in making the packing decisions. They showed that the worst-case performances of their algorithm and of FFD were complementary and concluded that the compound algorithm (CFB) that runs both the B2F and the FFD and outputs the best of the two solutions shall do better than each of the algorithms. But although it runs in twice the overall running time, even the compound algorithm does not improve the FFD notably.

Further attempts to beat the FFD by relaxing the constraint of $O(nlogn)$ running-time on the packing algorithm can be found in [62, 77]. In any case, a good news is that, for all these solutions, worst-case ratios approach 1 as the maximum object size approaches 0 (this meets the

human intuition). But at the end, except the enumeration of all possible combinations, it is difficult to find a real improvement of the FFD that can motivate our choice for it by guaranteeing better outcomes on non-restricted input lists of objects. It is shown in [33] that if the list is a divisible sequence, FFD always produces optimal packing. Further instances of the packing problem that theoretically can be solved optimally in polynomial time (i.e. without trying all packing) by setting profitable restrictions on the list are those where the number of object sizes are bounded independently of the number of objects. These algorithms would be the solution to our problem if we had the corresponding lists. Unfortunately we do not have either divisible lists nor given bounds on the number of object sizes in our groups of signals. There is also a particular class of algorithms that can find a suboptimal solution for the bin packing problem that is within a known percentage of the optimal solution for sufficiently large inputs of arbitrary lists in linear time. These algorithms are called asymptotic approximation schemes. Asymptotic polynomial approximation schemes can be found in [48, 58, 80]. In conclusion, the algorithms improving FFD might yield good performance, but at the price of either setting restrictions on the type of the list for the first ones or as the latter, giving up too much performance in the worst case and so, they cannot profitably serve our purpose of searching an algorithm with the best worst-case guarantee. We thus maintain finally the FFD.

11.4.2 The frames packing algorithm for the cost investigation

Our procedure for the packaging of signals follows the process described by algorithm 5.

Algorithm 5 PROCEDURE FramesPacking($Signals(P_i).d_g$)

1: Sort the elements of $Signals(P_i).d_g$ in the decreasing order of their lengths in a vector $vSignals(P_i).d_g$; *//* indexes 1,2,3, ...*
2: k:=1;
3: Initiate a frame $Frame_k$ of the corresponding priority; *//* First frame for the group*
4: **for all** signals in the vector $vSignals(P_i).d_g$ beginning with the first one **do**
5: Assign the signal to the first frame that can accommodate it; *//* FFD*
6: **if** no frame can accommodate the signal **then**
7: Add a new frame to the list of frames and put the signal inside;
8: **end if**
9: **end for**
10: **return** $NbFrames(P_i).d_g$; *//* Number of frames used by* $Signals(P_i).d_g$
11: **return** The packing list of the frames; *//* The composition of the frames, i.e. the packets*

The cost of a partition P is computed by the procedure "Cost" the processing of which is described in algorithm 6.

Algorithm 6 PROCEDURE Cost(P)

1: Cost(P):=0;
2: **for all** parts P_i of P **do** *//** $P = \{P_1, P_2, \ldots, P_n\}$
3: NbFrames(P_i):= 0;
4: **for all** groups of signals $Signals(P_i).d_g$ **do**
5: FramesPacking($Signals(P_i).d_g$); *//* Returns the number and the composition of the frames used*
6: NbFrames(P_i):= NbFrames(P_i) + NbFrames(P_i).d_g;
7: **end for**;
8: Cost(P):= Cost(P) + NbFrames(P_i);
9: **end for**;
10: **return** Cost(P); *//* Number of frames used, Total cost of the partition*

11.5 Improving the partition

11.5.1 The Kernighan & Lin strategy

Despite its popularity, the K&L heuristic [86] is not a standard solution for every partitioning problem. It was designed to solve bi-partitioning problems on undirected graphs for which the output must be two parts with equal sizes. Our problem is totally different from the general formulation of the K&L algorithm, for example: We have more than two parts. Thus, our problem necessitates a multi-way partitioning solution rather than the original two-way K&L solution. The sizes of the nodes of the graphs on which we are working are very different from each other. Thus, swaps may not be sufficient. The sizes of the parts of the partitions are also very divers and it is not obligatory to maintain them constant during the partitioning. They can change as the free space on each device is known. Thus, the improvement algorithm for our problem must allow the nodes to be moved both without and with swaps, as the case may be. The cost of the partition is given by the packing of the packets rather than the sum or the weights of the cutting edges. Thus, in its classical form, the K&L algorithm does not fit to our problem.

The most famous extension of the K&L that allows the moves without swaps is the algorithm of Fiduccia and Mattheyses (F&M)[49]. F&M extended the K&L algorithm to handle multi-terminal graphs (i.e. hyper-graphs) by redefining the cut metric and the move operation. Like the K&L, the F&M allows each node to move only once. However, F&M abandoned the idea of interchanging two nodes in a single move. They adopted instead the move of a single node. Consequently, for each move, only the gain of moving each single node must be computed. Then, instead of swapping two nodes like in the K&L, only the node with the highest gain is moved. This led to a faster and more flexible algorithm that supports unbalanced partitions. In order to avoid that all the nodes move to one part, F&M introduced a balance criterion ($r = |A|/(|A| + |B|)$) that states the minimum number of nodes for each part. The third major change was done on the data structure. F&M used an array of lists to store the gain of the nodes. This structure enables to find the best next move and to adjust the recomputed node gains in constant time, since the list of the candidates for a move in each part is kept sorted every time.

Several variations of the F&M algorithm have been proposed to solve similar problems. For example, still considering the problem of bi-partitioning hyper-graphs, [90] proposed a refined method for choosing the best cell to be moved. The cost function is the number of nets in the cut set of the hyper-graph. This solution introduced the binding number for the nets, a value that indicates how strongly a net is bound to a part, and the notion of the level gains which measures the potential decrease of the size of the cut set that would result from moving a node from its home part to the other part. The number of level gains is an input parameter for the algorithm. At each iteration of the algorithm, the free node with the largest gain vector is chosen to be moved. The components of the gain vector of a node are the level gains of this node, beginning with the first level gain. The definition of the first level gain is based on the gain concept used in the graphs partitioning algorithms. With the binding number and the level gains, this algorithm is very powerful for partitioning hyper-graphs, but like the F&M it is tailored for bi-partitioning problems for which the cost is the number of nets in the cut set. Neither the bi-partitioning nor the cost matches with the purpose of our matter. However, the idea of the binding and the gain levels can be useful guides for moving the nodes of *CDFM* graphs if adapted to a multi-way partitioning problem that supports the definition of our cost function.

There are several ways in which a two-way partitioning algorithm can be adapted to solve a multiple-way partitioning problem. One way is to use the two-way partitioning algorithm to construct the partition progressively, i.e. going from the total stock of components, the parts of the partition are built successively. At each execution of the bi-partitioning algorithm, the components of the part under construction are chosen from the remaining stock of components in a way that the cost will be the best at the end. For illustration, we can build the first part by moving the first components into a cluster until the desired size is reached, then we apply a

two-way optimization algorithm between the new part and the rest of the system, and we process the same way for the further parts. Following this method, it might be simple to minimize the cost of the partition with the first parts. But as this is achieved by maximizing the connections inside the remaining set of components, it will be hard to obtain good partitions afterward. Another way to use a bi-partitioning algorithm to solve a multi-way partitioning problem is to make use of the two-way partitioning algorithm hierarchically. In this context, there are two methods for the hierarchical partitioning. The first method consists of finding a starting bi-partition and applying recursively the algorithm on each part until the desired number of parts is found. For example, for a n-parts partitioning, we can firstly split the system into two equal parts, optimize the cost of this partition and then we split each part into two other parts and apply the optimization on each new bi-partition and so forth until we obtain n parts. This method also suffers from the problems of the preceding one. Thereto, it might propagate the deficiencies of a bad result obtained in the first partitioning into the following partitioning operations, inducing important defaults for a large number of parts.

A further idea consists of building a start partition and apply a pairwise optimization among the parts of the partition using the two-way improvement technique. For example, based on the intuitive apprehension that, in order to remove a net from the cut set, the probability that more than one node will have to be moved should increase with an increasing number of parts, the algorithm of [90] was enhanced in [109] following this orientation and adapted to the multi-way partitioning of hyper-graphs. this solution tries to take advantage from the power of the level gains. This approach guarantees the conservation of the connectivity acquired in the starting partition, but it is still not better than the preceding ones concerning the overall quality of the partition. For illustration, if we reduce the number of signals exchanged between two given parts, this does neither guarantee a reduction of the overall inter-parts communication nor does it absolutely induce the reduction of the number of frames used for the overall system communication. In fact, the communication between two parts is not the matter of improvement in our problem. We are rather interested in the cumulative communication of a part with all its neighbors, since the frames used by a part to communicate contain the signals that are addressed to several neighbors. Because of the singular formulation of our problem, none of these solutions can be applied one-to-one to it. We need an original solution that is coupled with our cost function and that can optimize the partition uniformly. This solution must extend its view on the whole neighborhood of each candidate for a move, instead of considering only one neighbor during each move.

11.5.2 The improvement technique

The clustering algorithm (cf. algorithm 4) used the best next assignment method, i.e. each node was assigned to a part regardless of its closeness to the other nodes that were not in this part. We can improve the result of the best next assignment by moving the nodes from parts to parts in a way that improves the quality of the partition. The quality of a partition is given by the compactness of the clusters and the cost of the partition, i.e. the number of frames used for the inter-clusters communication. To uniformly improve the quality of the partition, it is necessary to consider the gain of moving each node from its actual part to all its possible destinations concurrently, so that each node is moved into the part to which it is at the highest bound. So we can achieve more compact clusters. But, these perturbations will be useful only if they do not deteriorate the cost of the partition. Thus, depending on the priorities assigned to the optimization goals, a good perturbation, i.e. one that increases the quality of the partition, might be a perturbation that results into more compact clusters and at least without a deterioration of the cost of the partition or better with an amelioration of the cost, or one perturbation that consumes fewer frames with the less augmentation of the compactness.

The possible destinations of a node v are its neighbor parts, i.e. each part that contains a node that is related with v, i.e. that exchanges signals with v or is bound with v by a Tag-relation. The binding of a node to a cluster determines the goodness or the badness of moving this node in

order to enhance the compactness of the cluster. Given a node v of a $CDFM$ model, its binding to a part P_i is measured by its closeness to the nodes v_{i_k} of P_i in relation with the number of nodes contained in P_i, i.e.:

$$Binding(v,P_i) := \frac{1}{|P_i|} \sum_{v_{i_k} \in P_i} Closeness(v, v_{i_k}) \tag{11.8}$$

The higher the binding of a node to a part, the closer is the node to this part. If a node has the same closeness value with different parts of the partitioning, then it is closer to the one with the least number of nodes than to any other of the rest. There might be some nodes in a part P_i that do not exchange any data with v, but that might be either Tag-related or not related with v at all. The formulation of the binding takes all these nodes in consideration, since the binding is based on the closeness metric that already takes the Tags into account and considers all the nodes of the given part. Based on equation 11.8, we can define:

- The internal binding $IBinding(v_{i_k}, P_i)$ of a node $v_{i_k} \in P_i$ as the binding of v_{i_k} to its actual home part and
- The external binding $EBinding(v_{i_k}, P_j)$ of a node $v_{i_k} \in P_i$ relative to a neighbor part P_j as its binding to this part.

Thus, since a node might have several neighbor parts, each node will have several external binding values too. A part that does not contain any node that is related with a node v has a binding value zero with v.

We also define the binding difference of a node v_{i_k} relative to a neighbor part P_j with:

$$DBinding(v_{i_k}, P_j) := EBinding(v_{i_k}, P_j) - IBinding(v_{i_k}, P_i) \tag{11.9}$$

As each node v_{i_k} might have several possible destinations P_j within a move operation, the best destination for a node will be the one to which it is best bound, i.e. $max_{P_j}(DBinding(v_{i_k}, P_j))$.

Each configuration of the partition must respect the *excludes* relations and the capacities of the parts. Clearly, a node cannot be moved into a part that excludes it. Theoretically, the best configuration is the one in which:

1. the *excludes* relations are respected,
2. the capacities of the parts are respected,
3. each node is assigned to the part to which it has the highest binding value and
4. the cost is optimal.

The first two points (i.e. 1. and 2.) must be observed by each move operation in order to guarantee the consistency of the partitioning with the design constraints. Furthermore, with such a move operation it cannot happen that all the nodes migrate into the same parts. The two last points (i.e. 3. and 4.) do not always behave in the same sense. The highest binding does not always imply the best frame sharing and vice versa. Thus, some trade-offs must be done between the compactness and the cost optimization. If our ultimate objective is to achieve the best cost with the partitioning algorithm, we also have to respect the strategic relationships as far as possible. For example, following the importance that we give to each of these two factors, we can classify them in a way that the partitioning prioritizes our goals. Anyhow, the improvement algorithm will depend on the order of preference assigned to the objectives above. Making trade-offs between the compactness and the cost of a partition imply that both must be measurable. We measure the compactness of a partition P having n parts as the average compactness over all the parts $P_i (1 \leq i \leq n)$ of the partition P.

$$Compactness(P) := \frac{1}{n}\sum_{i=1}^{n} Compactness(P_i) \tag{11.10}$$

The compactness of a part P_i of a partition P is defined by its (internal) binding value as follows:

$$Compactness(P_i) := \sum_{v_{i_k} \in P_i} IBinding(v_{i_k}, P_i) = \frac{1}{|P_i|} \sum_{(v_{i_k}, v_{i_l}) \in P_i^2} Closeness(v_{i_k}, v_{i_l}) \tag{11.11}$$

for all v_{i_k} and $v_{i_l} \in P_i$ with $v_{i_k} \neq v_{i_l}$.

