A Universal Communication Model for an Automotive System Integration Platform

Thilo Demmeler [1], Paolo Giusto [2]
[1] Bayerische Motoren Werke AG, Technology Office Paolo Alto 94301 CA USA
[2] Cadence Design Systems Inc., San Jose, CA 95134, USA
Thilo.Demmeler@bmw.de, giusto@cadence.com

Abstract

In this paper, we present a virtual integration platform based design methodology for distributed automotive systems. The platform, built within the 'Virtual Component Co-Design' tool (VCC), provides the ability of distributing a given system functionality over an architecture so as to validate different solutions in terms of cost, safety requirements, and real-time constraints. The virtual platform constitutes the foundation for design decisions early in the development phase, therefore enabling decisive and competitive advantages in the development process. This paper focuses on one of the key-enablers of the methodology, the Universal Communication Model (UCM). The UCM is defined at a level of abstraction that allows accurate estimates of the performance including the latencies over the bus network, and good simulation performance. In addition, due to the high level of reusability and parameterization of its components, it can be used as a framework for modeling the different communication protocols common in the automotive domain.

1. Introduction

The increasing demand for comfort, information, and safety in a car is satisfied through the introduction of distributed architectures with electronic control units (ECUs). The sharing of data between ECUs that communicate over automotive buses allows integration of additional functionality at lower costs. Moreover, a modular function can be distributed over a network of ECUs.
Different kinds of communication protocols with their own specific strengths have been introduced in the past, for example the Controller Area Network (CAN) [1] or ByteFlight [2]. Other protocols such as TTP [3] [4] are going to be introduced with the goal to provide more dependable and fault tolerant networks that enable the step towards *by-wire* technology for braking and steering [5] [6].
In this scenario of increasing complexity, the main challenge for the automotive industry is to reduce production costs, shorten development cycles, and guarantee for a premium safety concept. To achieve these objectives a shift in the system design process has to take place [7] [8] [9].
This paper will proceed as following: first, the design methodology for the virtual integration platform and the virtual design workflow are shown. Then, the transition from an *ideal* communication model (zero time), to a *realistic* model with performance based on the underlying architecture model is drawn. Broadcasting and communication cycle layouts of the UCM are described next. Then, the communication matrix and the data frame packaging are introduced. Finally, an outlook and a summary conclude the paper.

2. The Virtual Integration Platform

The development process in the automotive domain starts with the analysis phase, where a functional network is developed, and proceeds to the specification phase, where algorithms for the functional components are *defined.* The system design phase determines the distribution of the functionality onto an architectural network. In the next phase, a composition of functional components is implemented onto the target hardware and finally the system is calibrated in the car [10]. A seamless flow through the development stages is as important as the ability of building a virtual prototype very early in the development stage. Substantial for valuable prototyping results is a seamless transition from an ideal world assumption to the real world where the application is supposed to run.
The proposed virtual integration platform is built within the Virtual Component Co-Design (VCC) tool set [11] as shown in Figure 1. The basic concept is to have a behavioral model of the system with ideal world assumptions in terms of zero software execution and communication delays, which is separated and independent from an architectural model that represents an implementation variant [15]. By mapping the functionality onto the architecture, a specific system partitioning is chosen. The system models are transformed into performance models that include

close-to-real execution and communication delays.

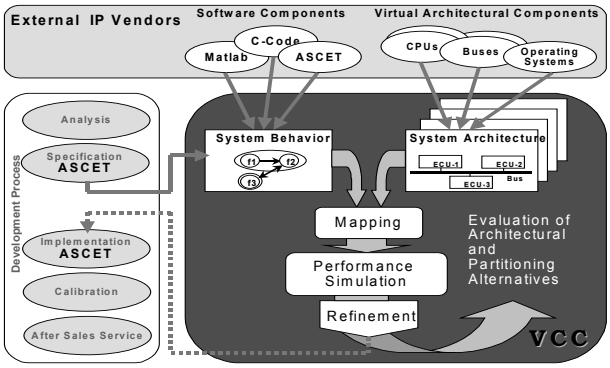

Figure 1. The VCC Virtual Integration Platform

VCC comprises the framework for intellectual property integration and authoring. [16]

