MICROSCOPIC MODELING OF SUPPORT SYSTEM FOR IN-VEHICLE EMBEDDED SYSTEMS

M. Courrier, F. Simonot-Lion, Y.-Q. Song

This paper deals with the modeling of systems supporting in-vehicle embedded applications. The modeling principle is based on the development of reusable component models with clearly defined interfaces. The design of a whole application model is obtained by using these components and the resulting global model can be simulated for the verification of temporal properties and for the optimization of an implementation.

1. Introduction

The improvement of the safety in a car, the minimization of pollution and the optimization of traffic management are important challenges for automotive industries. A solution of these problems lies in the design of Electronic Control Units, embedded in cars and assuming particular functions as door control, engine control, ... These different units are realized thanks to software applications distributed on computers connected to networks. Therefore the main objectives are, on the one hand, to find the best hardware architecture and the best distribution of functions (**optimization**), in terms of several criteria such as cost, memory size, ... and on the other hand to prove that the result is compliant with safety and real-time requirements (**validation**). The purpose of this paper is to present a method of modeling such applications in terms of queuing networks and of simulation of these models. Principles of modeling are based on the development of reusable component models.

The *optimization* and *validation* process depends on two complementary activities. The first one consists in designing the **Operational Architecture** ; the given data of the problem are a set of functions, their links and their behavior, and a set of hardware component ; the aim of this activity is to define a hardware architecture (computers, networks, ...) and a distribution of the functions. It is not easy to have a deterministic algorithm which calculate the best partitioning of the set of functions. In this paper we

don't tackle this problem.

So, we consider that the data of the second activity is a given operational architecture and that the purposes of this second activity are firstly to verify that the operational architecture has the specified properties and, secondly, to determine if it is the best architecture by comparison with other ones. For economic reasons, this optimization and verification process is based on the exploitation of application models. Below (Figure 1) we illustrate this process. The models library contains the models of different hardware components ; this allows a modular approach for the modeling process.

Figure 1 : Validation process

The application model to be simulated is structured in several abstract levels from the functional level (described disregarding the distribution and hardware performance problem) to the hardware level. More precisely, we can identify the following elements :

- **Functional architecture** : it represents the functions, their links and their behavior. In the context of in-vehicle applications, the specification of these functions are closed to programming concepts. That is why we use in this paper terms like “task” (function), “software architecture” (functional architecture) or “message” (data link). The functional architecture specifies the behavior of application in response to external stimuli.
- **Hardware architecture** : it is the model of the execution support of the previous architecture (computers with their operating systems connected on networks with their protocols). In the context of in-vehicle applications, computers are micro-controllers (hardware component) composed of a Central Processing Unit (CPU), a memory and some input/output devices.
- **Operating System** : it manages the computer resources for the tasks. It interfaces hardware and tasks. This concept will allow us to change a hardware component

without changing the functional architecture.

- **Central Processing Unit** : it is the heart of a micro-controller. The execution delay of a function depends of it (set of instructions, architecture of the CPU, ...)
- **Communication system** : it supports the messages exchanges. Its Medium Access Protocol and its transmission rate influence the application performances.

Modeling activities of electronic units are usually done thank to the VHDL or VHSIC hardware language description. But this languages are not well-suited to model the functional architecture. Moreover in-vehicle applications are distributed on network such as CAN. Several models of this network are available. Some of them evaluate the performance by analysis of the model. For example [9], [10] and [13] obtain results in the worst case and under strong hypothesis ; [11] proposes an analysis algorithm of a model based on Timed Petri Nets. Finally, some models are based on queuing systems formalism and offer different levels of precision such as in [7], [8], [12]. Note that the same formalism is currently used for the modeling of operating systems and especially the scheduler which has an important impact on the application performances. So this bears out that the validation is processed thanks to the queuing systems formalism. Furthermore we have built a platform based on the SES Workbench [14] in order to assume both modeling and verification activities.