11.5.3 The improvement procedure

The primary gaol of the improvement procedure is to optimize the cost of the actual partition, but this cannot be done at the cost of the compactness. Theoretically, the cost of a partition must decrease as the compactness increases. Obviously, the compactness of a partition increases if the nodes that are very closed are assigned to the same cluster. The improvement procedure follows this logic. It first moves the nodes in a way that increases the compactness of the clusters and then it searches the optimal cost. The nodes are sorted in a list following their individual capability to augment the compactness of the clusters and then moved in this order. For this reason, we call this procedure the List Assignment Procedure (LAP). The binding difference is used to measure the likelihood of a node to be moved. If a node is moved, only the binding values of its neighbor nodes will be affected by this move operation. This operational organization will strongly optimize the runtime complexity of the LAP. A move operation will affect only the compactness and the communication of the origin and the destination parts of the node moved, i.e. as follows:

$$Compactness(P_i + v) := Compactness(P_i) + EBinding(v, P_i) \tag{11.12}$$

$$Compactness(P_i - v) := Compactness(P_i) - IBinding(v, P_i) \tag{11.13}$$

The LAP (i.e. algorithm 7) searches the partition that provides the best cost with the best possible compactness. If two configurations provide the same cost, the LAP retains the one with the best compactness. The node the move of which has the highest potential to augment the compactness of the partition is moved into the closest part, called the best destination, i.e. the neighbor part to which the node has the highest binding. A node v is bound to a part only if there is no excludes relation between them and there is at least one node of this part that is related with v. If the capacity of a part is exceeded after a move operation, the LAP takes the nodes with the least binding out of this part and makes them occupy the free space created by the move in the origin part of the moved node.

Furthermore, the LAP has an intuitive appeal. It supposes that a single move operation can difficultly produce a perceptible impact on the quality of the partition. Thus, it might be necessary to operate many more moves than achievable within a single pass to obtain a measurable change on the cost of the partition. It thus continues to perturb each configuration that is not better than the actual best partition, hoping that several consecutive perturbations might ameliorate the partition. The number of consecutive perturbations is given by the user. This parameter is at the same time the termination condition for the algorithm. Thus, LAP terminates either because the given number of passes is achieved or because it has found the best configuration of the partition. Thank to its ability to process consecutive perturbations on provisional configurations, the LAP is able to jump from a hole. With this hill-climbing feature, the LAP will never be trapped in a local minimum of the cost, but will rather explore all the parts of the solution space before stopping. For the assignment of each node to its best destination, the LAP uses, as a further complexity optimization matter, a priority list that is always maintained sorted without a particular ordering

Algorithm 7 PROCEDURE ListAssignment (InitialConfig)

```
1: ActualConfig:=InitialConfig;
2: Compute Cost(ActualConfig);
3: Compute Compactness(P_i) for each part P_i of ActualConfig;
4: repeat
5:    Unlock all nodes v in ActualConfig;
6:    Extract the neighbor parts of each node v; //* Parts that content each at least one neighbor of v
7:    if NeighborParts(v):= {} then
8:       print "v HAS NO POSSIBLE DESTINATION"; //* v cannot be moved
9:       Lock v;
10:   else
11:      Compute DBinding(v,P_j) for each node v and for each neighbor part P_j of v;
12:      Lock all nodes v for which all DBinding(v,P_j) are negative; //* Home part is the best destination
13:      Place the highest positive DBinding(v,P_j) of each unlocked node v in a sorted queue Q with the highest DBinding(v,P_j) on the top; //* List of unlocked nodes corresponding to the best destination of each
14:   end if
15:   while there are unlocked nodes in the queue Q do
16:      Move the Top Node of Q to BestDestination(TopNode); //* the closest part of Top Node
17:      Remove this Top Node from the queue Q and lock it;
18:      Update the DBinding(v,P_j) values of its neighbor nodes and investigate the best destination of each;
19:      Lock each of them for which all DBinding(v,P_j) are negative; //* Home part is the best destination
20:      Place the highest positive DBinding(v,P_j) of each of them at the corresponding position in Q so that Q is maintained sorted with the highest DBinding(v,P_j) value on the top;
21:      if the capacity of BestDestination(TopNode) is exceeded after the move then //* Because of the move
22:         repeat
23:            Move the node v with the smallest positive DBinding(v,BestDestination(TopNode)) into the origin of Top Node; //* Conditional swap
24:            Lock it;
25:            Update Compactness(BestDestination(TopNode)) and Compactness(Origin(TopNode));
26:         until the capacity of BestDestination(TopNode) ≤ the size of (BestDestination(TopNode) ∪ Top Node);
27:      end if
28:   end while
29:   Compute Compactness(ActualConfig);
30:   Compute Gain(ActualConfig):= Cost(InitialConfig)-Cost(ActualConfig);
31:   if {Gain(ActualConfig) > 0} OR {Gain(ActualConfig)==0 AND Compactness(ActualConfig) ≥ Compactness(InitialConfig)} then
32:      InitialConfig:=ActualConfig;
33:   else
34:      Decrement the number of passes; //* This is an input parameter for the algorithm
35:   end if
36: until all the passes are executed;
37: return The InitialConfig;
```

effort. Thereto, after each move, only the binding values of the neighbors of the moved node and the compactnesses of the involved parts are updated so that the compactness of the partition that is found after a pass is computed simply by adding the most actual compactnesses of the parts. The update for a part consumes only one addition. Definitely, the control process of the LAP is quite simple. It allows the user to define the number of iterations of the algorithm.

11.6 Conclusion

Regarding the mapping, the cost of a partition is defined as the number of frames used for the inter-device communication. This is computed by packing the inter-device signals in the allocated frames. We follow the first fit decreasing procedure to do that. As the starting partition was built following a best next assignment method, we can improve it by moving the nodes from a part to another one in the sense that improves its quality. The quality of a partition is defined by its cost and its compactness. These two attributes do unfortunately not always behave in the same sense. To achieve the necessary compromises, we first observe the incidence of a potential move on the compactness of a partition before operating the move, hoping that it will not deteriorate the cost. In order to determine the best move for a node, the improvement procedure considers all its possible moves concurrently. A node is then assigned to the part to which it is best bound. The subsequent augmentation of the compactness of the partition likely reduces the number of inter-device signals, thus the cost of the partition. However, as only a succession of moves can produce a perceptible change on the quality of the partition, several moves are operated consecutively before the evaluation (i.e. in terms of costs) of the consequences of the perturbation on the quality of a partition. The configurations that are better than the initial configuration are stored as intermediate results. The LAP then compares these results at the end and picks out the one with the lowest cost and the highest compactness.

Chapter 12

Applications

In this chapter we discuss the applications of the design method provided in this thesis. As the functionality of the active cruise control (ACC) is distributed on several devices (see section 2.1.2), the ACC's presents a good choice to illustrate the profits induced by novel systems' architectures. We firstly present the ACC functionality to which we applied our partitioning algorithms. Then, we explain the objectives of our investigation in this application case and the scenario adopted for the investigation. In the following, we comment the investigation itself and then, we compare and comment the results obtained from the different architectures that were obtained from varying the configuration of the system's platform and the number of the required clusters.

12.1 The application case

12.1.1 Presentation of the application case

We applied our design method on the specification of the active cruise control (ACC) functionality that was previously introduced in the chapters 2, 7 and 8. As shown in figure 12.1, the functionality of the ACC is distributed on several devices that are interconnected through a high-speed CAN bus in the power-train and a lower-speed CAN bus in the body.

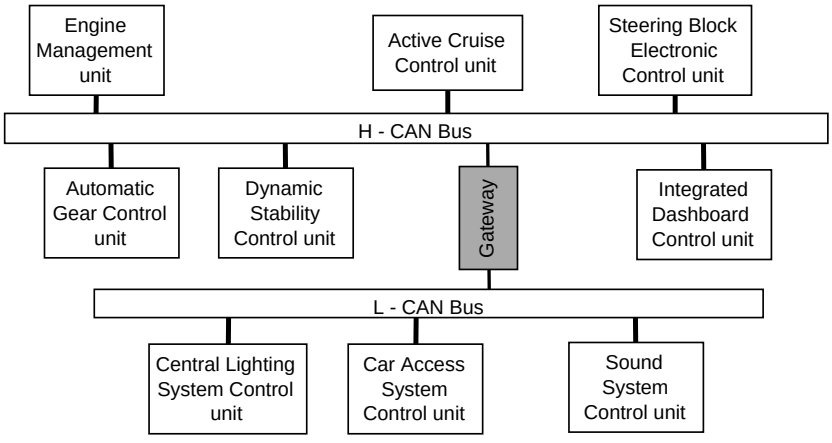


Figure 12.1: The ACC's CAN bus interconnection

In this architecture, we distinguish a device dedicated to the main functions of the ACC. This device is called itself ACC. It executes for example the obstacle sensing, the lane prediction and the security distance and speed control functions while the functions that are in charge of e.g. the acceleration, the deceleration and the throttle adjustment are located on the engine management unit (EEM). The automatic dynamic stability control unit (ASC) provides the trajectory information management, the manipulation of the hydraulic brakes and the velocity management functions

while the automatic gear control unit (EGC) provides the gear status management and the automatic gear-to-speed adjustment functions. The dashboard control unit executes the ACC-related visual and audio signalization for the driver. This includes e.g. the information about the ACC's activities, the announcements and requirements of the ACC for the driver, etc. The functions for the automatic activation of the brakes, the blinkers, the flash lights, etc. are located in the lighting control and the steering block control units.

12.1.2 Objectives and scenario

Basing on the implementation of the ACC presented above, our objective was to prove that better architectures could be found if the designers had the possibility to generate several solutions and to compare them. As we explained along this thesis, generating multiple architectures for such a complex system is practically not possible without a CAD support. With the method defined in this thesis, it will be possible for systems architects to generate several architectures easily and in a very short time, by varying for example the partitioning parameters, e.g. the set of output devices, the individual capacity of the different devices, the partitioning criteria, the priorities given to the objectives of the partitioning, etc. Thus, different architectures can be created and compared. Remember that the quality of an AES architecture was defined on the basis of the consumption of the communication frames and the compactness of the clusters of the functional components. Regarding the frames consumption, the best solution must consume the smallest number of frames. Regarding the compactness, highly compact solutions have a higher quality. The choice of the best solution depends on the objectives of the partitioning, for example on the decision about the quantity of space that must be reserved in the architecture of a system under construction in prevision to its future functional enhancements. Very compact partitions can difficultly be enhanced. Less compact solutions consume more frames. At this point of the design, the designer is free to decide the convenient trade-offs. An illustration of this decision-making is given in section 12.3.

Anyway, for each utilization of our partitioning method, the following steps are necessary:

1. The specification of the system under design must be modeled following the FN format.
2. The partitioning tool transforms this input into a synthesis model, i.e. in a $CDFM$ format.
3. The closeness metrics and the closeness function are calculated.
4. Using the closeness function, the synthesis model is pre-clustered and then clustered following the algorithms described in the procedures 2 and resp. 4 (chapter 10).
5. The value of the resulting partition is estimated with the cost procedure (of algorithm 6 in chapter 11).
6. And the improvement procedure defined in algorithm 7 (chapter 11) is started.

As the main goal of this application was to investigate the impact of our partitioning technique on the design of the system's architecture, we had to compare the solutions found with our algorithms with the existing original solution, considered as the reference solution. This is the architecture of the ACC presented in section 12.1. In this solution, the system described in section 12.1 was implemented on the devices shown in figure 12.1, i.e. as 9 clusters. This implementation uses 46 frames to communicate. Thereby, the part of the system that is implemented in the power train of the concerned vehicle uses 34 frames. This is the part around the high-speed CAN bus, i.e. 6 clusters. Assuming that the functions can be shifted from a device to another one, we compared different mapping options found with our partitioning algorithms. In order to investigate the impact of new architectures on the quality of the partition, we firstly considered that the system's architecture cannot be changed, i.e. we built 9 clusters just as in the reference partition,

but with different assignments. This solution is identified with "9 new" in contrast to the original "9 orig" solution of the reference architecture (in results table 12.2). Then, we began to decrease the number of clusters. Given the number of clusters, we firstly applied the clustering algorithm (i.e. algorithm 4) on the synthesis model ($CDFM$) of the input specification. Then, in order to fairly compare the different solutions,

1. we measured the number of signals that realized the inter-clusters communication, i.e. the number of signals derived from $\bigcup_i Tokens(P_i)$ where each $Tokens(P_i) := \left\{ T_{ij}^k, for\ each\ destination\ cluster\ j;\ k \in \mathbb{N} \right\}$
2. before applying the frames packing algorithm (i.e. algorithm 5) on these signals in order to obtain the number of frames that will be necessary to package and ship them.

Secondly, we applied the improvement algorithm on the solution found with the clustering algorithm, we counted the inter-cluster signals and then we applied once more the bin packing on them. These operations are presented with more details in the following sections and the results are given in section 12.3.

12.2 The investigation

12.2.1 The models

Figure 12.2 shows an extract of the graphical representation of the FN model of the ACC. The corresponding $CDFM$ format is shown in figure 12.3 with the original configuration of the mapping. The clusters are materialized in figure 12.3 by the dotted lines. As seen in figure 12.2, the FN is an intuitive model. The components are defined by their ports and connected through directed connectors (i.e. directed ports). Remember that each component shown in these figures is an atomic entity representing either a functional component or a data block of the ACC system's specification. Each port is associated with an interface. An interface specifies the information that is provided (or needed) by the associated ports. This is a typical modeling style used in the high-level design of complex systems. However, with the FN, the connectors clearly indicate the communication paths. But it is not easy to recognize the information that is exchanged through each connector. For example, when a single port is connected with several ports, i.e. associated with multiple connectors, a partitioner must analyze the contents of the concerned interfaces in order to associate the data elements to the connectors that materialize the corresponding communication. Thereto, the model has too many interfaces. These problems are solved in the $CDFM$ format.

The $CDFM$ representation of the system's functional specification is an undirected graph of atomic components in which each two communicating nodes are related by means of a single undirected edge (see definition of model transformation in section 8.3.2). The dashed rectangles represent each the communication interface of the two nodes that are related by the associated edge. The set of data objects contained in each communication interface represents the data elements that are exchanged between these nodes during an activation time of the system. These sets of data determine the weight of the corresponding edges as defined in section 8.2. At this level of the design, the inter-components communication data objects are observed like tokens and they also behave as such (see section 8.1.2). But for the data flow analysis, these data objects, i.e. these tokens, must be transformed into signals following the formula explained in equation 11.1. The figure 12.3 shows all the atomic components that have been mapped on the principal device involved in the functionality of the ACC, called itself ACC. One component that runs on the ACC sensor, i.e. the *Lens heater*, is shown. It is also shown that the system's functionality is distributed on several devices, e.g. the ASC, the EGC, the EEM, etc. that can be seen in the picture. But since the detailed knowledge of the actual implementation of the system is useless

for the understanding of the experimentation, the figure does not depict the whole ACC-related contents of the satellite devices. Note that there are more satellite devices (see figure 12.1) that are not shown in this picture.