2.1. Import of Functionality

In the specification and the implementation phases, the ASCET-SD tool is commonly used in the automotive domain for algorithm development and code generation for single processor units. ASCET-SD [12] is a typical *ideal world* representative tool set that assumes no execution delays during the simulation. An ASCET-SD/VCC automated import flow that preserves functional specification details such as hierarchy, interface and scheduling information is currently underway [8]. An imported ASCET-SD model is represented in VCC as a hierarchy with the project at the top level that comprises the functionality of one entire ECU. The modules at the next lower level state the functional components, which are the smallest mapping unit that can be distributed over the system network. The processes, included in the modules, are the smallest schedulable unit and constitute the leaf blocks in the hierarchy. Finally, tasks are the aggregation of processes, which have the same scheduling policy. To enable the VCC performance estimation, the source code of the processes, which share a considerable amount of data within the module, are imported as white boxes. Furthermore, the scheduling information of the functional ASCET-SD model in terms of ordering, timing, priorities and properties can be preserved, as well as the data exchange and the interfaces of the processes, the modules and the entire project. Alternatively behavioral models can be imported manually as “C” white or black boxes of plain C or from the Matlab tool set [13] in the near future.

2.2. Distribution of Functionality

VCC, as a platform tool, allows the distribution of the functionality - the software modules and projects - over the target resources - and provides a way to explore design alternatives quickly by simulating the distributed system under real time constraints. Trade-off analysis of function distribution, ECUs' loads, and costs of communication enable the design optimization process. A safety analysis can be performed either on the overall system or on specific partitions of the system incrementally. Therefore, the underlying communication model has to model aspects related to real time constraints and has to be flexible enough to allow re-distribution of the software modules within a cluster. Furthermore, in order to achieve design efficiency, the highest grade of automation is necessary that takes into account the dependencies of functional distribution and communication configurations.
In a first evaluation step, the architecture of the imported ASCET-SD projects can be re-modeled. This enables the benchmarking of the performance simulation results, which includes the automatic estimation of task run time and modeling of communication delays.

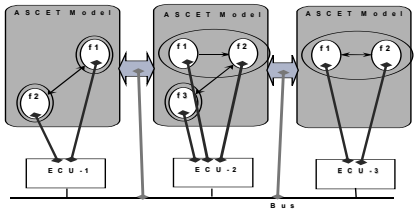

Figure 2. Mapping of ASCET-SD Projects

In the further design steps, the functionality can be redistributed and different kind of architecture alternatives can be explored. The former ASCET-SD project structure (one project per ECU) might dissolve as modules that belong to the same projects can be mapped to different ECUs (Figure 3)

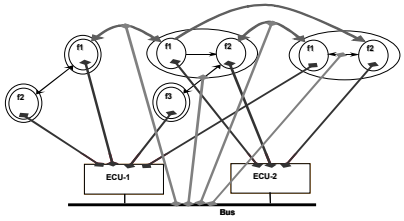

Figure 3. Optimized System Mapping

2.3. Design Work Flow

The design process on the VCC platform can be outlined as following:

1.) Definition of a behavioral diagram in VCC by importing functional components in form of software projects and modules.
2.) Generation of an ideal communication between the functional components in the behavioral model, which does not consider delay or error handling.

3.) Creation of an architectural diagram in VCC
4.) Mapping of the software modules onto the CPU of a cluster by either retaining the mapping of all modules of the project to the according ECU or not.
5.) Generation of the CPU scheduling. This step can be done either manually or automatically by the import step if the original scheduling information is preserved.
6.) Functional simulation. No communication performance is estimated. The software execution time is estimated. [17]
7.) Design iteration by re-distribution of the functionality and tuning of the scheduling of single CPUs.
8.) Initialization of the UCM performance model. Automated generation of an initial communication matrix that carries the dependency of the functional system mapping.
9.) Performance simulation. The bus communication delays are estimated. Bus latencies are still inaccurate due to the missing UCM configuration.
10.) Definition of a specific bus protocol implementation by UCM parameterization. Definition of the communication cycle layout. Data frame definition.
11.) Performance simulation including the bus latencies.

Phase1 and 4 to 7 are described in [8] in detail.
Phases 2, 8, 9, 10 are explained later in the paper.

3. Functional Model

The separation of the functional model from its architectural implementation is the key abstraction of the methodology and the prerequisite for full-scale distribution alternatives for the functional components.