Note that the proposed methodology is suitable to any distributed applications subject to a precise functional architecture description. We have applied it to the in-vehicle embedded systems under some hypothesis : the Operating System is OSEK/VDX [3], [4] and the communication system is CAN [1].

The rest of the paper is organized as following. Section 2 gives the modeling methodology. Section 3 shows an example of the component, illustrates how to build the global model of the application and how to exploit it. Finally, section 4 brings the conclusions on this work.

2. Modeling principles

We present below how to build the interchangeable component models to be included in the models library appeared in Figure 1. These component models should be integrated into the global architecture model, the interface of each of them must be clearly defined. As it has been told above, one of the objectives of this work is to be able to simulate and compare the performance of the operational architecture when different micro-controllers are used, the models of this kind of components must therefore be detailed, this justifies comes the need of the microscopic models.

2.1. Component models

Considering the application processes using the OSEK/VDX operating system running over different micro-controllers [2], [5], [6] interconnected by a CAN network, the components to be modeled are software and hardware as shown by Figure 2. They are: Micro-controllers, CAN controllers, CAN bus, OSEK/VDX,

Application and Environment.

Figure 2 : Software and hardware

In order to provide modularity in the modeling process and interchangeability of hardware component models, we adopt following principles :

- The models of the different micro-controllers (respectively CAN controllers) must be independently developed and have a common interface
- The model of the CAN bus is unique as well as that of OSEK/VDX
- The functional architecture of the application as well as the environment can be considered as the inputs to the model of OSEK/VDX

For the hardware architecture, this leads to four models, each playing an accurate role. The model of the CAN bus represents the message transmission among stations following CAN protocol. The model of the CAN controller, which could be more or less detailed, represents the communication interface of each node. The model of a micro-controller represents the computing capacity of a node, from which the tasks execution times are determined. The model of OSEK/VDX describes the way the resources are handled : processor time sharing (scheduling), inter-tasks communication primitives (messages, event, ...), etc.

2.2. Interface of the component models

The definition of the interaction between the different models consists in on the one hand fixing with accuracy the nature of the exchanged information and on the other hand constructing the necessary interfaces.

Figure 3 clearly shows the need of two interfaces : <OSEK/VDX - micro-controller> and <OSEK/VDX - CAN controller>.

Figure 3 - Interaction definition among component models

2.1.1. Interface between tasks handled by OSEK/VDX and the micro-controller.

Two following points should be modeled :

- allocation of the processor to tasks, this relates to the scheduling of OSEK/VDX
- model of the task in terms of instruction executions when the processor is assigned to a task

Two ways can be used to represent the computing load generated by a task. The first consists in estimating the task processing time (the upper bound), the model is thus coarse-grained.

To get a better simulation accuracy, the second way we proposed uses the model of the microscopic micro-controllers model with which we convert a load, represented by a sequence of instructions, to the corresponding exact execution time. The model is thus fine-grained. The problem arisen here is how to describe these sequences. In fact, at the functional architecture level, the nature of the hardware support is masked.

Nevertheless, the model of the micro-controller needs for its input an instruction sequence described in its specific language (machine code) in order to get a good accuracy on the simulation results. We should thus utilize a converter to translate an instruction sequence described in the generic description language at the functional architecture level to its equivalent description in the specific language of the target micro-controller. As we can see in Figure 3, the generic language is the C language while the converter is a cross compiler.

2.2.2. Interface between tasks managed by OSEK/VDX and the CAN controller.

The role of the CAN controller is to handle message transmission and message reception according to CAN protocol. OSEK/VDX operating system makes transparency the communication support to users. It allows a task to send messages towards the mailboxes which are addressed using OSEK/VDX identifiers hiding thus the underlying communication controller. Nevertheless for this message exchange mechanism, there is neither priority concept nor reference for mapping a mailbox to one or several buffers of the underlying communication controller (this is left for

implementation free choice). For instance, using a CAN controller with 15 buffers (Intel 82527) will be the same for a user than using a CAN controller with only two buffers in tandem (Philips 80C200). But in the simulation model this difference must be taken into account, i.e., the implementation choice must be modeled, otherwise the performance evaluation will not be able to provide useful results. This is why we propose an interface to handle on the one hand the assignment of CAN identifiers to messages and on the other hand the way to use the resources (buffers) of the chosen CAN controller.