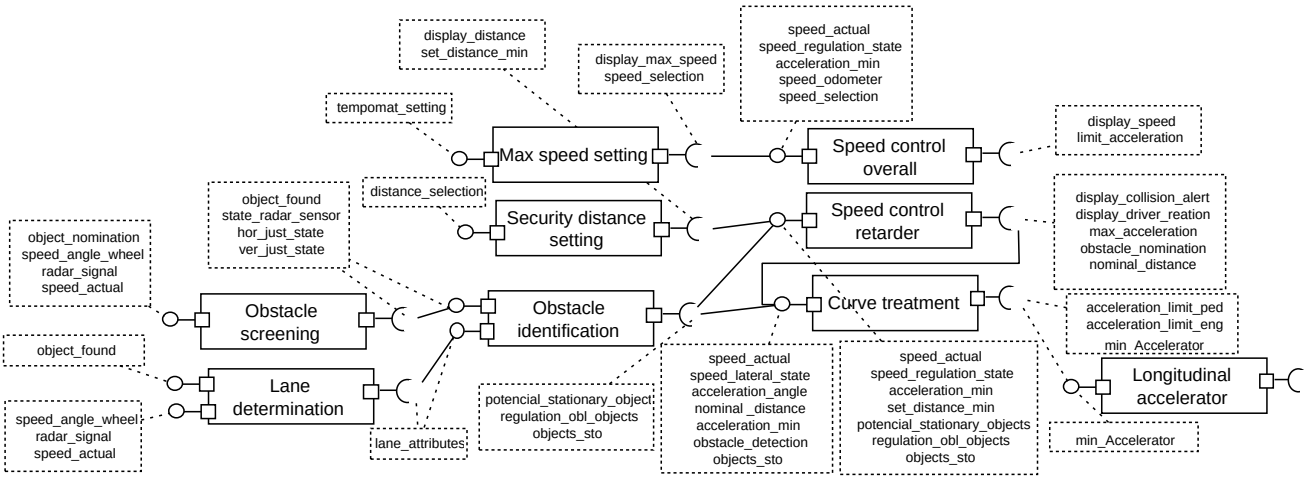


Figure 12.2: The *FN* specification of the ACC: Extract

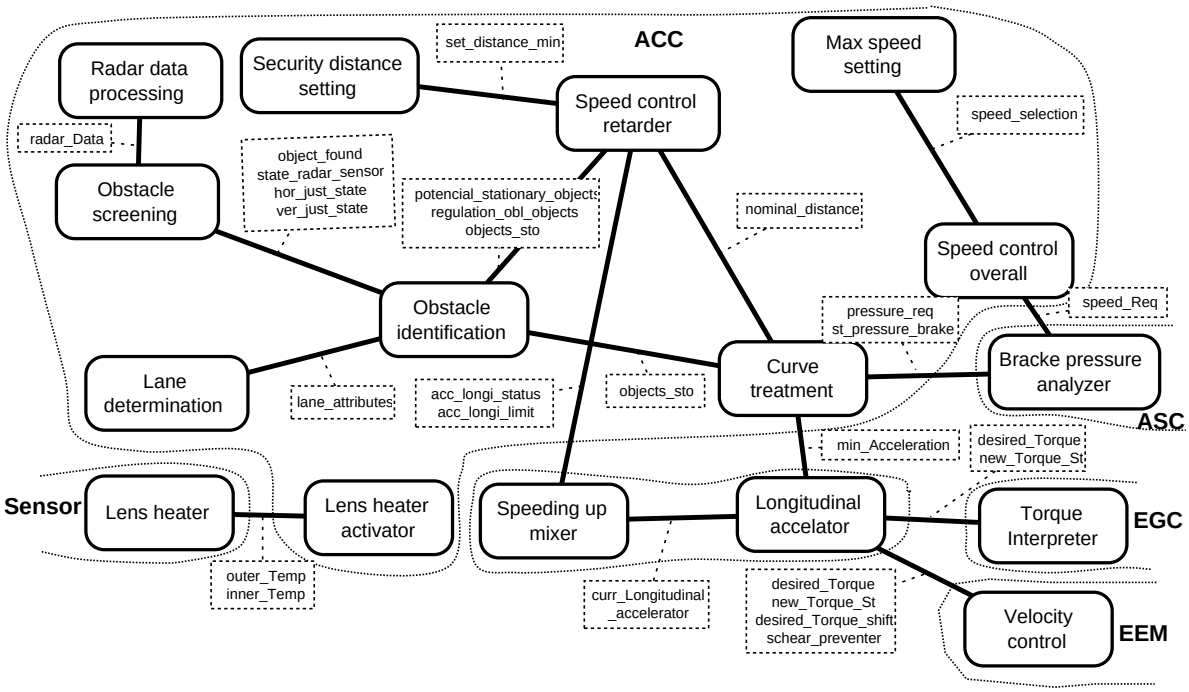


Figure 12.3: The corresponding *CDFM*-formatted specification of the ACC

Even with this fairly-reduced representation of the functional specification of the ACC, a visual exploration of the reference architecture (figure 12.3) shows that the distribution of the functional components of the ACC is not optimal, at least from the point of view of the inter-devices communication. For example, the functions *Max speed setting* and *Speed control overall* are located on the ACC device although they do not communicate with any other function of this device, but their communication partners are rather on the ASC, the dashboard and the steering block control units. This is a consequence of the device-oriented design process that is usual in the AES domain. Several reasons can justify this architecture, e.g. the ACC is a relative new feature that uses some functions that already exist in the system. Thus, basically if there is no justification for a redundant implementation of such functions, it is natural to confirm their actual assignment

and pick the results of their computations and to transmit them through the bus to the devices that need them. This approach results in an economy in the implementation of such functions. But, it induces the consumption of more frames for the inter-device communication. Moreover, this distribution of the functions can have negative consequences on the real-time behavior of the system.

12.2.2 The attributes of the tokens

Table 12.1 shows more ACC-related functions (v_i) with the outgoing tokens of each of them (i.e. Tokens T_{ij}^k) and their attributes. The destination of the tokens is not integrated in the table because of its lower relevance for the partitioning. In fact, as shown in figure 12.1, the devices that run these functions communicate through a CAN network. Thus, after the clustering, the parts of the system will communicate through such a frame-oriented communication system, in which the addressee of a frame is not considered, but rather the sender. However, note that the destinations of the tokens can be seen in figure 12.3. In the chapters 7 and 8, we defined the resolution, the frequency, the priority, the occurrence and the freshness attributes of $CDFM$ tokens. In table 12.1, the resolution ($T_{ij}^k.res$) is given in bits. The frequency ($T_{ij}^k.freq$), i.e. the period, and the freshness ($T_{ij}^k.fresh$) are given in milliseconds.

Using the formula 11.1 we can easily calculate the number of signals corresponding to each token. Remember that:

$$nbSignals(T_{ij}^k) = \begin{cases} 1 & \text{if } T_{ij}^k \text{ is short} \\ \left\lceil \frac{T_{ij}^k.res}{(maxframeL - segmentCode)} \right\rceil & \text{if } T_{ij}^k \text{ is long} \end{cases} \quad (12.1)$$

For illustration, each token that is equal to or shorter than 64 bits (payload of a CAN frame) is equivalent to one signal. This is the case for most of the tokens shown in table 12.1 (i.e. with $T_{ij}^k.res \leq 64\ bits$). Considering a *segmentCode* of 1 Byte, token "*promtOrderingSt*" (26th row of the table) will produce for example 2 signals (one with 7 Bytes and one with 5 Bytes).

We distinguished 4 classes of priority for the tokens, thus for the resulting signals:

- The priority class 1 defines the highest order of priority within the signals. In addition to vehicle deactivation signals, the priority class 1 includes highly prioritized ASAP signals such as the demand to activate the brakes when an obstacle is identified.
- The priority class 2 includes the vehicle activation signals as well as the signals announcing severe functioning irregularities, such as blind radar, etc.
- The classes 3 and 4 are the lowest priority classes. They contain the status monitoring signals, the alive confirmation signals and so on.

We defined three groups for the date of occurrence of the signals:

- The group A is reserved for the signals the first occurrence of which is immediately after the starting of the system. The starting of the system is defined as the time at which the electric circuit is firstly completed, e.g. when the driver turns the key in the ignition block.
- The group B is reserved for the signals that begin to appear after a certain time, e.g. some functions wake up only when the vehicle velocity reaches a given speed, e.g. *security dist setting* is activated only when the vehicle goes at 20 km/h minimum. Equally, some other functions wake up after the system has run for a given time.
- The group C is reserved for the event-triggered and sporadic signals.

All these signals must be mapped on the frames of the CAN communication network.

Table 12.1: The input ACC's specification for the application

Ids	Functions v_i	Tokens T_{ij}^k	$T_{ij}^k.res$	$T_{ij}^k.freq$	$T_{ij}^k.prio$	$T_{ij}^k.occur$	$T_{ij}^k.fresh$
1	**Radar data processing**	radarScreData	24	10	1	B	10
2	**Security dist setting**	setDistMin	16	10	2	B	1000
3	**Obstacle screening**	stateRadarSensor	4	10	3	B	1000
		horJustState	12	20	3	B	20
		verJustState	12	20	3	B	20
4	**Speed control retard**	regulOblObjects	24	10	2	B	10
		accLongStatus	4	20	3	B	60
		accLongiLimit	4	20	3	B	60
5	**Obstacle identification**	potentialStatiObj	16	10	2	B	20
		objectFound	2	10	1	B	10
		objectsSto	4	10	1	B	10
6	**Lane determination**	laneAttributes	32	10	1	B	10
7	**Lens heater**	outerTemp	8	100	2	B	300
8	**Lens heater activator**	innerTemp	8	1000	4	B	2000
9	**Speeding up mixer**	currLongAcc	8	20	2	B	40
		currAccOverall	8	20	2	B	20
10	**Curve treatment**	nominalDistance	24	10	1	B	10
		pressureReq	16	10	1	B	10
		minAcceleration	12	20	2	B	60
11	**Longitudinal accelerator**	longitudinalAcc	12	20	2	B	60
		desiredTorque	8	20	2	B	20
		desTorqueShifft	8	10	2	B	20
12	**Max speed setting**	speedSelect	8	10	1	B	1000
13	**Speed control overall**	speedStatus	8	10	2	B	1000
		speedReq	12	10	2	B	1000
14	**Brake pressure analyzer**	statPressBrake	10	10	2	A	200
15	**Torque interpreter**	newTorqueStat	8	10	2	A	200
16	**Velocity control**	currTorqueSt	8	10	2	A	50
		schearPreventer	12	20	3	B	20
17	**Clutch pedal sensor**	pedalSt	2	20	3	B	40
		pedalTensioner	8	10	3	B	30
		pedalNook	12	100	3	B	200
18	**Signal processing clutch**	levelSignaling	3	20	3	B	60
		operationSt	2	20	3	B	40
		levelReq	2	20	3	B	50
19	**Target area control**	radarOrientation	4	100	3	C	100
		angleSetting	8	100	3	C	100
20	**Wheel brake control**	brakingDemand	12	10	1	B	10
		clampLevel	3	10	2	B	100
		pressureIndication	2	10	2	B	10
21	**Wheel braker**	clampLevelSetting	4	10	2	A	20
		pressureSett	16	10	1	A	10
		slowdounInfo	8	10	4	B	1000
		demandResponse	1	10	3	B	200
22	**Engine interposition**	celerityReq	12	20	1	B	20
		torqueMomment	12	10	2	A	20
23	**Engine control**	aliveTorque	16	100	2	B	200
		minTorque	12	100	2	B	200
		revolutionMin	16	100	3	B	200
		revolutionFault	2	100	2	B	200
		speedRegulationSt	16	100	3	B	500
24	**Wheel rotation**	speedFR	8	20	3	B	100
		speedFL	8	20	3	B	100
		speedBR	8	20	3	B	100
		speedBL	8	20	3	B	100
25	**Wheel move balance**	tolerancebalFR	12	1000	3	C	
		toleranceBalFL	12	1000	3	C	
		toleranceBalBL	12	1000	3	C	
		toleranceBalBR	12	1000	3	C	
26	**Setting function**	promptOrderingSt	96	10	1	B	10
		on-off	4	100	2	A	20
27	**Display status**	displaySettingSt	4	10	3	A	200
		torqueLevel	8	10	2	B	20
		obstacleDist	8	10	2	B	30
		handoverReq	8	100	4	C	100
...	**...**	...	...	...	...	...	...

12.2.3 The partitioning

As we can observe in the models of the ACC introduced above, the strategic constraints and the resulting *needs* relationships between the elements of the functional model of the ACC have been already solved within the reference solution. For these reasons, the partitioning process done in this application did not need a pre-clustering. Furthermore, there was no *excludes* relation neither between the functions nor between the tokens. Thus, the functions that were partitioned (see table 12.1) are all atomic components, more precisely super-modules (see section 10.2), and the super-graph is free from *excludes* relations. Note that the resolution of *needs* and *excludes* relations is objectively not significant for the objectives defined in this investigation. Consequently, as the input model was free from *needs* and *excludes* relations, the closeness metric between two components was the number of exchanged signals, i.e. $Tag(e_{ij}) = 0,\ \forall i, j$ (see the definition of $Tag(e_{ij})$ in the section defining the closeness metrics, i.e. in section ClosenessMetrics. With these dispositions, the objective of the clustering was finally to cluster the functions depending on the heaviness of their communication. The results are reported in table 12.2.