3.1. Functional Networking

The communications between the software components are *naturally* modeled as shared memories, called behavioral memories (BM) in VCC [8]. As shown in figure 4, BMs can be referenced by an unlimited number of read-, write-, or read/write-modules. The non-consuming data access is realized through BM-read or BM-write function calls, which are invoked in the process source code that is generated in the export step. After importing *incrementally* either single modules or complete projects, the I/O interface of the top-level blocks is determined through unbound behavioral memory references [8].
The functional system network is finally created by manually binding the BM references of the top-level blocks to the corresponding BMs. As shown in figure 5 the system gets stimulated within a model of the environment. The possible test spectrum ranges from specific component stimulation to a closed loop simulation of the entire network, for example by applying a vehicle dynamics model.

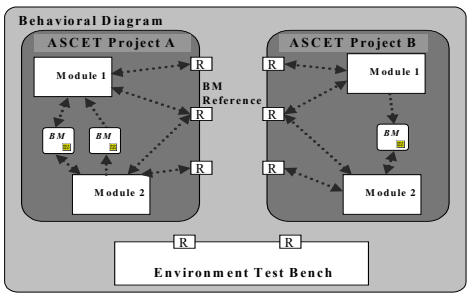

Figure 4. Software projects as imported into VCC

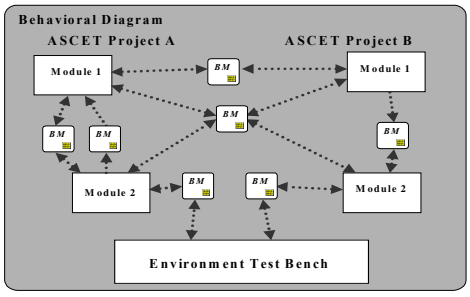

Figure 5. Architecture independent Behavior

Note that the behavioral memories in this use model are not intended to really map them onto architecture. They state an ideal communication between modules without any performance aspects.

3.2. Shared Memory Types

As a result of mapping the functional components onto the architectural components, the mapping of a communication arcs for the BM references to a *communication pattern* is automatically inferred by VCC. A *dynamic* performance model is assigned next.
Only BM's that are connected to modules that are mapped onto different ECU's, participate to the bus traffic. This states an essential information for the VCC user, which is revealed by VCC after (and dependent on) the distribution of the functional components. We differentiate between two different types of behavioral memories:
Register type behavioral memories that represent messages or global variables that are not sent between ECUs.
Bus type behavioral memories (BBM) that are sent over the bus.
The division ratio states an important information in the design process as the quantity of BBMs induces the busload.

3.3. Message Protection

ASCET differentiates between global variables and messages, which are preemption-protected global variables. The protection mechanism of the messages is re-modeled in VCC within separate additional processes, which are automatically generated by VCC at the import step. [8] The aggregation of processes in the modules differs from the aggregation of processes in the tasks, which results in a dependency between the system mapping step and the message protection mechanism. This dependency is dissolved in the proposed design workflow by the VCC tool that re-generates the message protection processes after (and therefore in dependency of) each new system mapping or after tuning the scheduling.

4. The Architectural Model

An automotive system network consists of several ECUs connected to a bus. For fault tolerance reasons, redundancy may be introduced by using multiple bus channels. An ECU classically consists of at least a host controller, a bus controller, and a physical bus driver. As shown in figure 6, the software running on the host is usually separated in hardware independent application software and a communication layer that is hardware (and application) dependent [14]. Also, a real time operating system (RTOS) that provides services to the SW depends on the underlying architecture. In order to achieve behavior/architecture independence and therefore the reusability of functional components, only the application software is appropriate to be imported and modeled as a behavior in VCC. All other layers are modeled as architecture models in VCC.

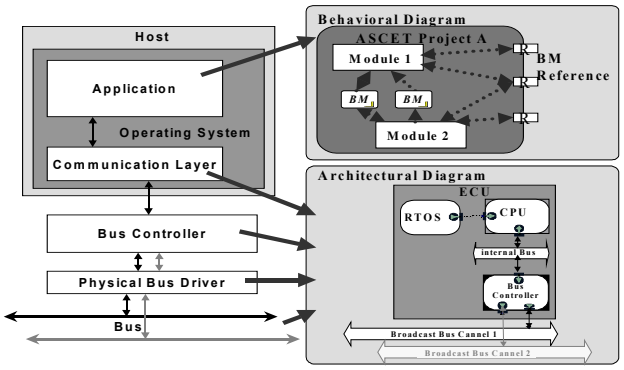

Figure 6. Modeling an ECU in VCC

The interface between host application and architecture is independent from a specific bus implementation in the proposed design methodology. The messages that *produce* the bus traffic are automatically determined from the functional mapping of the system.

