The Components Data Flow Machine: An Intermediate Modeling Format to Support the Design of Automobiles E/E Systems Architectures

Augustin Kebemou and Ina Schieferdecker

Abstract The design of the architectures of automobiles E/E (Electric/Electronic) systems consists in the allocation of the hardware platform and the distribution of the computing and the communication loads of the application software within the allocated hardware. This operation is called the partitioning. Following the actual model-driven design schemes, the input of the partitioning is generally a functional specification of the system under development in the form of communicating software components that must be mapped on the allocated hardware platform. However, even though these models are sufficient to describe the structure of a system, they are not good enough to support a CAD-supplied partitioning. They lack the facilities needed to support the analysis of the data flow and to investigate the closeness between the elements of the specification, thus to support the mapping. In this paper, we define the Components Data Flow Machine (*CDFM*), a modeling format that is defined to support the design of automobiles E/E systems architectures. The *CDFM* defines the semantics of a synthesis model that results from a transformation of standard models like SysML, EAST ADL or AUTOSAR models.

Key words: automotive systems, architecture, design, partitioning, mapping

1 Introduction

With the increasing demand for electronic-actuated features in automobiles, two solutions are broadly proposed to optimize the cost of new vehicles. The first solution

Augustin Kebemou
Fraunhofer Institute for Software and Systems Engineering (ISST)
Mollstrasse 1, 10178 Berlin, Germany

Ina Schieferdecker
Fraunhofer Institute for Open Communication Systems (FOKUS)
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany

Please use the following format when citing this chapter:

Kebemou, A. and Schieferdecker, I., 2008, in IFIP International Federation for Information Processing, Volume 271; *Distributed Embedded Systems: Design, Middleware and Resources*; Bernd Kleinjohann, Lisa Kleinjohann, Wayne Wolf; (Boston: Springer), pp. 89–100.

proposes to develop flexible and particularly portable automotive software components. The second solution is to reduce the amount of hardware in automobiles' E/E systems. In reality, these two options are complementary since the efficient usage of the hardware resources is achieved by both a goal-oriented definition of the architecture of the E/E system and an advantageous resource allocation policy. This is the duty of the partitioning. The optimal partitioning must minimize the quantity of processing units, memories and cables that are needed to execute the functionality of the system, to store its data and its software code and to realize the inter-device communication. The partitioning involves three activities: The allocation, the mapping and the deployment. The allocation is concerned with the design of the physical configuration of the system. This consists in the definition of the number of devices (ECUs, sensors, actuators, gateways), the definition of their individual equipments (processing units, memories, internal buses, etc.), their positioning within the physical system (i.e. the topology of the system) and the choice of the communication media and protocols for the inter-device communications. The mapping deals with the distribution of the working load of the system's application among the available devices while the deployment is the task of distributing the individual computation power of the devices among the tasks and assigning the available memory space and the intra-devices communication bandwidths to the system's data. During the mapping, each functional component of the system is assigned to one or several devices depending on the required redundancy grade of its implementation. Two functional components that are assigned to different devices must communicate through the inter-devices communication channels. As these are mostly bus systems running frame-oriented communication protocols like CAN, MOST, FlexRay, LIN, etc., the mapping must also pack the inter-devices communication data in the communication frames in the most economical way.

Thus, a good partitioning must assign closely related components, e.g. highly communicating components, to the same device in order to minimize the inter-devices communication and maximize the hardware sharing [1]. Currently, this is done manually by highly experienced system architects. They usually add the new software components on the existing system without changing the precedent contents of the devices. When the existing devices are overloaded, they generally decide to add new devices. 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's architecture is practically equivalent to a design from scratch, economically unsupportable in this fast evolving industry where the time to market is vital for each OEM. But, the direct consequence of this practice is the excessive number of buses and processors installed in the vehicles. Moreover, without efficient CAD techniques to support the partitioning, unexperienced designers cannot expect to design good systems. A CAD-supported partitioning will allow automotive systems architects to investigate and compare different architectural options. This necessitates a global view of the system's specification and a reasonable degree of portability of the software components at the system-level. With the implementation of the concepts developed within the AUTOSAR[2], standard and platform-independent software components

will enlarge the solution space of the partitioning of automobiles' E/E systems and thus, will allow the consideration of much more architectural options and enable the design of more cost-sensitive E/E systems than today. The E/E design can take advantage of this only if it is provided CAD-support for the partitioning.