For the determination of the quality of the partitions, remember that each frame provides a payload capacity (8 bytes for CAN frames) of user data that can be occupied by several signals, i.e. in the case of multiplexed frames. According to table 12.1, the period of validity (i.e the freshness requirement) of the most signals is longer than the required periodicity of their occurrences. Thus, these signals enjoy each a quite long individual *waitingtime* (cf. equation 11.3) that allowed a right flexible configuration of the groups $Signals(P_i).d_g$ defined in section 11.2 and thus a flexible packing of the signals in the frames. Remember that each set $Signals(P_i).d_g$ is a group of signals that can be shipped together, and thus can be packed together in the same frames. The signals contained in each set $Signals(P_i).d_g$ are all produced by the same cluster. But, note that except the frames used by the ACC device, most of the frames used by the other devices that contain the signals involved in the functioning of the cruise control might also contain several signals that are totally irrelevant for this functionality. To take this fact into account, it was sufficient to introduce a placeholder signal with the corresponding resolution for each such signal in the lists of signals to be packed, more precisely in the corresponding groups $Signals(P_i).d_g$ so that we could be sure that the number of frames resulting from the frames packing will fairly reflect the result obtained with the whole system. Fortunately, this disposition did not impose an important modification of the frames packing algorithm since the placeholder signals were defined with the same attributes, i.e. resolution, occurrence and frequency like the real signals that they were supposed to replace.

12.3 Results

Table 12.2 summarizes the results of our investigation. In the second row, the approximated augmentation (+) or diminution (-) of the number of signals realizing the inter-clusters communication is given in comparison to the original 9-clusters solution in percent. The third row of the same table documents the economy in the number of frames used to realize the inter-clusters communication for each investigated solution. The compactness of the partitions, calculated as defined in the equation 11.10 and given in the fourth row of the table of the results, was used to measure the opportunity to adopt a partition. With regard to the number of components in a cluster, we distinguished very low (vl), low (l), medium (m), high (h) and very high (vh) levels for the compactness. Depending on the range of the values found for the compactness, such a categorization might lead to several interpretations. But, with a basic knowledge of the AES architectural design domain, we could easily interpret the compactness levels in terms of the ability of the corresponding architecture to integrate the rest of the system or to leave room for their own functional enhancements.

Theoretically, the 1-cluster solution should be the optimal solution, i.e. the one with 0 external signal, 0 frame consumption and the highest compactness. But, as shown in table 12.2, this was not the case. The reason is quite simple: The ACC is not an isolated feature. In fact, in the

1-cluster solution, all the functions of the ACC are assigned to a single device that unifies the sensor with the rest of the functions involved in this functionality. Thus, for these functions, the inter-functions communication is realized within the device, but there is still a need to communicate with e.g. the actuators that cannot be located in this device. Furthermore, with this solution, several functions are detached from their "natural" home-devices and placed into the ACC device. Even so, these functions must communicate each with its home-device. This explains the poor compactness of the partition with the 1-cluster solution. In fact, even if the unified ACC cluster is very compact, the compactness of the other clusters is thereby so deteriorated that the global compactness of the system is dramatically penalized. Thereto, since the functions detached from their home must continue to communicate with their "brothers" and with other associated devices, this solution augments the number of frames. A similar tendency is observed for the 2-, the 3- and the 4-clusters solutions. The following partitions made economy of the number of inter-clusters signals and of the frames. Particularly, the 5-, the 6- and the 7-clusters partitions seam to yield the best costs, but with different levels of compactness.

Number of clusters	1	2	3	4	5	6	7	8	9 orig	9 new
Variation of the number of inter-clusters signals	+25	+18	+5	+11	-58	-42	-29	-16	0	-11
Variation of the number of frames	+13	+8	+6	+2	-11	-8	-6	-3	0	-3
Compactness of the partition	vl	vl	l	l	vh	h	h	m	m	m

Table 12.2: Impact of the partitioning on the cost of the architecture

As the quality of a partition is given by the compactness of the clusters and the cost of the partition (section 11.5.2), the 5-clusters solution seems to be the best solution. This solution uses some fewer frames than the 6-clusters solution, but its very high compactness indicates that it will probably not be a good partition when embedded in the rest of the AES from which the application case was extracted. Obviously, it will be easier to integrate the partition with 6 clusters in the rest of the system or to enlarge its functional contents than it will be with the 5-clusters solution.

12.4 Conclusion

The application shown in this chapter demonstrates the usefulness and the power of our partitioning solution. Depending on the tuning of the partitioning parameters, the partitioning tool implementing this method will report the different solutions and the designer is free to choose the one that satisfies its goals. Even if this example is sufficient to demonstrate the efficiency of our design solution, we agree that more investigations on different examples with different sizes, different attributes and different constraints would better support the experimentation of our solution. But, due to the severe policy of protection of intellectual property that is imposed in the automotive industry by the stringent conditions of the competition ruling the automobile market, there is no concrete example of AES, that can be entirely made public. The application case presented in this thesis is a concrete example.

Chapter 13

General conclusion

In this chapter, we firstly summarize the work we have done for this thesis, i.e. the identification of the problem, its comprehension, the extraction of its core issues, its presentation, its final definition and the solution proposed. Basing on a practical industrial observation, we identified the problem related with the need of a methodical and CAD-supported approach to the design of AES architectures. This problem was motivated and defined in this thesis as the combination of a partitioning problem and a bin packing problem, both formulated as multi-objective optimization problems. In order to enable rapid comprehension of the solution we proposed to this problem, we present our solution schema in the form of a design scenario. Since the solution targets specifically on the problem defined in this thesis, we also give an outlook on its possible expansions, for example for addressing more generally defined similar problems or for enlarging the application field of our solution. We thereby enumerate the questions that are related with the extensions of our solution, but that could not be treated in the defined range of the present doctoral dissertation. For these reasons, the outlook is intentionally given in such a way as to encourage interested researchers to confirm and fulfill the results presented in this thesis.

13.1 Summary

In the actual context that is characterized by the rapid evolution of the quantity of software- and electronic-actuated features in automobile E/E systems and the related hard competition in the automobile market, it is necessary to design optimal architectures for automobiles' E/E systems in a relative short time. In this thesis, we proposed to solve this problem by providing a CAD-supported method for the architectural design of automobiles' E/E systems. As settled in chapter 1, we defined the problem to be solved as a partitioning problem that we firstly formulated as follows: Given the functional specification of an E/E system and the related constraints, our aim is to provide a CAD-supported method to find the best architecture of this E/E system, i.e. the architecture that minimizes the usage of the hardware (i.e. processing units, memory elements, communication cables) and that concurrently optimizes the functioning and the quality (i.e. the performance, the reliability, etc.) of the resulting system. Following a procedural discussion on the above defined partitioning problem, we concluded, again in chapter 1, that the problem must be solved by a goal-oriented clustering of the system's logical functions, called the mapping, the result of which must define the logical devices of the system that must be deployed on the allocated components of the system's hardware platform. We then closed chapter 1 with the presentation of our approach to solve this problem, the intended scientific contributions and the organization of the detailed treatment of the problem.

After a profound study of the relations between software, electronic components and automobiles' E/E systems in chapter 2, we gave a clear and particularly complete definition of E/E systems as AES. Then, a comparative study of the different design processes that are usually followed in the design of both embedded systems and automotive E/E systems allowed us to define the system-

oriented approach to the architectural design of AES that is explained and analyzed in the chapters 3 and 4. Our design process follows a co-design approach, i.e. the system's components are concurrently designed. Since it allows a global view on the system, the co-design approach is the best solution to enable efficient partitioning of the system's functionality within the given platform. However, following a system-oriented design approach within a model-driven system development scheme, it was necessary to make sure that the input models satisfy the requirements of both the system-level design and the partitioning. In fact, while the system-level design needs abstract and coarse-granular models, the partitioning is done optimally with precise and fine-granular models. Thus, ideally, the right models must incorporate these contradictory features at the same time. As we were not aware of such an all-rounder modeling solution, we started an expansive study of the capabilities of the modeling solutions used in the AES design domain with the goal to evaluate their ability to support the system-level design on the first hand and the partitioning on the second hand. This part of the work included an overview of the state-of-the-art in modeling AES (chapter 5), the evaluation of the usual AES modeling solutions (chapter 6) and the proposition of a modeling solution that is optimized for the partitioning (in chapter 7). To achieve the latter, we defined the FN (definition 7.1) to formulate the input functional specifications of the AES under design. The FN is a modeling format that is based on AUTOSAR, SysML and EAST ADL concepts of atomic software components with ports, interfaces and connectors, but that is featured to enable easy tracing of the inter-components communication paths. We completed the FN with the HN (definition 7.2) for capturing the hardware platform characteristics (i.e. hardware components and their attributes) and so, we could give a formal definition of the partitioning problem addressed in this thesis in its original form (definition 7.3).

FN models allow to capture the system's functional architecture in a way that the closeness information between its components is highlighted. However, we rapidly noticed that despite its power, the FN could not allow the handling of the communication between the system's software components as needed for the partitioning, e.g. the extraction and the measurement of the heaviness of the communication. We therefore defined the synthesis model, called the $CDFM$, in chapter 8 to fill the drawbacks of the FN. With the synthesis model, each FN model that is given to the partitioning tool can be transformed into an undirected graph on which the common partitioning algorithms can be applied, enabling in this way a CAD-supported approach for the partitioning of AES system-level functional specifications. In order to allow the measurement of the closeness between the system's software components, we introduced the concept of tokens in the $CDFM$. $CDFM$'s tokens enable the analysis of the data flow within the AES functional models. They also enable the automatic extraction and the computation of the closeness between the functional components of the system. In addition to these features of the $CDFM$ enabled by the concept of tokens, $CDFM$ models are almost dynamical models. Each $CDFM$ model only needs to be meaningfully enhanced with the desired communication mechanisms and the corresponding protocols to be transformed into dynamical models the behavior of which can be simulated.

Nevertheless, following a more precise examination of the problem to be solved from the point of view of the quality of the input models, it was clear that with the chosen design process, the allocation of the hardware platform for an AES under design could not precise more than the sample of the available electronic devices (ECUs, sensors, actuators), their individual capacities, their cabling (inter-devices communication buses) and the associated communication protocols. Thus, the corresponding partitioning process must firstly execute the allocation, then the mapping and finally the deployment, whereas the allocation determines the sample of the devices that are available in terms of their individual capacities, i.e. the capabilities of the devices that can be used to implement the system's functionality. Note that a partitioning process typically proceeds a certain number of loops through these operations. However, since we cannot determine the capacities of the needed devices at the high-level of the design, we must take for granted that the allocation, just as the deployment, is given. Note that the deployment is the concern of the components supplier. The deployment deals with the scheduling of the tasks on the processing units and similar low-level design activities. Consequently, the partitioning problem to be solved

in this thesis was reduced to the mapping, i.e. the optimal distribution of the system's software components to the allocated devices (figure 13.1).

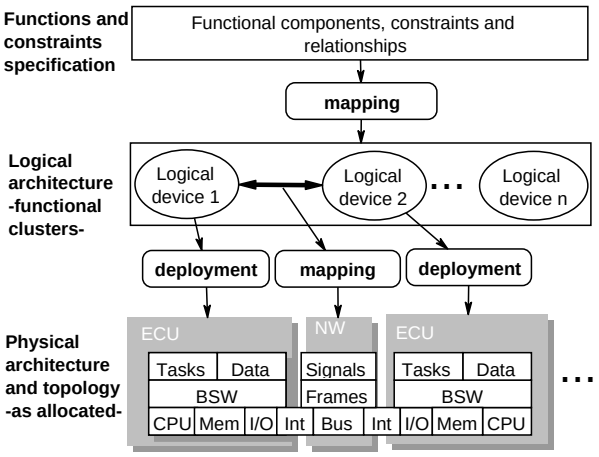


Figure 13.1: The partitioning

Since the input functional specifications (i.e. the FN models) are made of inter-communicating atomic software components like those defined in AUTOSAR models, each component can be assigned to a device only entirely or not and two components that are assigned to different parts of the system communicate over networking bus as shown in figure 13.2. Because of the general characteristics of the most automotive communication protocols (see section 2.2), it is assumed that in the final system, the system's devices will communicate through a bus network that runs a frame-oriented communication protocol and probably makes use of a frames multiplexing technique. A highly representative example of this class of protocols, the CAN, was introduced in section 11.1 for more comprehension of frame-oriented communication and frame multiplexing.

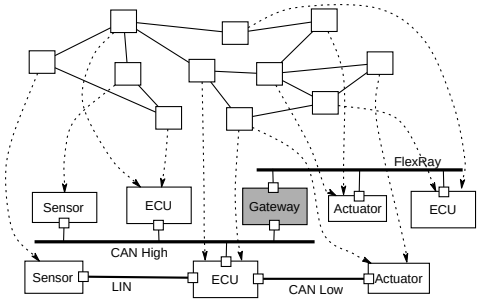


Figure 13.2: The mapping

As a consequence to these conditions, the primary goal of the mapping became the minimizing of the inter-devices communication, i.e. the minimizing of the number of frames used to exchange the information through the buses. These conditions enforced an adjustment of the problem definition that aimed at keeping our work conform with the reality and at the same time stay close to our declared goals. We then reformulated the partitioning problem as follows:

Problem formulation 13.1. (**The partitioning**) Given A a FN functional specification of an AES, given a sample of the capacities of the platform's devices on which the specification of A can be implemented, find the best partition of A, each part representing a device that will communicate with the others through a bus network running a frame-oriented communication protocol, so that the bus load is minimal, the functioning and the performance of the system are guaranteed and the strategic constraints of the design are fulfilled.

Our solution to this problem can be roughly resumed in the two following steps: Given such a model A,

1. We firstly transform A into a $CDFM$ model.
2. Then, we find a partition of the $CDFM$ model of A that minimizes the number of frames used for the inter-devices communication and hopefully also minimizes the need of hardware units within the devices under the observation of the given design constraints.

Remember that minimizing the usage of the hardware units within the devices is the duty of the deployment. Each FN model can be automatically transformed into a $CDFM$ model following the transformation rules described in chapter 8. To realize the partitioning, i.e. the mapping, we defined quite conventional but specific partitioning algorithms that take the strategic concerns of the design into account. Our solution to this complex multi-objective optimization problem is given in the third part of this thesis. It begins with the synopsis of the state-of-the-art in the resolution of partitioning problems that is given in chapter 9. Then, our partitioning strategy is presented in chapter 10. It follows three clearly distinguished steps including the pre-clustering, the clustering and the improvement of the result of the clustering. The pre-clustering is the procedure by which the *needs*-relations established between the model elements are solved. The software components that must be implemented together on the same device are related by the means of *needs* relations in $CDFM$ models. Similarly, those that might never be implemented on the same device are related by the means of *excludes* relations. Note that *needs* and *excludes* relations also exist between the tokens for similar reasons. The pre-clustering procedure merges the software components of a given $CDFM$ model that need each other into super-modules and thus, results in a super-graph of super-modules and super-edges as defined in section 10.2.