4.1. Architectural Services

An architectural component in VCC contains architectural services that are virtual C++ functions, which model both, the performance and also some of the functionality of the component. As shown in figure 7 the UCM is modeled by a stack of architectural services modeling the single bus components of the network cluster. VCC determines the path from the architectural topology netlist and links the necessary protocol components into the UCM. Changing the topology does not require remodeling the UCM services.

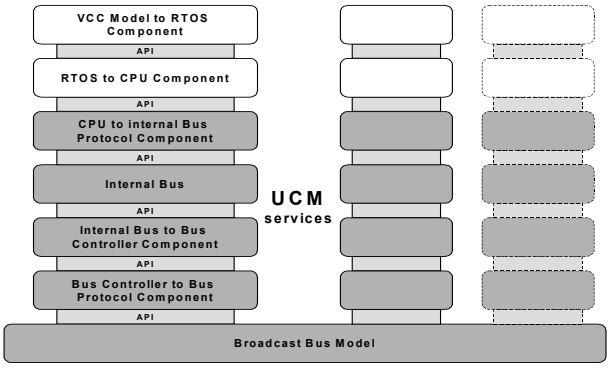

Figure 7. Modeling an ECU with architectural services

In the real application, the bus transactions and the functionality running on the CPU, are parallel activities. VCC supports this paradigm because there is no specific activation semantics bound to the architectural services; they can schedule and handle asynchronous events. This enables the UCM to run asynchronously from the modeled host functionality and the RTOS of the CPU, therefore modeling reality.

The UCM approach covers communication delays that are bus protocol specific, such as packaging the messages into frames, the frame transmission policy and the queuing mechanism of data frames. The latency of the host / bus controller interface is not considered yet in the sense that we have not modeled the performance related to access data from the application SW to the bus controller. The reason being, this delay is negligible (nsec) with respect to the global performance of the entire system (SW scheduling and communication protocol – usually in the order of msec's) The introduction of blocking mechanisms[1] that model this delay is left to a further refinement of the UCM.

[1] Blocking means the sending thread is delayed until the data has been transmitted by the bus controller

4.2. Broadcast Bus Model

Broadcasting is modeled through architectural bus memories local to each node. They are essential for modeling communication latencies in the performance simulation. The bus communication delays can take quite a long time for example in case of an over load or a bus failure. Hence, the functional components that are mapped onto different ECUs can read different values of the same bus message, which is represented by one BBM in the behavioral diagram. The same is possible if several read/write modules of the same BBM exist in general. The communication software layer that handles the data exchange between bus controller memory and host application is not modeled within the UCM. Therefore, it is assumed that a read/write component that reads a bus message always reads the most actual value, independent if the last update was coming from the module itself or through a bus transaction. As shown figure 8, each BBM has a corresponding local bus memory (LBM) in each node of the cluster. Accordingly, a bus transaction is broadcasted to each ECU of the cluster, even if the behaviors mapped to this ECU will not read some of the messages.

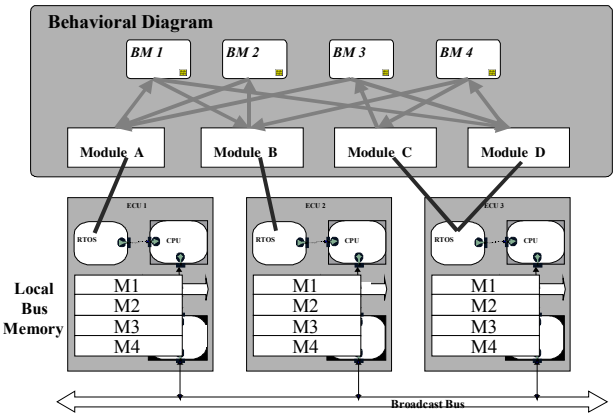

Figure 8. Local Bus Memory Allocation

The functional components mapped on an ECU access the bus message values directly from the corresponding LBM of the ECU. The access is tied to the BBM read and write functions and therefore dependent to the scheduling of the RTOS. On the other side, the architectural services of the UCM are controlling and updating the LBMs of each node of a cluster asynchronously, considering the bus latency in accordance to the traffic over the network.