2 Problem Presentation

However, a CAD-supplied partitioning tool needs a model that can enable the analysis of the data flow and highlight the closeness between the elements of the specification. This is not provided by the mostly used modeling formats, e.g. SysML[3], EAST ADL[4] and AUTOSAR[5] models. Although these prominent meta-models optimally describe the logical structure of E/E systems, they cannot be used to synthesize the inter-components communication or to determine the closeness between the elements of a model as it is needed to make the mapping decisions. For example, there are generally multiple connectors and interfaces joining two communicating components, making the tracing of the communication paths extremely difficult for a computer system. To solve this problem, we defined a modeling solution called the *FN* -for "Functional Network"- that copes with the deficits of the usual automotive modeling solutions concerning the requirements for a CAD-supplied partitioning such as the screening of the communication paths and the traceability of the communication data. The *FN* is an intuitive modeling solution that inherits the concepts of interconnected software components with ports and interfaces from UML, SysML and EAST ADL, plus the atomicity and portability principles defined within the AUTOSAR. But, in contrast to an AUTOSAR VFB (Virtual Functional Bus), the *FN* interfaces allow clear screening of the communication paths and an easy tracing of the data flowing on each connector by transforming for example the branched connectors found in a VFB into single P2P connectors.

Each *FN* model can be formally defined with a quintuple $\langle F,R,P,I,C\rangle$ as follows: 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 the data components in the specification, 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 specification, 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 the connectors in the specification, 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 and the RTEEs (Runtime Environment Events); 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; $C_i.Int = C_i.src.Int \cap C_i.dst.Int$.

Due to its P2P conception of ports inter-connections, the FN enables the production of specifications that are more compliant with the requirements of an automatic partitioning than the standard modeling solutions. Nevertheless, the FN does not provide any advanced feature to synthesize the communication and extract the closeness values between the elements of a specification. This can be achieved with a formal representation format on which efficient mathematical tools can be used to analyze and quantify the relationships between the elements of the functional specification of a system. We call our solution the "Components Data Flow Machine ($CDFM$)". The $CDFM$ is a synthesis model that enables the automatic analysis of the inter-components communication, the determination of the closeness values between them and the assignment of the exchanged data to the communication frames. A more detailed specification of the requirements for such a synthesis model concerning the partitioning of automobiles E/E systems is given in section 3. Then, the $CDFM$ and the rules that govern the translation of FN models into $CDFM$ models are defined in section 4 and illustrated in section 7 while the annotations of $CDFM$ models and their formal definition are presented respectively in section 5 and section 6.

3 Requirements for 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, in particular the closeness between the components. Reflecting the system architecture requires that the synthesis model must be at least at the same level of granularity with the input model. Enabling rapid metrics estimations requires that as much information as possible is known before the partitioning begins. 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 or Petri nets and those based on graphs. In contrast to graph-based representations that consider a unique system state, FSMs [6] and Petri nets-based representations [7] are powerful in modeling and verifying the dynamics of a system. But, they are obviously not the best representation forms when the architecture of the system 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) [8] and the FSM with Coprocessors (FSMC) [9]. Even these forms cannot reproduce the system's architecture in a useful way. Moreover, they considerably suffer from the state explosion problem.

The most usual graph-based systems representations include data flow graphs (DFG)[10], control flow graphs (CFG), data control flow graphs (DCFG)[11] and

task graphs. DFGs are well-featured to describe the data dependencies. 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 whose 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 [10, 12]. Like in [13], various special task graph-based modeling formats have been used for problems that are similar to the one presented in this work. In [14], 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 [15] to model the communication between a set of tasks. These solutions yield static models that however effectively reproduce the structure and the communication of a system, providing a good basis for our synthesis model.