To find an effective clustering, the components of the functional specification must be assigned to the allocated devices following their closeness. Depending on their relative closeness to each other, the clustering procedure groups the nodes of the super-graph produced by the pre-clustering into logical devices following the process defined in section 10.3. Each such logical device is a cluster of software components that are so close to each other that the result of their actual distribution within the logical architecture of the system is supposed to minimize the inter-devices communication by concurrently fulfilling the practical design constraints, for example those concerning the production conditions of the system's components, the conditions of their procurement, the technical factors of their implementation, the composition of the existing system, the factors influencing the marketing of the products, etc. The closeness metrics used for the clustering are defined on the basis of the quantification of the *needs* and the *excludes* relations and of the inter-components data flow analysis enabled by the definition of the tokens. We formally defined a closeness function that combines several closeness factors including the magnitude of the inter-components communication, their potential to share the resources and the relationships induced by the constraints and the strategic concerns of the design between them. The clustering procedure itself is realized by a QT algorithm that takes the *excludes* relations into account. The QT assignment strategy was investigated in chapter 9 as the best solution for a partitioning problem for which the number of clusters is not pre-defined, but rather the possible dimensions of the resulting clusters. The result of the clustering is a set of clusters of software components that will be assigned each unbroken to a device of the platform.

As the clustering procedure follows a best-next assignment strategy, there was no possible statement in relation with the quality of the resulting partition. We consequently decided to solve this problem by means of an improvement process, i.e. a process that allows to make a statement on the optimality of a partition. This problem is solved in chapter 11. In order to explain the usage of the communication frames, we begin this chapter with the presentation of the idea of frames multiplexing and its consequences on the partitioning and we introduced it in section 11.1 with the example of the CAN. Logically, before improving a partition, it is necessary to evaluate it. We addressed the evaluation of a partition in section 11.2. As we defined the cost of a partition in terms of the number of frames used for the inter-devices communication based on a protocol that makes use of frames multiplexing, the cost function for the evaluation of a partition came out to be a bin packing problem (section 11.2.2). We designed a bin packing algorithm based on the FFD

(First Fit Decreasing) packing strategy to investigate the cost of the partitions and we explained it in section 11.4.2. The choice of the FFD strategy follows the discussion given in section 11.3 at the end of which the FFD was investigated as the packing strategy that at the best copes with the requirements of the cost investigation procedure.

Finally, we investigated the possibility to improve the quality of a partition. Based on the conclusions of the related discussion (resumed in section 11.5.2), we defined the improvement technique used to ameliorate non-optimal partitions. The improvement procedure is implemented by a powerful adaptation of the K&L principles of perturbations based on move operations. In reality, excepting the idea of moving nodes, our improvement algorithm behaves very differently with the K&L algorithm. For example, in contrast to the 2-way partitioning for which the K&L algorithm was designed, our algorithm solves a multi-parts partitioning problem. Thereto, in order to ameliorate the cost of a given partition, it moves the software components individually from a part to another one instead of swapping them between the parts as processed by the K&L solution. This definition of the move operation is justified by the fact that, firstly the dimensions of the parts must not be equal and secondly the number of the parts is not pre-defined as a fix number. A good partition can necessitate less devices than the preceding one. Further innovations of our improvement algorithm include the sequencing of the move operations, the choice of the nodes to be moved, the order in which these nodes are moved and the mechanisms by which the overloading of the allocated devices is prevented. Another particularity of this algorithm is its ability to conserve the established *excludes* relations. The improvement algorithm returns a set of clusters (i.e. parts, logical devices) and the corresponding set of optimally packed frames.

The quality of the solution presented in this thesis for the partitioning problem of AES is demonstrated in chapter 12. The partitioning process described in this thesis has been successfully applied to the functional specification of the ACC. Note that even though this application is based on the CAN, the solution developed in this thesis can be used with all frame-oriented communication networks. Anyway, although the time complexity of the partitioning algorithms for the architectural design of AES as defined in this work is not of significant relevance as we stated in section 9.1.2, our algorithms are proved to yield good performance and they are highly flexible, since they leave room for user interaction at every step of the partitioning and they can be used in many different cases depending on the actual definition of the problem. For example, if the number of devices is predefined instead of the capacity of the available devices or if the number of frames assigned to each device is given instead of giving a total number of available frames that must be shared between the devices, etc. Such particular cases can be perceived as the outlook for the enhancement of the work presented in this thesis.

13.2 Outlook

For the application that is presented in chapter 12, we considered a closeness function that combines the communication metric and the relationships induced by the strategic and the design constraints between the components of the input model of the ACC (i.e. defined as Tags in section 10.3.1). With regard to the results, this simple and unburdened closeness function was sufficient to demonstrate the efficiency of our design method. Thus, an optimal configuration of the design artifacts and a globally optimized adjustment of the design parameters will certainly produce better solutions. For example, some configurations of the partitioning of AES the investigation of which might be interesting include (the list is not exhaustive):

- The variation of the closeness metrics: As discussed in section 10.3.1, we can modify the definition of the communication metric, the resource sharing metric, the constraints and strategic relationships metric, etc. and investigate the impact of their variation on the performance of the partitioning.
- The variation of the closeness function: Different combinations of the closeness metrics will probably induce different consequences on the performance of the partitioning. It might be

also interesting to research the optimal weighting factor for each closeness metric in relation with different combination factors, etc.

- The experimentation of different communication protocols: We considered in this work the CAN as an example for the frame-based communication protocols used in the AES domain, but we used the related concepts very abstractly, i.e. independently of the particularities of the concrete protocols. It is certainly interesting to investigate the effects of such particularities on the quality of the partitioning method. For example, how is the outcome of the partitioning with a protocol following a different frame multiplexing technique?

- The observation of different optimization goals: Different classifications of the closeness factors induce different priorities of the optimization goals. A manipulation of this partitioning factor will produce different solutions. Depending on the closeness factors considered and the order of priority given to each of them, the result of the partitioning will be differently appreciated in terms of the quality. We can imagine a field of points in a geometrical space representing each such a configuration of the partitioning from which a solution front (e.g. a Pareto front) must be identified.

- The consideration of larger systems: As discussed in section 12.4, it will be interesting to apply this partitioning method with its different variations on a very large system in which the closeness criteria will be more globally considered.

Each of these points is a full topic of investigation for itself that cannot be effectively covered within a single doctoral thesis. However, as we have provided the basic theoretical work for solving the partitioning problem of AES in this thesis, we believe that the next steps in this area will be easier. We thus strongly encourage interested researchers to joint us in investigating the open points enumerated above. Moreover, we think that similar research, including the investigation of the usefulness of our solution, for the design of the architectures of other kinds of distributed systems that must be agile and adaptable (dynamically or not) or that are required to evolve rapidly and to be constantly actualized or that have to share moving software agents, etc. might lead to very interesting conclusions.

Appendix A

Zusammenfassung der Dissertation

Partitionierungsorientierte Modellierung und Architekturdesign von automobilen E/E-Systemen

Autor: Augustin Kebemou

Vor dem Hintergrund der gegenwärtigen rapiden Zunahme von software- und elektronikgetriebenen Bestandteilen in modernen Fahrzeugen und der damit verbundenen harten Wettbewerbssituation im Automobilmarkt ist es notwendig, optimale Architekturen für automobile E/E-Systeme (Elektrik/Elektronik) in relativ kurzer Zeit zu entwerfen. In dieser Dissertation schlagen wir vor, dieses Problem durch eine CAD-gestützte Methode zu lösen.

A.1 Motivation

Heutzutage müssen Autohersteller schöne, verlässliche und sichere Fahrzeuge mit kraftvollen Motoren, robuster Technik und hohem Komfort bauen. Diese Anforderungen werden durch den Einsatz von eingebetteten elektronischen Steuergeräten erfüllt. Automobile E/E-Systeme sind sehr komplexe Systeme, deren softwarebasierte Funktionalitäten auf mehrere Steuergeräte (ECUs, Sensoren und Aktoren) verteilt sind, die nicht nur zusammenarbeiten, sondern auch voneinander abhängen. Wegen der wachsenden Nachfrage nach mehr Komfort, Sicherheit und Umweltverträglichkeit ist die Entwicklung dieser E/E-Systeme eine zentrale Aufgabe für Autohersteller geworden. Für jedes neue Fahrzeug muss entschieden werden, welche Steuergeräte und wie viele von jeder Sorte zum Einsatz kommen. Gegenwärtig werden diese Designarbeiten von hoch erfahrenen Systemarchitekten, die üblicherweise Systemintegratoren genannt werden, manuell durchgeführt. Die so entworfenen Fahrzeuge funktionieren zufriedenstellend, aber die manuelle Suche nach der richtigen Architektur, d.h. die richtigen Komponenten, die beste Topologie des Systems und die beste Nutzung der Leistung des Systems, ist zeitaufwendig und erfahrungsabhängig. Denn sie ist mehr eine Kunst als eine Ingenieurleistung. Sie ist deshalb auch kein Garant dafür, dass am Ende die beste Architektur gefunden wird. Im Gegenteil, diese kreative Arbeit erfordert ein optimistisches Vorgehen in der Praxis. Die Architekten arbeiten nach ihrem Gefühl und hoffen, dass das Ergebnis trotzdem gut ist. Die risikoärmste Vorgehensweise, die leider am weitesten verbreitet ist, ist die folgende: Um neue Funktionalitäten in einem Fahrzeug zu implementieren, fügen die Systemarchitekten dem bereits existierenden System neue Steuergeräte hinzu, ohne es zu verändern.

Diese vorsichtige Vorgehensweise hat den Vorteil, dass sie die Menge der Arbeit, die die Integration von neuen Funktionen erfordert, verringert. Sie lässt sich also mit der Tatsache begründen, dass die existierenden Systeme gut funktionierende und verlässliche Konfigurationen mit stabilen Kommunikationsmatrizen sind. Ein neues Design des Systems würde eine Entwicklung von Grund auf bedeuten, die natürlich noch mehr Zeit in Anspruch nehmen würde und deshalb wirtschaftlich ungünstig

wäre in diesem Industriebereich, in dem die Entwicklungszeit entscheidend ist. Sie ist also eher realistisch als pragmatisch. Aber sie hat viele negative Konsequenzen für die Wettbewerbsfähigkeit der OEMs. Dies drückt sich z. B. durch die erhöhte Anzahl an ungenutzten Hardwareressourcen in den Fahrzeugen und die daraus folgenden hohen Verkaufspreise und Nutzungskosten für die Kunden aus. Auf dem Weg zu kommerziellen Alleskönnerfahrzeugen werden noch mehr software- und elektronikgetriebene Funktionalitäten in automobilen E/E-Systemen hinzugefügt. Dies wird sich wieder auf die Designkosten, die Fahrzeugpreise, die Entwicklungszeit, die Betriebskosten (Energie-, Kraftstoffverbrauch, Instandhaltung, etc.) und die Qualität (Leistung, Sicherheit, Verlässlichkeit, etc.) der Fahrzeuge, auswirken. Diese Perspektive stellt eine große Herausforderung für die Autohersteller dar. Können wir weiterhin so viele Steuergeräte wie neue Funktionalitäten einbauen? Wenn nicht, wie hoch ist das Risiko beim Entwurf völlig neue Architekturen? Was würde das eigentlich kosten? Was würde uns dann die Erfahrung unserer Architekten kosten? Dies sind einige der Fragen in Zusammenhang mit dieser Problematik. Einfach gesagt kann die Herausforderung wie folgt formuliert werden: Da wir die ständig wachsenden Forderungen der Verbraucher nach Komfort-, Sicherheits- und Umweltverträglichkeitsfunktionen in ihren Fahrzeugen nicht stoppen können, wie können wir weiterhin verlässliche und sichere Fahrzeuge schnell herstellen, die jede mögliche Funktion bieten, aber dennoch kostengünstig sind?

Die Antworten auf diese Frage beinhalten im Allgemeinen die folgenden Vorschläge: Eine radikale Änderung der Entwicklungsmethoden, die Standardisierung von Systemkomponenten, die Harmonisierung von Designprozessen und die Entwicklung neuer Technologien.

- Eine Veränderung des Designprozesses ist notwendig: Der aktuelle Designprozess basiert auf einem komponentenorientierten Ansatz, was zu einem Übermaß an in dem System installierten Komponenten und einer starken Kommunikation zwischen den Komponenten führt. Um mehr Funktionen auf weniger Komponenten zu implementieren, ist es notwendig, die derzeitige komponentenorientierte Entwicklung zugunsten einer systemorientierten Entwicklung aufzugeben. Dies haben alle Automotiv Designer bereits erkannt, und das ist auch gut so.

- Eine Standardisierung sowie Harmonisierung der Prozessen sind erforderlich: In dieser Industrie, in der die gute Zusammenarbeit zwischen Herstellern und Zulieferern besonders wichtig ist, in der die Basisfunktionen so identisch sind, dass sie unter den Herstellern austauschbar sind, in der ein Produkt im Allgemeinen nur eine Variante eines anderen ist, ist der Bedarf an einer klaren Basis für die Kommunikation, einer systematischen Wiederverwendung von Lösungen und einer profitablen Wettbewerbsplattform offensichtlich. AUTOSAR und FIBEX sind vielversprechende Initiativen, die sich um eine Standardisierung innerhalb der Automotiv-Domäne bemühen.

- Das Aufkommen neuer Technologien wird mit Sicherheit einige Probleme lösen: Prominente Beispiele hierzu umfassen die sich herausbildenden Mechatronik- und By-wire-basierten Innovationen und die neuen automobilen Kommunikationsprotokolle wie TTCAN, TTP, MOST, FlexRay, Byteflight, LIN, etc.

