# Design and Development of Component-Based Embedded Systems for Automotive Applications

Marco Di Natale

Scuola Superiore S. Anna, Pisa, Italy
marco@sssup.it

**Abstract.** Automotive software systems are characterized by increasing complexity, tight safety and performance requirements, and need to be developed subject to substantial time-to-market pressure. Model- and component-based design methodologies can be used to improve the overall quality of software systems and foster reuse. In this work, we discuss challenges in the adoption of model-based development flows, and we review recent advances in component-based methodologies, including existing or upcoming standards, such as the MARTE UML profile, ADL languages and AUTOSAR. Finally, the paper provides a quick glance at results on a methodology based on virtual platforms and timing analysis to perform the exploration and selection of architecture solutions.

## 1 Introduction

The automotive domain is experiencing evolutionary changes because of the demand for new advanced functions, technological opportunities and challenges, and organizational issues. The increased importance and value of electronics systems and the introduction of new functions with unprecedented complexity, timing and safety issues are changing the way systems are designed and developed and are bringing a revolution in the automotive supply chain. New standards and methodologies are being developed that will likely impact not only automotive electronics systems, but also other application domains, which share similar problems.

The automotive supply chain is currently structured in tiers

- ***Car manufacturers (or Original Equipment Manufacturers OEMs)***.
- ***Tier 1 suppliers*** who provide electronics subsystems to OEMs.
- ***Tier 2 suppliers*** e.g., chip manufacturers, IP providers, RTOS, middleware and tool suppliers, who serve OEMs and more likely Tier 1 suppliers.
- ***Manufacturing suppliers*** providing manufacturing services.

Currently, automotive systems are an assembly of *components* that are designed and developed in house or, more often, by Tier 1 suppliers. These subsystems have traditionally been loosely interconnected, but the advent of active-safety and future safety-critical functions, including by-wire systems, and the interdependency of these functions is rapidly changing the scenario. Furthermore,

F. Kordon and T. Vardanega (Eds.): Ada-Europe 2008, LNCS 5026, pp. 15–29, 2008.

subsystems are developed using different design methods, software architectures, hardware platforms, real-time operating systems and middleware layers. To give an idea of architecture complexity, the number of Electronic Control Units (ECUs) in a vehicle is presently in the high tens, and further increasing. In the face of this scenario, OEMs need to understand and control the emerging behavior of the complex distributed functions, resulting from the integration of subsystems. This includes both functional and para-functional properties, such as timing and reliability. The supply process, traditionally targeted at simple, black-box integrated subsystems, will evolve from the current situation, where the specifications issued to the OEMs consist of the message interface and general performance requirements, to more complex component specifications that allow plug-and-play of portable software sub-systems.

The essential technical problem to solve for this vision is the establishment of standards for interoperability among IPs, both software and hardware, and tools. AUTOSAR [1], a world-wide consortium of almost all players in the supply chain of automotive electronics, has this goal very clear in mind. However, several issues need to be solved for function partitioning and subsystem integration, in the presence of real-time and reliability requirements, including:

- **Composability and refinement of subsystems.** The automotive industry together with the avionic industry was the first to embrace model-based design, as a tool to remove coding errors and to speed up the software development process. This approach was made possible by the introduction of powerful simulation tools where the functionality of the system is captured with a mathematically-oriented formalism, such as Simulink [12]. However, the definition of a process that goes from system-level to component models, in which behaviors are formally and unambiguously defined, such that they can verified at design time, and that allows for automatic code generation, is a quite challenging task. Such a process would indeed require that all relevant functional and non-functional constraints and properties are captured by the models used at all levels and that model semantics is preserved at each refinement/transformation steps. In Section 2 we discuss the issues related to model-based development and we review the impact of AUTOSAR on the design methodology.
- **Time predictability.** This issue is related to the capability of predicting the system-level timing behavior (latencies and jitter), resulting from the synchronization between tasks and messages, but also from the interplay that different tasks can have at the RTOS level and the synchronization and queuing policies of the middleware. The timing of end-to-end computations depends, in general, on the deployment of the tasks and messages on the target architecture and on the resource management policies. In Section 3, we review issues in this domain.
- **Dependability.** The deployment of the functions onto the ECUs and the communication and synchronization policies must be selected to meet dependability targets. A system-level design tool should integrate support for design patterns suited to the development of highly-reliable systems with

fault containment both at the functional level and at the timing level. Such tools should also support the automatic construction of fault-trees to compute the probability of a hazard occurrence.

Complex automotive functions, including active-safety and safety-critical systems, are characterized by non-functional requirements, including timing and performance, requirements for safety, and cost, together with reusability and extensibility of the architecture artifacts. System-level analysis and new modeling and analysis methods and tools are not only needed for predictability and composability when partitioning end-to-end functions, but also for providing guidance and support in the evaluation and selection of the electronics and software architectures. In Section 4, we provide the description of a design methodology based on virtual platforms in which models of the functions and of the possible solutions for the physical architecture are defined and matched to select the best possible hardware platform with respect to performance. Opportunities for the automatic synthesis of the software architecture are also discussed.