2.3. Levels of model detail and model hierarchy

Until now we described the interaction among the component models by considering each of them as a black box. Here we will be interested in the way for modeling the contains of these black boxes. For this we have to choose a good compromise between the level of the model detail and its usability. It is true that a detailed model can gives accurate results but considering the size and the complexity of our whole system, we can not take into account all parameters. Fortunately, in practice not all parameters have the same influence on the system performance. One of the important step for the success of a simulation is then to identify the important parts which will be the object of a detailed modeling.

The use of hierarchical tools like SIM language [14] allows us to model a component with different detail levels, i.e., from a general view to more and more detailed levels. In fact, we can define the different desired detail levels for different units within a component model as well as for different component models according to their importance. Figure 4 shows the four function blocks of the CPU that should be detailed.

Figure 4 Function blocks of the CPU model

The input to this model is provided by the application processes in terms of a machine code sequence representing the required processing. The code execution is emulated in the first block. This code enters then into the model of the pipeline which allows to evaluate the execution time of the code. This time also depends on the access time to the peripherals, the ROM and the RAM. We model thus these peripherals as well as the busses linking these peripherals to the pipeline. We note that each of the above defined function blocks could be the object of a more or less detailed modeling according to its importance and complexity.

3. Example of component models and model exploitation

We developed different component models representing some hardware components of our in-vehicle applications : OSEK/VDX, CAN bus, CAN controllers and CPUs (Siemens C167 with Intel82527 [5, 6] and Motorola MPC555 with TOUCAN [2]). These models are developed under an industrial contract with PSA Peugeot-Citroën automobile company. Because of the confidentiality, in this section we only describe the component model of OSEK/VDX. An example is also given to illustrate how to exploit the global model by using the proposed framework.

3.1. OSEK/VDX component model

An operating system provides for an application program a framework within which this program runs. The main features of OSEK/VDX are to handle the processor sharing, to provide the communication mechanisms, to synchronize the tasks and to share other resources. The two principal elements are : task scheduler to correctly share the processor and primitives to handle tasks, alarms, messages and events. An application program is described at two levels :

- at the scheduling level, we describe a task by giving all information necessary for its scheduling : task type (basic task or extended one), with or without preemption, priority. Figure 5 shows the model of OSEK/VDX scheduler. Four task states are identified and the transitions between states are specified.
- at the task description level, we describe in the task code the primitive calls such as event waiting, task activation, message sending, ... The model takes into account the execution time of these primitives.

Figure 5 Model of OSEK/VDX scheduler

The execution duration of a primitive depends on the implementation of OSEK/VDX. For each version of OSEK/VDX and for each supporting micro-controller, it is necessary to evaluate the primitive execution time. In some case this time can also be provided by the implementator.

The performance parameters are mainly the response time for a task running request, utilization of the processor, overheads because of the preemption, number of ready tasks, ... We point out that an interrupt is considered as an external event and its

corresponding ISR (Interrupt Sub Routine) as a task.

3.2. Model exploitation

The component models described above are included in a library. Figure 6 shows how to build and utilize a model of an operational architecture.

Figure 6 Procedure to build and to use the global model

The first step consists in withdrawing the component models from the “model library” to construct a global hardware architecture model according to the “hardware description file” and “parameters file”. The second consists in compiling the model to generate an “executable simulator”. And the third consists in the simulation using as inputs the “application model” in terms of tasks with their distribution and the “interrupt load” (external events). We note that observable parameters are specified in the “trace specification file” and the simulation results are stored in “trace files”. Actually the definition of the observable parameters is done using probes provided by SES/Workbench. Some component models can be parameterized. In this figure we show two different ways to parameterize the component models :

1. during the building of the global model. In this case the values of these parameters are fixed.
2. before the running of the global model. In fact, SES/Workbench allows a simulator to read a configuration file before each simulation. This second way is more flexible but requires one more configuration file.