Obwohl diese Vorschläge das Problem scheinbar lösen, reichen sie offensichtlich nicht aus! Um z.B. den Bedarf an den stetig wachsenden Kommunikationsanforderungen zu decken, benötigen E/E-Designer immer leistungsfähigere Kommunikationssysteme. Dies wird einerseits durch die Entwicklung von leistungsstarken und spezialisierten Kommunikationsprotokollen erreicht und andererseits durch das Hinzufügen von neuen Bussen/Kabeln in das System. Kabel hinzuzufügen bedeutet, das Kabelgeflecht zu erweitern. Dies wiederum bedeutet eine Zunahme des Gewichts, also des Energieverbrauchs, die Erhöhung des Verkaufspreises der Wagen und möglicherweise weitere Quellen für technische Probleme. Wir können die Kosten optimieren, wenn wir sowohl die Designkosten als auch den Hardwareressourcenverbrauch reduzieren. Die Designkosten können wir dadurch reduzieren, dass wir effiziente und CAD-gestützte Designtechniken zur Verfügung stellen, womit auch unerfahrene Designer gute Architekturen entwerfen können. Dafür müssen wir das Architekturdesign,

das gegenwärtig eher eine Kunst ist, in eine Ingenieurdisziplin umwandeln. Den Hardwareverbrauch können wir reduzieren, indem wir die Systemarchitekturen so optimieren, dass die Menge an Hardwareressourcen, die für die Implementierung der erforderlichen Funktionalitäten benötigt wird, reduziert wird. Dies kann dadurch erreicht werden, indem wir die Prozessoren und die Memories, die in dem System installiert sind, sowie die Kabel für die Kommunikation zwischen den Steuergeräten optimal nutzen.

A.2 Problemlösung

Unser Ziel war es, eine automatisierbare Methode für das Design der Architekturen von E/E-Systemen vorzuschlagen. Diese Methode soll helfen, unabhängig von der Erfahrung und von dem Gefühl des Designers innerhalb kurzer Zeit eine optimale Architektur für jedes E/E-System zu finden. In dieser Dissertation haben wir das zu lösende Problem als ein Partitionierungsproblem definiert, das wir zunächst wie folgt formulierten: Anhand der funktionalen Spezifikation eines E/E-Systems und der entsprechenden Design-Constraints, gesucht wird die optimale Architektur, die die gegebenen Anforderungen erfüllt. Die optimale Architektur ist die, die gleichzeitig die Verwendung von Hardwareressourcen (Prozessorenleistung, Datenspeichern, Kommunikationskabel, etc.) minimiert und die Qualität (Leistung, Verlässlichkeit, etc.) des daraus entstehenden Systems optimiert. Die optimale Architektur zu finden heißt dann, die minimale Anzahl der benötigten Hardwarekomponenten zu ermitteln, die kostengünstigsten Hardwarekomponenten auszusuchen, von jeder die kostensparendste Kalibrierung zu bestimmen und die vorhandenen Ressourcen am wirkungsvollsten auszunutzen. Dies natürlich unter Berücksichtigung der gegebenen Designanforderungen und Constraints und mit der Garantie, dass das daraus entstehende System verlässlich funktioniert. Dies ist ein schwieriges Optimierungsproblem. Verlässlichkeit und sicheres Funktionieren sind bei minimalen Ressourcen offensichtlich schwer zu erreichen.

Nach der verfahrenstechnischen Diskussion über dieses Problem haben wir in Kapitel 1 geschlussfolgert, dass das Problem durch ein zielorientiertes Clustering der Funktionen des Systems gelöst werden kann. Ein Clustering bildet das sogenannte Mapping ab, dessen Ergebnis die logistischen Komponenten des Systems darstellt. Kapitel 2 beschäftigt sich mit der Definition von automobilen E/E-Systemen. Es enthält eine tiefgreifende Untersuchung der Beziehungen zwischen Software-, elektronischen Komponenten und automobilen E/E-Systemen. In den Kapiteln 3 und 4 erlaubte uns eine Vergleichsstudie der verschiedenen Designprozesse, die üblicherweise sowohl beim Design eingebetteter Systeme als auch von E/E-Systemen eingesetzt werden, einen systemorientierten Prozess zu definieren, der das Design von optimalen Architekturen für E/E-Systeme unterstützt. Da dieser Prozess einem Co-Design-Ansatz folgt, benötigt es System-Level-Modelle. Folglich müssten wir angesichts der Komplexität von E/E-Systemen zunächst sicherstellen, dass die Inputmodelle die Anforderungen des System-Level-Designs und der Partitionierung erfüllen. Denn, während System-Level-Modelle meistens abstrakt und von grober Granulierung sind, erfordert die Partitionierung genaue und feine Modelle. Die Inputmodelle für das System-Level-Clustering müssen idealerweise diese widersprüchlichen Voraussetzungen gleichermaßen erfüllen.

Da wir eine solche Allround-Modellierungslösung nicht kannten, haben wir eine ausführliche Untersuchung der Modellierungslösungen, die für das modellbasierte Design von E/E-Systemen benutzt werden können, durchgeführt. Das Ziel der Untersuchung war die überprüfung der Fähigkeit der vorhandenen Modellierungstechniken, die Aktivitäten des System-Level-Designs einerseits und der Partitionierung andererseits zu unterstützen. Dieser Teil der Dissertation umfasste eine Übersicht über den Stand der Technik in der Modellierung von E/E-Systemen in Kapitel 5, die Bewertung der üblichen Modellierungslösungen in Kapitel 6 und in Kapitel 7 den Vorschlag einer Modellierungslösung, die für die Partitionierung optimiert ist. Unsere Modellierungslösung, genannt FN für Functional Network, definiert eine Syntax für die Formulierung der Funktionalität eines E/E-Systems. Das FN ist ein Modellierungsformat, das auf AUTOSAR-, SysML und EAST ADL-Konzepten (z.B. atomare Softwarekomponenten mit Ports und Connectors, Interfaces, etc.) basiert, das aber die Ausfilterung der Kommunikationswege zwischen den Software-Komponenten erleichtert. Wir komplettierten das

FN mit dem *HN* (Hardware Resource Network), um die Hardwareplattform zu erfassen. Mit diesen Angaben konnten wir eine formale Definition des Partitionierungsproblems liefern. Wir bemerkten jedoch, dass das *FN*-Modellierungsformat trotz seiner Leistungsfähigkeit die Kommunikation zwischen den Software-Komponenten nicht so darstellen kann, wie es für die Partitionierung optimal ist.

Wir definierten daher ein Synthesemodell in Kapitel 8, genannt *CDFM* für Components Data Flow Machine, um die Nachteile des *FN* auszugleichen. Mit dem Synthesemodell kann jedes *FN*-Modell in einen ungerichteten Graph umgewandelt werden, auf den die allgemeinen Partitionierungsalgorithmen angewendet werden können. Um die Analyse des Datenflusses zwischen den Software-Komponenten und die Ermittlung der Brüderlichkeiten zwischen ihnen zu ermöglichen, haben wir das Konzept der Tokens in das *CDFM* eingeführt. Die Tokens sind so definiert, dass außer ihrer optimalen Fähigkeit, den Datenfluss abzubilden, sie aus *CDFM*-Modelle leicht Verhaltensmodelle machen können. Jedes *CDFM*-Modell braucht nur mit den gewünschten Kommunikationsmechanismen und den entsprechenden Protokollen versehen zu werden, um in ein dynamisches Modell umgewandelt zu werden. Nichtsdestotrotz war nach genauerer Prüfung des zu lösenden Problems hinsichtlich der Qualität der Modelle klar, dass mit dem gewählten Designprozess die Definition der Hardwareplattform nicht präziser sein kann als die Angabe der verfügbaren Steuergeräte mit ihren Kapazitäten, ihrer Verkabelung (Busse) und der dazugehörigen Kommunikationsprotokolle. Daher musste sich der Partitionierungsprozess auf das Mapping, d. h. die optimale Verteilung der Applikationssoftwarekomponenten eines E/E-Systems auf die zugewiesenen Steuergeräte, konzentrieren.

Da die funktionale Spezifikation des Systems aus atomaren Softwarekomponenten besteht, kann jede Komponente nur ganz oder gar nicht einem Steuergerät zugewiesen werden. Zwei Softwarekomponenten, die verschiedenen Steuergeräten zugewiesen werden, können nur über einen Netzwerkbus kommunizieren. Aufgrund der allgemeinen Merkmale der meisten automotiven Kommunikationsprotokolle wird angenommen, dass die Kommunikation über die Busse nach Frame-orientierten Protokollen erfolgt, die bestimmte Frame-Multiplexing-Techniken nutzen. Um diese Konzepte zu klären, haben wir das Funktionieren eines prominenten Beispiels dieser Klasse von Protokollen in Kapitel 11, nämlich den CAN, zusammengefasst. Als Konsequenz dieser Bedingungen war das Ziel des Mappings das Minimieren des Verbrauchs der Kommunikationsressourcen, das wir formal durch die Anzahl der verbrauchten Frames formalisiert haben. Diese Orientierung erforderte eine Anpassung der Definition des zu lösenden Problems, die zur folgenden Formulierung führte: Gegeben seien $\mathscr{A}$, eine *FN*-funktionale Spezifikation eines E/E-Systems, die Kapazitäten der einzelnen Steuergeräte, auf die $\mathscr{A}$ implementiert werden kann, die Anforderungen und die Constraints des Designs, gesucht wird die beste Aufteilung der Elemente von $\mathscr{A}$ über die vorhandenen Steuergeräte, sodass die Buslast minimal ist, die Funktionalität des Systems gewährleistet ist, die Leistung optimal ist und die strategischen Bedingungen des Designs erfüllt sind.

Unsere Lösung dieses Optimierungsproblems ist in dem dritten Teil dieser Dissertation beschrieben. Sie kann grob in den zwei folgenden Schritten zusammengefasst werden: Gegeben sei ein *FN*-Modell $\mathscr{A}$,

1. Wir wandeln zunächst $\mathscr{A}$ in ein *CDFM*-Modell um,

2. Dann, unter Berücksichtigung der vorgegebenen Einschränkungen, suchen wir eine Partition des *CDFM*-Modells von $\mathscr{A}$, das die Anzahl der Frames, die für die Kommunikation verwendet wird, minimiert.

In dieser Lösung kann jedes *FN*-Modell automatisch in ein *CDFM*-Modell umgewandelt werden. Um die Partitionierung zu realisieren, haben wir spezifische Partitionierungsalgorithmen entworfen, die die strategischen Gegebenheiten des Designs berücksichtigen. Die Partitionierung ist wie folgt dokumentiert: Kapitel 9 zeigt eine Übersicht zum Stand der Technik bei der Lösung von Partitionierungsproblemen. Unsere Partitionierungsstrategie wird in Kapitel 10 beschrieben. Sie folgt drei klar definierten Stufen: Pre-Clustering, Clustering und Optimierung des Clusterings. Die Softwarekomponenten, die zusammen auf das gleiche Steuergerät implementiert werden müssen werden durch Needs-Beziehungen miteinander verbunden. Diejenigen, die niemals auf das gleiche Steuergerät

implementiert werden dürfen, sind durch Excludes-Beziehungen miteinander verbunden. Das Pre-Clustering ist die Operation, die die Needs-Beziehungen zwischen den Modellelementen löst, indem es die Softwarekomponenten eines *CDFM*-Modells, die in Needs- Beziehung zueinander stehen, zu Supermodulen bündelt und so einen Supergraph ergibt. Das Clustering gruppiert die Supermodule in Abhängigkeit von ihrer relativen Distanz zueinander in logistische Steuergeräte. Die dafür benutzten Closeness Metriken und die Closeness Funktion sind in Kapitel 10 formalisiert. Das Clustering selbst wird durch einen Quality-Threshold Algorithmus (QT) realisiert, der die Excludes-Beziehungen berücksichtigt. Die Optimierung des Clusterings ist in Kapitel 11 beschrieben. Sie ist durch eine Anpassung des Kernighan&Lin-Algorithmus implementiert. Kapitel 12 beschreibt die Anwendung dieser Lösung auf das Architekturdesign des ACC (Active Cruise Control). Die englische Zusammenfassung der Dissertation steht in Kapitel 13.

A.3 Wissenschaftliche Beiträge der Dissertation

Das Ziel dieser Dissertation war es, ein automatisierbares ingenieurmäßiges Vorgehen für die Bestimmung der Architekturen von E/E-Systemen zu produzieren. Diese Operation wird in der AUTOSAR-Terminologie ”System-Generation” genannt. Wir haben dafür einen Designprozess definiert, die notwendigen Modellierungskonzepte herausgearbeitet, die Verhältnisse zwischen den Komponenten eines E/E-Systems formalisiert, und wir haben effiziente, CAD-gestützte Partitionierungsalgorithmen entworfen. Insgesamt können wir festhalten, dass diese Dissertation einen weiten Überblick über die Entwicklung von automobilen E/E-Systemen bietet. Dabei können wir die folgenden Punkte hervorheben:

1. Eine ausführliche Hintergrundinformation über das Wesen der automobilen E/E-Systeme und ihrer Entwicklung. Diese umfasst eine besonders gründliche Untersuchung der Spezifikation, der Modellierung und des Architekturdesigns von automobilen E/E-Systemen und der damit verbundenen Designprozesse und dokumentiert den aktuellen Stand der Technik, die aktuellen Anforderungen und die aktuellen und zukünftigen Herausforderungen bei der Entwicklung von E/E-Systemen.

2. Eine Studie der Wechselwirkungen zwischen E/E-Designprozessen, -Modellen, -Abstraktionsniveau und -Designoperationen.

3. Der Entwurf eines Frameworks für die Bewertung und die Klassifizierung von Modellierungssprachen, die in das Design von automobilen E/E-Systemen eingesetzt werden können. Dieser umfasst eine eingehende Prüfung und eine Klassifizierung der wichtigsten Modellierungssprachen, die in der Automotiv-Domäne verwendet werden, hinsichtlich ihrer Fähigkeit, die Implementierung komplexer eingebetteter Systeme zu unterstützen.

4. Die Definition von Modellierungslösungen für die Abbildung von E/E-Systemen, die die Partitionierung unterstützen.

5. Der Entwurf eines originalen Modellierungsformats für die Synthese von E/E-Systemen, das auch für Validierungstätigkeiten, d. h. für Simulation, Verifizierung oder Test, verwendet werden kann.