For diagnostic reasons, an architectural status memory register can be introduced that allows to feedback mode changes from the bus model to the behavior of an ECU. The behavior could check the status of the register to decide what to do next, for example in case of failure in the transmission. Diagnosis mechanisms are left to a further refinement of the UCM. A synchronization mechanism between bus controller and host, as common for a time driven protocol can also be realized via the status register.

The LBMs are service internal vectors that are not directly visible to the user in VCC. The bus traffic can be analyzed in VCC via view ports and probes tied to the different architectural bus services. They *filter* the LBM vectors in a way that the communication can be visualized in a clear form.

4.3. Universal Communication Model

The central idea of the UCM is to provide an open framework that models the performance of the two basic automotive bus concepts, time- and event-driven. We envision using the UCM in a refinement process. At the beginning of the exploration stage, the designer may start with the UCM being set up with some default parameter that, for instance, make it run in a pure event-driven mode – less expensive than the time driven case.

The refinement down to the different kinds of existing bus protocols, is done step by step by the VCC user through adapting the UCM parameter settings. It even can be used as a framework for investigating the performance of new communication protocols not yet available off-the-shelf, because the UCM provides highly reusable building blocks. Once the UCM settings are matching a specific communication bus protocol, we expect accurate performance results to be obtained that allow qualitative assessments, at least for the system running properly under no fault conditions. The implementation of the UCM infrastructure is underway. We are planning to provide simulation results based upon a real automotive application quite soon.

Error cases are not covered by the UCM in a first step as they often rely on hardware features of specific bus components. As future work, specific bus protocol features can be implemented in VCC by either refining the architectural service models, where for example failure states could easily be implemented, or by explicitly importing specific bus protocol models e.g. from silicon suppliers.

4.4. Message Packaging

The communication delays mainly depend on the data frame packaging and the activation policy of the frames [18]. A single bus transaction always causes some overhead, for example because of the frame header or the inter-frame gap. In order to keep the system costs low; the net-bandwidth usually is increased by packing as many messages of a sender ECU into data-frames as possible. The packaging process is supported in the VCC tool set by UCM parameterization. The frame names and the overheads are specified within the frame properties. The frame size is not limited in VCC and can be adapted to any kind of protocol.

Next, the activation policy has to be assigned to the data frame that defines if it is either a time- or event-triggered frame. In case the designer selects a frame to be time

driven, the sending activation is generated from the architectural service that models the bus controller. A time driven frame has a fixed assignment in the communication cycle by defining the sending interval in the frame properties. In case a frame is defined to be an event driven frame, the activation policy is derived from the corresponding BM-write event of the underlying functional model. Limits or timeouts can superpose the sending event if defined in the frame properties. Alternatively also event frames can be sent at a periodic interval time.

4.5. Communication Cycle Design

In the proposed UCM, we allow the composition of a communication cycle at compile time that assigns static parts for time-driven data frames and dynamic parts for event-driven frames, as shown in figure 9. This leads the designer to explore the performance of different communication protocols.

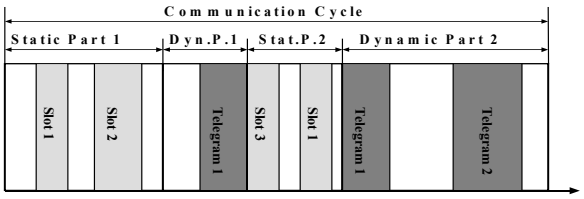

Figure 9. Communication Cycle of the UCM

In the static parts of the communication cycle, the time-driven frames are transmitted in slots according to a statically defined TDMA scheme. In the dynamic parts event-driven frames, called telegrams, are transmitted according to an arbitration algorithm. The communication cycle, represented by state machine in figure 10, is controlled by a global synchronous time.