Figure 7 shows an example of a global hardware architecture model composed of two nodes using OSEK/VDX as their OS. The executable simulator is generated using the tools furnished by SES/Workbench. In this figure, the role of « initialization » is to read the file describing the functional architecture and the task distribution as well as the interrupts generated by the environment. In this example file, 6 periodic tasks

$T_i(P_i : \text{period}, C_i : \text{execution time})$ with the same priority are distributed on 2 nodes. Node 1 receives one interrupt every 5000 time units and executes an ISR of duration 500. Node 2 receives two interrupts with ISR of duration = 100 and of period = 5000.

Figure 7 Example of a global hardware architecture model

The table below shows the task set distributed on N1 and N2 according to two configurations :

T_i	P_i	C_i	Configuration 1	Configuration2
1	22000	10000	N1	N1
2	22000	8000	N1	N1
3	10000	300	N1	N2
4	10000	300	N2	N2
5	10000	400	N2	N2
6	15000	200	N2	N2

Figure 8 shows one of the simulation results we can obtain. It consists of the inter-activation intervals of the task T3 for the two different configurations. One can see that Configuration 1 can not meet the time constraint on the period of the task T3 while Configuration 2 can meet it.

Conclusion

This paper dealt with the modeling of systems supporting in-vehicle embedded applications. The modeling principle based on the development of reusable

component models with clearly defined interfaces is explained and the models of CAN controllers, micro-controllers, OSEK/VDX are developed. An example is provided showing that the design of a whole application model can be obtained by using these components and the resulting global model can be simulated for the verification of temporal properties and for the optimization of an implementation.

Figure 8 Inter-activation intervals of Task T_3

References

- [1] ISO, "Road vehicles – Interchange of digital information – Controller Area Network for high-speed communication", ISO 11898, 1994
- [2] Motorola, "RCPU RISC central processing unit, Reference manual", 1996
- [3] OSEK/VDX, "Communication", Version 2.1 revision 117, June 1998
- [4] OSEK/VDX, "Operating System", Version 2.0 revision 115, October 1997
- [5] Siemens, "C167xx, 16-Bit CMOS Single-Chip Micro-controllers", User's manual, 1996
- [6] Siemens, "C16x Family instruction set", 1996
- [7] Simonot-Lion F., Y.Q.Song, J.Raymond, "Validating real-time applications distributed over CAN: an interoperability verification", 4th international CAN conference, ICC'97, Berlin (Germany), October 1997
- [8] Song Y.Q., F. Simonot-Lion, P. Belissent, "VACANS - A tool for the validation of CAN-based applications", Proceedings of IEEE WFCS'97, Barcelona (Spain) October 1998.
- [9] Tindell K., A.Burns, "Guaranteeing messages latencies on control area network (CAN)", 1st international CAN conference, Maintz (Germany), September 1994
- [10] Tindell K., H.Hanson, "Babbling idiots, the dual-priority protocol and smart CAN controller", 2nd international CAN conference, September 1995
- [11] Toussaint J., C. Philippe, F. Simonot-Lion, "A model of Can-Based Applications for the Verification of Temporal Properties", SICICA'97, 3rd IFAC Symposium on Intelligent Components and Instruments for Control Applications, Annecy (France), June 1997.
- [12] Upender B.P., A.G.Dean, "Variability of CAN network performance" 3rd international CAN conference, pp9.2-9.9, Paris (France), October 1996
- [13] Wang Z., H.Lu, G.Hedrick, M.Stone, "Message delay analysis for CAN based network", Proceedings of 1992 ACM SIGAPP symposium on applied computing, Kansas city, MO (USA), March 1992
- [14] SES, "Quick start and upgrade guide; creating models; simulating models; technical reference; SIM language reference; using the query language; modeling reference; Cosim reference", SES/Workbench, 1997