6. Die Definition eines Partitionierungsprozesses für das Architekturdesign von E/E-Systemen. Dies umfasst die Formalisierung von komplexen Closeness-Faktoren und die Entwicklung von leistungsfähigen Clustering- und Optimierungsalgorithmen.

7. Die Definition eines Verfahrens für die Bewertung von E/E-Architekturen mit den entsprechenden Metriken.

Glossary of Terms and Abbreviations

A

ABS Anti-Block System.

ACC Active Cruise Control.

ADC Analog Digital Converter.

AES Automotive Embedded Systems.

ASC Automatic Stability Control.

ASIC Application-Specific Integrated Circuit.

ASIP Application-Specific Instructions set Processor.

ASSP Application-Specific Standard Product.

AUTOSAR AUTomotive Open System Architecture.

AWF Almost Worst Fit.

B

BF Best Fit.

BFD Best Fit Decreasing.

Business-Critical Systems Best efforts on the availability and security requirements.

C

C2d Chrysler Collision Detection.

CAD Computer Aided Design.

CAN Control Area Network.

CDFM Components Data Flow Machine.

CPU Central Processing Unit.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

CSMA/CR Carrier Sense Multiple Access with Collision Resolution.

D

DSC Dynamic Stability Control.

DSP Digital Signal Processing.

E

EA Evolutionary Algorithm.

ECU Electronic Control Unit.

E/E Electric/Electronic.

EEM Electronic Engine Management.

EGC Electronic Gear Control.

EP Evolutionary Programming.

ES Embedded Systems.

ESs Evolution Strategies.

F

FF First Fit.

FFD First Fit Decreasing.

FN Functional Network.

FPGA Field Programmable Gates Array.

G

GA Genetic Algorithm.

GPP General Purpose Processor.

H

HDL Hardware Description Language.

HN Hardware resources Network.

I

I2C Inter-Integrated Circuits Bus.

ILP Integer Linear Programming.

K

K&L Kernighan & Lin Algorithm.

L

LCD Liquid Crystal Display.

LIN Local Interconnect Network.

LP Linear Programming.

M

MCU Micro-Controller Unit.

MILP Mixed Integer Linear Programming.

Mission-Critical Systems Best efforts on the functionality.

MOC Model Of Computation.

MOST Media Oriented System Transport.

N

NF Next Fit.

NoC Network-On-Chip.

NRE Cost Non-Requiring Engineering Cost.

P

POF Plastic Optical Fiber.

Q

QT Quality Threshold.

R

RTE RunTime Environment.

RTOS Real-Time OS.

S

SA Simulated Annealing.

SAE Society of Automotive Engineers.

Safety-Critical Systems Best efforts on the safety requirements, reliability, real-time.

SoC System-On-Chip.

SPI Serial Peripheral Interface.

T

TDMA Time Division Multiple Access.

TTCAN Time-Triggered communication on CAN.

TTP Time-Triggered Protocol.

U

UART Universal Asynchronous Receiver-Transmitter.

USB Universal Serial Bus.

V

VFB Virtual Functional Bus.

W

WF Worst Fit.

List of Figures

1.1 A subset of a modern passenger car's architecture: A network of sub-networks (source: Elektronik Automotive 01/2005, pp. 82) . . . 2
1.2 Activities in the AES development . . . 4
1.3 The partitioning with coarse-granular software components . . . 6
1.4 The allocation, the mapping and the deployment . . . 7
1.5 Detailed allocation and deployment . . . 8
1.6 Vertical logic of the thesis . . . 14

2.1 ECU architecture . . . 17
2.2 Automotive embedded systems . . . 17
2.3 The ACC radar sensor . . . 18
2.4 The ACC device . . . 18
2.5 ACC interconnection . . . 19
2.6 Some automotive in-vehicle communication technologies . . . 26

3.1 Sequential embedded systems design method . . . 27
3.2 Concurrent design method of embedded systems . . . 29
3.3 Specification languages in the HW/SW co-design method . . . 30
3.4 Design of the embedded software . . . 31
3.5 Usual processing components . . . 32

4.1 OEM and suppliers views in the development process of AES . . . 36
4.2 The components-based design process . . . 36
4.3 System-oriented design approach . . . 38
4.4 The AUTOSAR design approach (source: Autosar web content V23.4) . . . 39

5.1 AES design conceptual levels . . . 45
5.2 AES FAs basic modeling concepts . . . 46
5.3 AES hardware platforms basic modeling concepts . . . 46
5.4 Abstraction levels of the hardware devices . . . 47

7.1 AUTOSAR communication patterns . . . 69
7.2 The FN inherits the common concepts from the standards . . . 71
7.3 A partial graphical representation of the ACC's functionality following the FN format 72
7.4 The AUTOSAR ECU architecture (source: Autosar web content V2.0.1) . . . 73
7.5 Overview of the AUTOSAR layered ECU architecture . . . 74
7.6 Hardware infrastructure of a device . . . 75
7.7 Formal definition of the partitioning of automotive E/E systems . . . 76
7.8 Data object is a generalization of operations and data elements . . . 78

8.1 The FN graphical representation of the mileage inquiry . . . 88
8.2 The graphical CDFM representation of the mileage inquiry . . . 88
8.3 Model transformation: The $CDFM$ representation of the FN model of figure 7.3 . 89

9.1 Taxonomy of the partitioning methods 94

10.1 The partitioning process 103
10.2 The super graph 107

11.1 Format of a standard CAN data frame 117
11.2 Semantical relationships between the concepts for the frames multiplexing 118

12.1 The ACC's CAN bus interconnection 133
12.2 The FN specification of the ACC: Extract 136
12.3 The corresponding $CDFM$-formatted specification of the ACC 136

13.1 The partitioning 143
13.2 The mapping 143

List of Algorithms

1 PROCEDURE Partitioning(G: CDFM) 105
2 PROCEDURE Pre-Clustering(G: CDFM) 107
3 PROCEDURE InitialConfiguration(SuperModules) 112
4 PROCEDURE Clustering(SuperModules) 112
5 PROCEDURE FramesPacking($Signals(P_i).d_g$) 126
6 PROCEDURE Cost(P) 126
7 PROCEDURE ListAssignment (InitialConfig) 131

List of Tables

6.1 Programming languages, HDLs and ADLs are complementary 62
6.2 Level of support of AES modeling languages with regard to the cost of use 64
6.3 Level of support of AES modeling languages for the partitioning 65

12.1 The input ACC's specification for the application 138
12.2 Impact of the partitioning on the cost of the architecture 140

Bibliography

[1] *CAN Specification 2.0A, 2.0B available at www.can – cia.org.*

[2] *CAN Specification Version 2.0, Part A, Part B; Robert Bosch GmbH.*

[3] *DIRECTIVE 2000/53/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 18 September2000 on end-of life vehicles; - Official Journal of the European Communities; 21.10.2000.*

[4] *DIRECTIVE 2002/95/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 January 2003 on the restriction of the use of certain hazardous substances in electrical and electronic equipment; - Official Journal of the European Union; 13.2.2003.*

[5] *DIRECTIVE 2002/96/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 January 2003 on waste electrical and electronic equipment (WEEE): - Official Journal of the European Union; 13.02.2003.*

[6] *The Esterel v7 Reference Manual Version 7.30 - initial IEEE standardization proposal; Esterel Technologies, 3 Novembre 2005, France.*

[7] *I2C Bus Manual, Philips Semiconductors, March 24, 2003.*

[8] *ITU-T Serie Z; Languages and General Aspects for Telecommunication Systems; Formal Description Technics- Specification and Description Language (11/99).*

[9] *LIN Specification Package, Revision 1.3; Dec 13, 2002*
LIN Consortium available at $http://www.lin-subbus.org/$.

[10] *LIN Specification Package, Revision 2.0; Sep 23, 2003*
LIN Consortium available at $http://www.lin-subbus.org/$.

[11] *MOST Specification Rev 2.4, 05/2005, Most Cooperation*
www.mostcooperation.com.

[12] *Systems Modelling Language (SysML) Specification, OMG document: ad/2006-03-01; version 1.0 Draft.*

[13] *UML Profile for AUTOSAR; V1.0.0; AUTOSAR Adminstration web content, 28.04.2006.*

[14] *Universal Serial Bus Specification, Revision 1.1, September 23, 1998*
www.usb.org.

[15] *Universal Serial Bus Specification, Revision 2.0, Apri 27, 2000*
www.usb.org.

[16] AADL. http://www.aadl.info/.

[17] E. Aarts and J. Korst. Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. *Wiley-Interscience series in discrete mathematics and optimization John Wiley and Sons, Inc., New York, NY.*, ACM Computing Surveys 35:516., 1994.

[18] D.H. Ackley. *A Connectionist Machine for Genetic Hill Climbing*. Kluwer, 1987.

[19] A. Albert. Comparison of Event-Triggerd and Time-Triggered Concepts with Regard to Distributed Control Systems. *Embedde World 2004, p235-252*, 2004.

[20] C. J. Alpert and S.-Z. Yao. Spectral Partitioning: The More Eigenvectors, The Better. In *32nd ACM/IEEE Design Automation Conference*, 1995.

[21] P. Arato, Z. A. Mann, and A. Orban. Algorithmic Aspects of Hardware/Software Partitioning. In *ACM Transactions on Design Automation of Electronic Systems, Vol. 10, Nr. 1, pp 136-156*, Jan 2005.

[22] AUTOSAR. www.autosar.org.

[23] A. Baghdadi, N.-E. Zergainoh, W.O. Cesario, and A.A. Jerraya. Combining a Performance Estimation Methodology with a HW/SW Codesign Flow Supporting Multiprocessor Systems. In *IEEE Transactions on SW Engineering, Vol. 28, No 9*, Sep 2002.

[24] H. Bauer, J. Crepin, and K.-H. Dietsche. *Autoelektrik Autoelektronik; Systeme und Komponenten*. Vieweg, 2002.

[25] G. Berry and G. Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, pages pp 87–152, 1992.

[26] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee. *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.

[27] Lee Bilung and E.A. Lee. Hierarchical concurrent finite state machines in Ptolemy. In *Fist International Conference on Application of Concurrency to System Design, ACSD; pp 34 - 40*, March 1998.

[28] P. Binns, M. Engelhart, M. Jackson, and S. Vestal. Domain-Specific Software Architectures for Guidance, Navigation and Control. *International Journal of Software Engineering and Knowledge Engineering*, Vol 2, No 2, 1996.

[29] C. Bobda. *Synthesis of Dataflow Graphs for Reconfigurable Systems using Temporal Partitioning and Temporal Placement*. PhD thesis, Universitat Paderborn, Heinz Nixdorf Institut, Entwurf Paralleler Systeme, Germany, 2003.

[30] K. Buchenrieder and C. Veith. A Prototyping Environment for Control-Oriented Hardware/Software Systems Using States Charts, Activity Charts and FPGA. *EURODAC*, 1994.

[31] T.N. Bui and B.R. Moon. A Genetic Algorithm for a Special Class of the Quadratic Assignment Problem. In *DIMACS Series on Discrete Mathematics and Theoretical Computer Sciences, Vol 16, pp. 99-116*, 1994.

[32] Inc: BCANPSV2.0/D Rev. 3 CAN Bosch Controller Area Network (CAN) Version 2.0; Protocol Standard, Frescale Semiconductor.

[33] E.G. Coffman, M.R. Garey, and D.S. Johnson. Bin Packing with divisible Item Sizes. *J. Complexity*, 3:405–428, 1987.

[34] E.G. Coffman, M.R. Garey, and D.S. Johnson. *Approximation Algorithms for Bin Packing: A survey*, chapter 2 in Approximation Algorithms for NP-Hard Problems, pages 46–93. PWS Publishing, Boston, 1996.

[35] Luis Alejandro Cortes, Petru Eles, and Zebo Peng. Verification of Real-Time Embedded Systems using Petri Net Models and Timed Automata. In *8th International Conference on Real-Time Computing Systems and Applications (RTCSA 2002), Tokyo, Japan, pp. 191-199*, Mar 18-20, 2002.

[36] Luis Alejandro Cortes, Petru Eles, and Zebo Peng. A Petri Net Based Model for Heterogeneous Embedded Systems. In *17th IEEE NORCHIP Conference, Oslo, Norway, pp. 248-255*, Nov 8-9, 1999.

[37] J. Csirik and G.J. Woeginger. On-line Packing and Covering Problems. *Online Algorithms: The State of the Art, Lecture Notes in Computer Science, Springer, Berlin*, Vol. 1442:pp. 147178, 1998.

[38] A. L. Davis and R. M. Keller. Data Flow Program Graphs. *Computer*, Vol. 15, Nr. 2:26–41, Feb. 1982.

[39] Bruce Powel Douglass. *Real-Time UML. Developing Efficient Objects for Embedded Systems*. Addison-Wesley, 1998.

[40] EAST-EEA. Embedded Electronic Architecture. Definition of Language for Automotive Embedded Electronic Architecture v. 1.02. Technical report, ITEA, 30.06.2006.

[41] S. Edwards, L. Lavagno, E.A. Lee, and Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Models, Validation, and Synthesis. *Report*, Nov, 1999.

[42] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. In *Journal on Design Automation for Embedded Systems, vol. 2, pp. 5-32*, 1997.

[43] R. Ernst, J. Henkel, and T. Brenner. Hardware/Software Cosynthesis for Microcontrollers. *IEEE Design and Test of Computers*, pages pp 64–75, Dec 1993.

[44] A. Bauer et al. AutoMoDe - Notations, Methods, and Tools for Model-Based Development of Automotive Software. *SAE International*, 2005.

[45] U. Freund et al. Architecture Centric Modeling of Automotive Control Software. *SAE*, 2003.

[46] K. Etschberger. *Controller-Area-Network. Grundlagen, Protokolle, Bausteine, Anwendungen*. Hanser, 2002.

[47] ASAM e.V. FIBEX Field Bus Exchange Format; ASAM AE MCD-2[FBX] Version 2.0.1; 16.04.2007. *www.asam.net*.

[48] W. Fernadez da la Vega and G.S. Lueker. Bin Packing can be solved within $1 + \varepsilon$ linear time. *Combinatorica*, 1:349–355, 1981.