4 The *CDFM*

The synthesis model is intended to specify the components of a system, their communication and every relevant relationships between them. We defined it as a task graph (V, E, Ω, S) in with each node $v_i \in V$ represents a behavioral or a data component of the corresponding *FN* model. In contrast to *FN* models, it exists only one edge between two nodes of a *CDFM* model. 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. The semantic of such a node is reduced to: "These connected nodes exchange data in some way" , i.e. the direction is ignored by the edge itself. However, transforming multiple and oppositely directed connectors into a single undirected link introduces two problems: Firstly, we need a convenient interpretation of the original connections that will allow to properly capture the data shared between the connected nodes. Secondly, as the edges are undirected, the direction of the communication must be specified somewhere else.

We solved this problem by introducing the concept of tokens in the *CDFM*. A token models a data object that is exchanged between the nodes of a *CDFM* model. The set of the tokens flowing around the graph is Ω. 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 the beginning of the token or the end of the token. Independently of the connector through which a data object k is exchanged within a *FN* model, 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. As the edges are undirected, we model the direction of the communication in the tokens, i.e. the direction of a token defines the sense of its transfer. This definition of the *CDFM* leads to the following straightforward transformation of *FN* models into the corresponding synthesis models:

- Each component of a *FN* model is transformed into a node,
- Each connection of a *FN* model is transformed into an edge and
- Each data object exchanged between two components of a *FN* model is transformed into a token.

Note that several mechanisms can be used on this basic modeling format to describe the data exchange 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 since 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 *CDFM* models. But, as the *CDFM* is yet 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. Thus, a token is available at the date of its creation. Note that a token is available solely means that the token can be transfered. However, the date at which a token is sent is not absolutely the date at which it is available. Depending on its freshness requirements, a very hasty token must be transfered as soon as it is available while the transfer date of a less hasty token can be delayed. These concepts are introduced in the *CDFM* to support the scheduling of the communication and to control the occupation of the communication buses.

In addition to the technical factors of cost optimization such as the communication and the resource usage, the design of an E/E system typically underlies a full range of constraints and strategic concerns that arise from the commercial, the technological, the organizational circumstances of the design as well as the procurement, the production issues, etc. Consequently, some components of the functional model might be required to run on the same device while others are required to run on different devices, e.g. for safety reasons. These relationships between the components typically have heavy consequences on the partitioning and must be specified in the synthesis model. We model them by means of *needs* and *excludes* relations:

- Two nodes v_i and v_j are in a *needs* relationship, i.e. $needs(v_i,v_j)$ is TRUE, if v_i and v_j must be implemented on the same device;
- Two nodes v_i and v_j are in an *excludes* relationship, i.e. $excludes(v_i,v_j)$ is TRUE, if it is forbidden to implement v_i and v_j on the same device.

The *needs* and *excludes* relationships are also defined between the tokens. Note that several similar relationships can be defined on *CDFM* models. They are managed within the set *S*. So defined, the *CDFM* enables the synthesis of the communication, but it does not yet contain the information required to guide the partitioning, i.e. the

information on the basis of which the clustering decisions must be made during the mapping and those through which the cost and the quality of the resulting partition can be investigated. This is provided by means of attributes.

5 Annotations for the *CDFM*

Annotations for the nodes: The performance and the cost of a node are determined by its execution time, 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. 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 the heaps of its runnables are known (or can be estimated), the attributes of the nodes of a *CDFM* model include:

– The software size (swSize): The total amount of memory required to store the code and the data of the corresponding *FN* component when implemented in software.

– The hardware size (hwSize): The total amount of hardware components that would be used to implement the function of the corresponding component.

– The execution rate (eR): The maximum of the execution rates of the runnables of the corresponding component. The execution rate of a runnable is the mean number of times that it is executed during an activation time of the system.

– The priority (prio): The priority order of the most prioritized runnable of the corresponding component.

– The execution time (eT): The ”sum” of the execution times of the runnables of the corresponding component.

Annotations for the edges: The attributes of the edges of a *CDFG* model include:

– The weight (T): The set of tokens that flow over it during an activation period of the system.

– The access frequency (accFreq): The access frequency of the most accessed connector within the corresponding *FN* connection.

– The constraints (cons): Are given by all the consistent sets of all the constraints on the connectors of the corresponding *FN* connection. This include e.g. the latency, the reliability, the security, the safety constraints, etc.