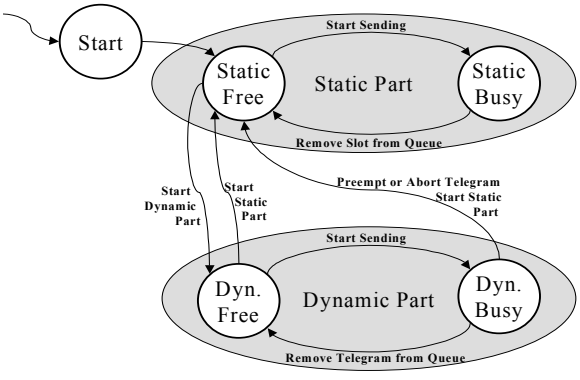

Figure 10. State Machine of the Bus Model

The bus state is Busy while a frame is sent. If the transaction is over, the data frame is removed from the sending queue and the bus mode changes back to Free. Cycle changes are usually performed in the state Free, except telegrams that can be interrupted to ensure a proper start of time driven frames. The interrupted telegram remains in a queue and is sent later in the next dynamic part.

4.6. Arbitration and Queuing

Although a properly defined time driven communication does not require queuing and arbitration, the idea is that the UCM provides a fall back solution in case of collisions. In other words, time slots are only possible when they are correctly assigned over a complete cluster, otherwise they are sent on a first-come first-serve basis. Therefore, the UCM provides a queuing mechanism for *both* event driven- and time driven frames. The VCC user gets a warning if there is a slot activation conflict in the static part of the communication cycle. This method enables a continuous and seamless design flow in VCC even for the deterministic adjustment of a time driven protocol, which usually takes a very significant effort in the development process to define *a priori*. Moreover, it makes it possible to seamlessly change the contingents between event- and time-driven frames within a cluster; therefore, providing the user with ways to explore mixed protocol solutions. Also functional components may be mapped onto a new node with a different bus protocol type.

4.7. Communication Refinement

The local bus memory vectors of all ECUs within a cluster is summarized in the *communication matrix* (CM).

	ECU1	ECU2	ECU3	ECU4
BM1	**Write**	Read	No Access	Read
BM2	Read	**Write/Read**	Read	No Access
BM3	No Access	Read	**Write**	Read
BM4	No Access	Read	**Write**	Read
BM5	Read	Read	Read	**Write/Read**

Table 1. Communication Matrix Example

After accomplishing the mapping of the system functionality, a first CM is generated by VCC in an initial form. This Initial-CM does not require any refinements for running a first performance simulation. The data packages to be sent have exactly the size of one BBM. The activation policy is fully event driven and reflects the scheduling of the behavioral memory write functions. Specific properties, for example the arbitration IDs are assigned randomly by VCC or by a specific user pre-defined

algorithm. The goal of this step is to have a first overview of the bus load and communication matrix complexity and set up the bus communication model although the estimated performance may still be different from a real system.
The communication matrix (CM) constitutes the basic means for the user to refine the bus configuration. A detailed CM shows how messages are packed to frames, what the sender and receiver nodes of the frames are and their properties.
The UCM has three different property categories:

a) Frame properties such as BM collection, activation policy, overhead and inter frame space.
b) Bus controller properties such as node name, collection of the send and receive frames.
c) Bus properties such as bandwidth and communication cycle layout.

4.8. Distribution Techniques

The methodology described so far provides a seamless flow over the different stages in the system design process. Because of the broad variety of mapping possibilities in the proposed methodology and because of the level of abstraction of the UCM - the bus communication layer SW is not modeled – a constellation where two or more nodes send the same message over the network bus, is throughout possible in VCC. This can happen if modules that were running on one ECU in ASCET-SD are distributed over the network. It is clear that more than one sender of the same message over the bus network is not a good VCC design solution and may show unwanted behavioral effects. For example, a sent message that was delayed over the bus could overwrite a more recent message of an ECU.
Due to the broadcasting concept this can never occur in reality. Ideally, the imported models should have only messages that have exactly one writer-module. As this is not guaranteed, it is in the responsibility of the VCC user to investigate this kind of communication constellation to ensure a proper functionality or to redesign the communication manually at this point. Either the designer maps all writer-modules of a specific BM onto the same node, or the situation is solved by creating two (or more) BMs instead of the original one, that then are referenced by only one writer-module. A copy mechanism, which needs to be implemented in the functional components, has to update the correct variables. The advantage is that the copy mechanism is now scheduled by the RTOS of the CPU and not at an unpredictable point in time caused by the bus model. In other words, the communication matrix should always have only one writer per line.