[49] C.M. Fiduccia and R. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. *Proceedings of the 19th Design Automation Conference*, pages 175–181, 1982.

[50] D.K. Friesen and M.A. Langston. Analysis of a Compound Bin Packing Algorithm. *SIAM Journal of Discrete Mathematics*, 4:61–79, 1991.

[51] Daniel D. Gajsk and Loganath Ramachandran. Introduction to High-Level Synthesis. In *IEEE Design & Test; Volume 11 , Issue 4; pp 44 - 54*, Oct 1994.

[52] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice Hall, 1994.

[53] D. D. Gajski and L. Ramachandran. Introduction to High-Level Synthesis. In *IEEE Design and Test of Computers, Vol. 11, No. 4, pp. 44-54*, Dec, 1994.

[54] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. ISBN 0-7167-1045-5, 1979.

[55] Michael R. Garey and David S. Johnson. A 71/60 Theorem for Bin Packing. *Journal of Complexity*, Vol. 1:pp. 65106, 1985.

[56] M.R. Garey, R.L. Graham, D.S. Johnson, and A.C. Yao. Resource Constrained Scheduling as Generalized Packing Problem. *Journal of Combinatorial Algorithms, Ser. A*, 21:257–298, 1976.

[57] D. Garlan, R. Monroe, and D. Wile. ACME: An Architecture Description Interchenge Language. In *Proceedings of CASCON'97*, Nov 1997.

[58] P.C. Gilmore and R.E. Gomery. A linear Programming Approach to the Cutting Stock Problem, Part 1 and 2. *Operational Research*, 1961 and 1963.

[59] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Co., Inc., Redwood City, CA., 1989.

[60] Hassan Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley, 2000.

[61] J. Gong, D. Gajski, and S. Bakshi. Model Refinement for Hardware/Software Codesign. *ACM Transations on Design Automation of Electronic Systems, Vol. 2, No 1, pp 22-41*, jan 1997.

[62] R.L. Graham. Bounds on Multiprocessing Anomalies and related Packing Algorithms. *Proc. of Spring Joint Computer Conference, Montvale, AFIPS Press*, pages 205–217, 1972.

[63] R. K. Gupta and G. DeMicheli. System Level Synthesis Using Reprogrammable Components. In *Proceedings of European Design Automation; pp 2-7*, 1992.

[64] L.W. Hagen and A.B. Kahng. New spectral methods for ratio cut partitioning and clustering. In *IEEE Trans. on CAD of Integrated Circuits and Systems, Vol. 11, Nr. 9, pp. 1074-1085*, 1992.

[65] B. Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of Operations Research, 13*, 2:311–329, 1988.

[66] R. W. Hartenstein. *A Comparison of Hardware Description Languages*, chapter 2 in Advances in CAD for VLSI, vol 7. Elsevier Science Publishers B.V., 1987.

[67] J. Henkel, T. Brenner, R. Ersnt, W. Ye, N. Serafinov, and G. Glawe. A Software-Oriented Approach to Hardware/Software Codesign. *Journal of Computer and Software Engineering 2(3), pp 293-314*, 1994.

[68] J. Henkel and R. Ernst. A Path-Based Technique for Estimating Hardware Run-time in Hw/Sw-Cosynthesis. In *Proceedings of the Eighth International Symposium on System Synthesis, Cannes, France, pp 116-121*, 1995.

[69] J. H. D. Herrmann, J. Henkel, and R. Ernst. An Approach to the Adaptation of Estimated Cost Parameters in the COSYMA System. In *Proceedings of the 3 rd International Workshop on Hardware /Software Codesign - CODES/CASHE '94, pp 100–107,*, 1994.

[70] J. Holland. Genetic Algorithms. In *Scientific american, pp. 44-50*, July 1992.

[71] J. H. Holland and al. Adaption in Natural and Artificial Systems. In *Press, Ann Arbor, MI., University of Michigan*, 1975.

[72] Farnam Jahanian and Aloysius K. Mok. Modechart: A Specification Language for Real-Time Systems. In *IEEE Transactions on Software Engineering, Vol 20, No 12*, Dec 1994.

[73] A.K. Jain, N.N Murty, and P.J. Flynn. Data Clustering: A Review. In *ACM Computing Surveys, Vol. 31, No. 3*, September 1999.

[74] A. A. Jerraya and al. *Multilanguage Specification for System Design*, chapter 3 in System-Level Synthesis, pages 103–135. Kluwer Academics Publishers, 1999.

[75] A.A. Jerraya, H. Ding, P. Kission, and M. Rahmouni. Behavioral Synthesis and Component Reuse with VHDL. *Kluwer Academic Publishers, Boston/ London / Dortrecht*, 1996.

[76] David S. Johnson, Alan J. Demers, Jeffrey D. Ullman, M. R. Garey, and Ronald L. Graham. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. In *SICOMP, Volume 3, Issue 4*, 1974.

[77] D.S. Johnson. Fast Algorithms for Bin Packing. *Journal of Computer and System Sciences*, 8:272–314, 1974.

[78] D.S. Johnson, C.R. Aragon, and al. Optimization by Simulated Annealing: An experimental evaluation; Part I , Graph Partitioning. *Operations Research*, 37:865–892, 1989.

[79] R.K. Jurgen. *Automotive Electronics Handbook, Second Edition*. Mc Graw-Hill-Handbooks, 1999.

[80] M. Karmakar and R.M. Karp. An efficient Approximation Scheme for the one-dimensional Bin Packing Problem. *In proc. 23 rd. Annual Symposium on Foundations of Computer Science*, pages 312–320, 1982.

[81] A. Kebemou. Partitioning Metrics for improved Performance and Economy of Distributed Embedded Systems. *IESS proceedings on IFIP TC10 Working Conference, pp 289-300*, Aug. 15-17 2005.

[82] A. Kebemou, M. Feldo, and A. Borusan. Formale Spezifikation des Structurierten Anforderungmanagements fur das Domain Engineering anhand eines Beispiels aus der Automobilindustrie. Technical report, Fraunhofer ISST, Berlin; Bericht 69, 2003.

[83] A. Kebemou and I. Schieferdecker. AutomotiveArchitect: A Partitioning-Centric Modeling and Architectural Design Environment for Automotive E/E Systems. In *Comming 2008*.

[84] A. Kebemou and I. Schieferdecker. The Components Data Flow Machine: An Intermediate Modeling Format for the Design of Automotive E/E Systems Architectures. In *Comming 2008*.

[85] A. Kebemou and I. Schieferdecker. Evaluating Modeling Solutions on their Ability to Support the Partitioning of Automotive Embedded Systems. *International Conference on Embedded and Ubiquituous Computing, Taipei, Taiwan*, 2007.

[86] B. Kernighan and S. Lin. An efficient Heuristic Procedure for Partitioning Graphs. In *Bell System Technical Journal*, Feb. 1970.

[87] Scott Kirkpatrick, D. Gelatt Jr., and M.P. Vecchi. Optimization by Simmulated Annealing. *Science*, 220:671 – 680, 1983.

[88] P. Kogut and P.C. Clements. Features Analysis of Architecture Description Languages. In *Proceedings of the Software Technology Conference (STC'95), Salt Lake City*, April 1995.

[89] K. Koutsougeras, C. A. Papachristou, and R. R. Vemuri. Data Flow Graph Partitioning to Reduce Communication Cost. In *Proceedings of the 19th annual workshop on Microprogramming; pp 82 - 91*, 1986.

[90] B. Krishnamurthy. An Improved Min-Cut Algorithm for Partitioning VLSI Networks. *IEEE Trans. Comput.*, vol C-33:pp 438–446, May 1984.

[91] P.J.M. Laarhoven and E.H.L. Aaarts. Simulated Annealing: Theory and Applications. *Kluwer Academic Publischers D. Reidel, Boston*, 1987.

[92] S. Lauesen. *Software Requirements: Styles and Techniques*. Addison-Wesley, 2002.

[93] L. Lavagno and al. Formal Models for Embedded System Design. *IEEE Design and Test for Computer*, 2000.

[94] Luciano Lavagno, Alberto Sangiovani-Vincetelli, and Ellen Sentovitch. *Models of Computation for Embedded Systems Design*, chapter 2 in System-Level Synthesis, pages 45–102. Kluwer Academics Publishers, 1999.

[95] E. A. Lee and D. G. Messerschmitt. Static Scheduling of Synchronous Data Flow Program for Digital Signal Processing. In *IEEE Transactions on Computers, 75(9):1235-1245*, Jan. 1987.

[96] E. A. Lee and A. Sangiovanni-Vincentelli. Comparing Models of Computation. In *International Conference on Computer-Aided Design pp. 234-241*, 1996.

[97] Thomas Lenghuaer. *Combinatorial algorithms for Integrated Circuit Layout*. John Wiley and Sons, England, 1990.

[98] M. Lopez-Vallejo and J. C. Lopez. On the Hardware-Software Partitioning Problem: System Modeling and Partitioning Techniques. In *ACM Transactions on Design Automation of Electronic Systems, Vol. 8, No. 3, pp. 269-297*, July 2003.

[99] D. C. Luckham and J. Vera. An Event-Based Architecture Definition Language. In *IEEE Transactions on Software Engineering, Vol 2, No 9, pp 717-734*, Sept 1995.

[100] N. Medvidovic and Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. Technical report, UCI and USC.

[101] M. Moriconi and R. A. Riemenschneider. Introduction to SADL 1.0: A Language for Specifying Software Architecture Hierarchies. Technical report, SRI International, March 1997.

[102] M. N. Murthy and G. Krishna. A Computationally Efficient Technique for Data Clustering. In *Pattern Recogn. 12, pp 153158*, 1980.

[103] M. Mutz, M. Huhn, U. Goltz, and C. Kroemke. Model Based System Development in Automotive. *SAE*, 2002.

[104] G. Nicolescu, S. Yoo, A. Bouchhima, and Jerraya A. A. Validation in a Component-Based Design Flow for Multicore SoCs. In *ISSS'02, Kyoto, Japan*, Oct 2002.

[105] R. Niemann and P. Marwedel. Hardware/Software Partitioning Using Integer Programming. *IEEE/ACM Proc. of EDAC 96, pp 473-479*, 1996.

[106] C. Papadimitriou. *Computer Complexity*. Addison Wesley,, 1994.

[107] S. Poledna, W. Ettlmayr, and M. Novak. Communnication Bus for Automotive Applications. *ess-circ2001 Proceedings*, 2001.

[108] C. Rupp. *Requirements Engineering und Management: Professionelle, iterative Anforderungsanalyse fur die praxis*. Hanser, 2002.

[109] L.A. Sanchis. Multiple-Way Network Partitioning. *IEEE Trans. On Computers*, vol.38. No 1:pp. 62–81, Jan. 1989.

[110] M. Shaw, R. DeLine, D. V. Klein, T.L. Ross, and ... Abstraction for Software Architectures and Tools to Support Them. In *IEEE Transactions on Software Engineering, vol 21 No 4, pp 314-335*, April 1995.

[111] A.J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-Partitioning. *Journal of Global Optimization 29 2004 Kluwer Academic Publishers. Printed in the Netherlands.*, 225:225241, 2004.

[112] Friedhelm Stappert and Carsten Rust. Worst Case Execution Time Analysis for Petri Net Models of Embedded Systems. In *Embedded Systems and Applications, pp.176-182*, 2003.

[113] W. Tracz. LILEANNA: A Parameterized Programming Language. *Proceedings of the Second International Workshop on Software Reuse, Lucca, Italy*, pages pp 66–78, March 1993.

[114] D.J. Ullman. The performance of a memory allocation algorithm. Technical report, Princeton Uninversity, Princeton, NJ, 1971.

[115] F. Vahid and D. D. Gajski. SLIF: A Specification-Level Intermediate Format for System Design. In *1995 European Design and Test Conference (ED&TC '95)*, 1995.

[116] F. Vahid and D.D. Gajski. Closeness Metrics for System-Level Functional Partitioning. *IEEE Proceding 1995, 328-333*, 1995.

[117] F. Vahid and J. Gajski. Specification and Design of Embedded Hardware/Software Systems. *IEEE Design and Test of Computers; pp 53-67*, 1995.

[118] F. Vahid, S. Narajan, and Daniel. D. Gajski. SpecCharts: A VHDL Front-End for Embedded Systems. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol 14, No 6*, June 1995.

[119] Mauricio Varea. *Modelling and Verification of Embedded Systems based on Petri Net oriented Representations*. PhD thesis, University of Southampton, United Kingdom, Sep 2003.

[120] S. Vestal. A Cursory Overview and Comparison of Four Architecture Description Languages. Technical report, Honeywell Technology Center, Feb 1993.

[121] S. Vestal. MetaH Programmer's Manual, Version 1.09. Technical report, Honewell Technologie Center, Feb 1993.

[122] A. Vikram and R. Sakellariou. Application Representations for Multiparadigm Performance Modeling of Large-Scale Parallel Scientific Codes. *International Journal of High Performance Computing Applications*, Vol. 14 , Issue 4:pp 304 – 316, Nov. 2000.

[123] Michael von der Beeck. A Comparison of Statecharts Variants. In *Proceedings of the Third International Symposium Organized Jointly with the Working Group Provably Correct Systems on Formal Techniques in Real-Time and Fault-Tolerant Systems ; pp 128 - 148*, 1994.

[124] A.C. Yao. New Algorithms for Bin Packning. *J. Assoc. of Computing Machinery*, 27:207–227, 1980.

[125] M. Yue. A Simple Proof of the Inequality FFD(L) $\leq$ (11/9)OPT(L) + 1, for all L, for the FFD Bin-Packing Algorithm. *Acta Mathematicae Applicatae Sinica*, Vol. 7, Nr. 4:pp. 321331, 1991.

[126] Minye Yue and Zhang Lei. A Simple Proof of the Inequality MFFD(L) $\leq$ 71/60 OPT(L) + 1, for all L, for the MFFD Bin-Packing Algorithm. *Acta Mathematicae Applicatae Sinica*, Vol. 11:pp. 318330, 1995.

[127] C.T. Zahn. Graph-theoretical Methods for Detecting and Describing Gestalt Clusters. In *IEEE Transactions on Computers, Vol. 20, No 1, pp. 6886.*, 1971.