Annotations for the tokens: The most relevant attributes of the tokens include:

– The direction (dir): It defines the sense in which the token is transferred. It is given by the source and the destination nodes of the token.

– The resolution or dimension (res): The number of bits that is needed to encode the corresponding data object.

– The frequency (freq): The mean frequency of emission of the corresponding data object.

– The priority (prio): The priority level that the corresponding data object enjoys in the occupation of a given communication channel.

– The date of occurrence (occur): The date at which the token is available.

– The freshness requirements (fresh): Determine the latest date at which the token must be sent.

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

6 Formal definition of the *CDFM*, model transformation

Given a *FN* model $A = \langle F, R, P, I, C \rangle$ of the functionalities of a 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 the nodes, E is the set of the 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 k 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 $v_i = \langle swSize, hwSize, eR, prio, eT \rangle$ 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 $e_{ij} = e_{ji} = \langle T, accFreq, cons \rangle$ 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} and $e_{ij}.cons$ is the set of constraints on the edge e_{ij}, i.e. on e_{ji}.
- Each token $T_{ij}^k = \langle dir, res, freq, prio, occur, fresh, cons \rangle$ where $T_{ij}^k.dir$ is the direction in which T_{ij}^k flows, (the direction is also 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.

The metric that is used to determine the weight of the edges of *CDFM* models is defined as follows: 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 (i.e. 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.:

$srcConnectors(A_i) = \{c \in C | c.src \in A_i.Int\}$ and

$dstConnectors(A_i) = \{c \in C | c.dst \in A_i.Int\}$, then

$width_{ij} = srcConnectors(A_i) \cap dstConnectors(A_j)$ and thus, 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:

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

7 Applications

The figures 1 and 2 illustrate the results of the transformation of FN models into $CDFM$ models. Figure 1 shows a part of the FN model of the ACC (active cruise control) functionality and figure 2 shows the corresponding $CDFM$ model. In the $CDFM$ version, the relationships between the components are clearly identifiable. Each connection is materialized by a single edge and the exchanged data objects are specified in terms of tokens associated each with the corresponding edge so that the magnitude of the communication between the components can be easily estimated and compared with each other.

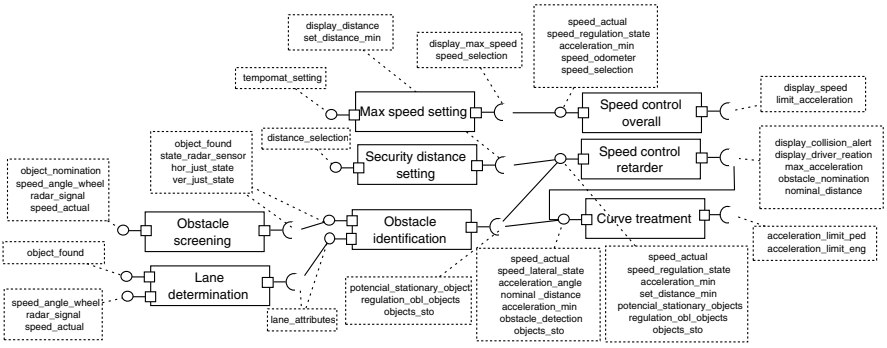

Fig. 1 The ACC FN model

The following simplified example illustrates the transformation of a FN connection in a $CDFM$ edge. 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 starting of the system (as read from the odometer) to the second component A_2. This is done through the sender-receiver