5. Conclusions

In this paper, we have proposed a methodology shift in the design phase of an automotive system. The usage of a virtual integration platform, which allows the distribution of functional components onto an architectural network is key for the shift. The supported levels of abstraction enable a seamless flow in the design process, from a broad variety of partitioning possibilities to refinement stages that allow qualitative performance assessments. The UCM framework provides a high grade of automation, although the designer has still to adapt the bus protocol properties and refine the communication matrix.
The CM of real automotive systems is a result of an extended development process, where many developers and external partners are involved. It requires a lot of experience and knowledge about the functional requirements and the system behavior. Prospectively the import of the necessary bus model properties in form of an appropriate communication matrix, which is available for specific designs in external databases, would add a lot of value to the proposal. As well, a high grade of automation should support the re-use and maintenance of communication configurations that are already refined in VCC to enable an iterative design process and its implementation.
Once a good performance model for the distributed functional components and their communication is generated, hardware relevant protocol characteristics such as specific failure mechanisms can be introduced as additional architectural features to the UCM. The modular definition of the UCM components as finite state machines provides the open infrastructure for adding models of *faulty* modes. This step will become more relevant when a potential system partitioning is found and finally a safety analysis should be performed to confirm the failure concept of the application.

Acknowledgements

The authors like to thank Dr. Maximilian Fuchs, Jürgen Ehret and Peter Schiele from BMW for the valuable discussions and comments. Also, special thanks to Aaron Beverly and Sherry Solden from Cadence Design Systems and the whole Cadence VCC team for the invaluable support and for the implementation of the communication model.

References

[1] Robert Bosch, *CAN Specification Version 2.0*, Technical Report ISO 11898, Robert Bosch GmbH, 1991.

[2] ByteFlight homepage, http://www.byteflight.com/

[3] TTP Forum. *TTP/C specification V0.5*. http://www.ttpforum.org/, 1998.

[4] H. Kopetz, R. Hexel, A. Kruger, D. Millinger, R. Nossal, A. Steininger, C. Temple, T. Fuhrer, R. Pallierer, and M. Krug. A prototype implementation of a ttp/c controller. *Proceedings of SAE Congress and Exhibitio*n, Feb. 1997.

[5] E. Dilger , L.Å. Johansson, H. Kopetz, M. Krug, P. Lidén, G. McCall, P. Mortara, B. Müller. Towards An Architecture For Safety Related Fault Tolerant Systems In Vehicles. ERSEL - European Conference on Safety and Reliability

[6] E. Dilger, T. Führer, B. Müller, S. Poledna, T. Thurner. X-By-Wire: Design of Distributed Fault Tolerant and Safety Critical Applications in Modern Vehicles. VDI - Verein Deutscher Ingenieure

[7] P. Schiele. Transition Methodology from Specifications to a Network of ECUs Exemplarily with ASCET-SD and VCC. SAE Technical Paper Series Nr. 2000-01-0720, 2000.

[8] Translating Models of Computation for Design Exploration of Real-Time Distributed Automotive Applications – Submitted – 2001

[9] S. Edwards, L. Lavagno, E. Lee, A. Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Methods, Validation and Synthesis. Proceedings of the IEEE, vol. 85 (n.3) - March 1997, p366-290.

[10] Vector Informatik. *Calibration of Electronic Control Units via CAN.* http://www.vector-informatik.de/english/products/index.html, Canape, 2000.

[11] Cadence Inc. Virtual Component Codesign Product Documentation. Cadence Inc., 1998.

[12] ETAS GmbH, Whitepaper ASCET-SD-, ETAS GmbH, 1998.

[13] Matlab. Homepage Technical report, Osek, http://www.iiit.etec.uni-karlsruhe.de /osek.

[14] OSEK/VDX Organisation. OSEK/VDX Operating System Specification 2.1. http//www.osex-vdx.org.

[15] L. Lavagno, A. Sangiovanni-Vincentelli and E. Sentovich. Models of Computation for Embedded System Design. 1998 NATO ASI, Proceedings on System Synthesis, Il Ciocco, 1998.

[16] E. Lee, A. Sangiovanni-Vincentelli. Comparing Models of Computation. Proceedings of ICCAD, 1996.

[17] Reliable estimation of execution time of embedded software. Submitted, 2001.

[18] I. Gutkin, P. Giusto, J. Ehret. Modelling the CAN bus within the VCC environment. Proceedings of the International Conference on Parallel and Distributed, Processing Techniques and Applications, 1999.