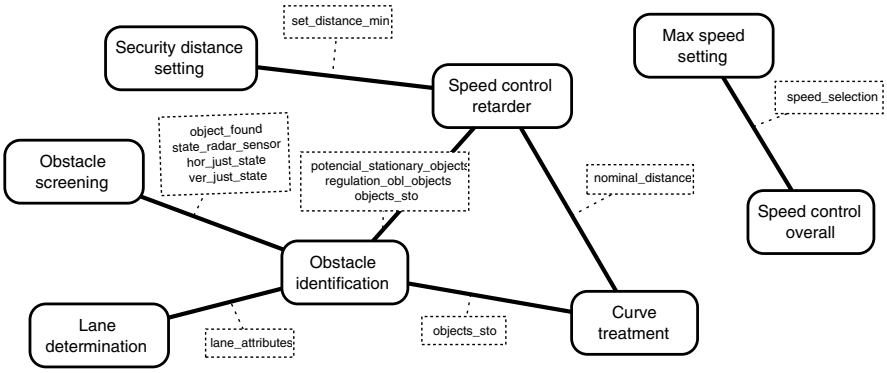

Fig. 2 The corresponding *CDFM* model

interface. Then following a time interval of 1 minute, A_1 triggers A_2 periodically to calculate the total mileage, i.e. the total distance that has been covered by the vehicle since the beginning of the actual trip. The mileage is communicated to A_1 that displays it to inform the user. This is done through the client-server interface. In an AUTOSAR model, these two communication interfaces would be specified by means of one sender-receiver and one client-server interface. But, in a *FN* model, they are specified with three connectors as shown in figure 3. 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. In the end, the weight of the edge e_{12} in the corresponding *CDFM* model shown in figure 4 is the set of data objects, i.e. tokens, exchanged between A_1 and A_2, resp. between v_1 and v_2.

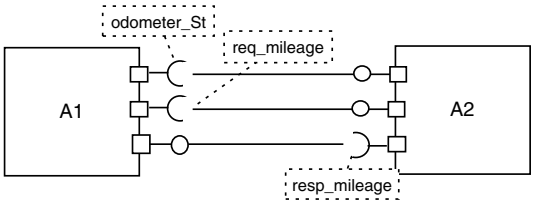

Fig. 3 The FN graphical representation of the mileage inquiry

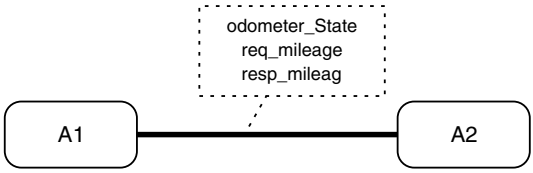

Fig. 4 Graphical representation of the corresponding CDFM

8 Conclusion

The *CDFM* provides a powerful modeling format for the design of E/E systems architectures. It is featured to support the analysis, the synthesis and the measurement of the data flow for the mapping in very complex E/E system specifications at the hight level. The representation of the components interconnections through single unified edges with the corresponding data flow simplifies the measurement and the comparison of the communication between the system components. The graph formulation of *CDFM* models enables the application of usual graph partitioning algorithms to realize the clustering. Due to the concept of directed tokens, the *CDFM* is well-adapted to support the frames packing problem that is inherent to the resource allocation in automotive communication networks such as CAN, MOST, LIN, etc. Furthermore, *CDFM* models are obtained from a simple and straightforward transformation of *FN* models that allow the implementation of CAD system to support the transformation of *FN* into *CDFM* models. Thereto, the *CDFM* format allows the design of flexible models, since *CDFM* models scalable and do not underly any restriction on the granularity of their elements (nodes and tokens). They can easily be enhanced to support the simulation.

References

1. 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.
2. AUTOSAR, "www.autosar.org."
3. *Systems Modelling Language (SysML) Specification, OMG document: ad/2006-03-01; version 1.0 Draft*.
4. EAST-EEA, " Embedded Electronic Architecture. Definition of Language for Automotive Embedded Electronic Architecture v. 1.02," ITEA, Tech. Rep., 30.06.2006.
5. *UML Profile for AUTOSAR; V1.0.0; AUTOSAR Adminstration web content, 28.04.2006*.
6. M. 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.
7. M. Varea, "Modelling and Verification of Embedded Systems based on Petri Net oriented Representations," Ph.D. dissertation, University of Southampton, United Kingdom, Sep 2003.
8. 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.
9. A. Jerraya, H. Ding, P. Kission, and M. Rahmouni, "Behavioral Synthesis and Component Reuse with VHDL," *Kluwer Academic Publishers, Boston/ London / Dortrecht*, 1996.
10. 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.
11. 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.
12. S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*, M. Norwell, Ed. Kluwer Academic Publishers, 1996.

13. 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.
14. 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.
15. 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.