## 2 Model-Based Design, Composability and AUTOSAR

Model-based design methodologies are increasingly adopted for improving the quality and the reusability of software. A model-based environment allows the development of control and dataflow applications in a graphical language that is familiar to control engineers and domain experts. The possibility of defining components (subsystems) at higher levels of abstraction and with well defined interfaces allows separation of concerns and improves modularity and reusability. Furthermore, the availability of verification tools (often by simulation) gives the possibility of a design-time verification of the system properties. However, when considered in the context of a design flow that starts from the early stages of architecture exploration and analysis and supports complex interacting functions with real-time requirements, deployed on a distributed architecture, most modern tools for model-based design have a number of shortcomings

*Lack of separation between the functional model and the architecture model:* such a separation is fundamental for exploring different architecture options with respect to a set of functionality and for reusing an architecture platform with different functions.

*Lack of support for the definition of the task and resource model:* Most model-based flows support the transition from the functional model directly to the code implementation. The designer has limited control on the generation of the task set and the task and resource model is scantly addressed. Placement of tasks in a distributed environment is typically performed at the code level. The specification of the task and message design and of the resource allocation is necessary to evaluate the timing and dependability properties of the system.

*Insufficient support for the specification of timing constraints and attributes:* The definition of end-to-end deadlines, as well as jitter constraints is often not considered by modeling languages.

*Lack of modeling support for the analysis and the back-annotation of scheduling-related delays:* Most tools support simulation and verification of the functional model, which is typically based on an assumption of zero communication and computation delays. The definition of the deployment on a given architecture allows the analysis of the delays caused by resource sharing. In a sound design flow, tools should support this type of analysis, and the communication and scheduling delays should be back-annotated into the model to verify the performance of the function on a given architecture solution.

*Issue of semantics preservation:* The generation of the code starting from a model description is not always performed in such a way that the original semantics is preserved. It is important that designers and developers understand under what conditions the code generation stage can preserve the model semantics and what are the implications of an incorrect implementation.

Some of these issues can be reviewed in more detail, with reference to an abstract design flow, which encompasses all the refinement steps, from the system-level view, down to the code implementation (Figure 1).

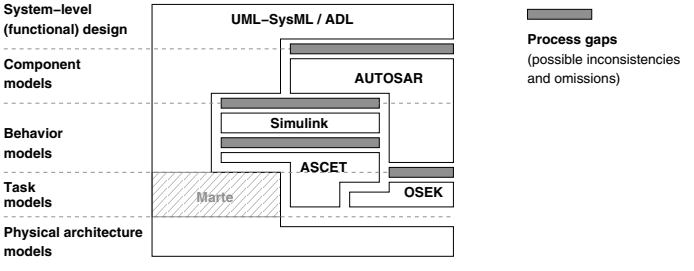


**Fig. 1.** An abstract development flow: standards and process gaps

The highest level in the description of the system corresponds to the early decomposition of high-level end-to-end functions (typically derived from user requirements). The system description is characterized by a behavior specification, but also by reliability and time requirements. Candidate languages and standards for system-level modeling, which may include a first-level decomposition into major functional blocks or subsystems, are the Unified Modeling Language (UML) and its specialized profile SysML, and Architecture Description Languages (ADL), like the EAST/ADL [22].

In order to allow for the specification and modeling of time and reliability requirements, UML has recently been extended by two profiles (specialized restrictions of the language semantics), the MARTE profile for the specification of timing requirements and properties [20] and the UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms [21]. Both standards are expected to provide support for expressing time and reliability properties and requirements. However, because of the need of dealing with the generality of the UML language, they are typically cumbersome (the MARTE profile

specification is currently 658 pages long) and must rely on faithful and efficient implementation by tool vendors, which presently cannot be guaranteed.

Subsystem specifications are then passed from OEMs to Tier-1 suppliers, who are responsible for their development. Although UML can still be used at this stage, the AUTOSAR development partnership [1], including several OEM manufacturers, Tier-1 suppliers, tool and software vendors, has been created with the purpose of developing an open industry standard for component specification and later integration.

To achieve the technical goals of modularity, scalability, transferability and re-usability of functions, AUTOSAR provides a common software infrastructure based on standardized interfaces for the different layers. The current version of the AUTOSAR model includes a reference architecture and interface specifications. AUTOSAR has been focused on the concepts of location independence, standardization of interfaces and portability of code. While these goals are undoubtedly of extreme importance, their achievement is not a sufficient condition for improving the quality of software.

The current specification has at least two major shortcomings. The AUTOSAR metamodel, as of now, is affected by the *lack of a clear and unambiguous communication and synchronization semantics* and the *lack of a timing model*. The AUTOSAR consortium recently acknowledged that the specification was lacking a formal model of components for design time verification of their properties. As a result, the definition of the AUTOSAR metamodel was started. Similarly to UML, the AUTOSAR metamodel is sufficiently mature in its static or structural part, but offers an often incomplete behavioral description, which is planned for significant updates in its upcoming revision. Furthermore, the standard does not address adequately issues related to timing and performance, therefore underestimating the complexity of current and future applications, in which component interactions generate a variety of timing dependencies due to scheduling, communication, synchronization, arbitration, blocking, buffering. The reuse of a component requires that the behavior of the reused components and the result of the composition with respect to time can be predicted in the new configuration. If this problem is not addressed, the composition will eventually lead to (possibly transient) timing problems. The definition of a timing model for AUTOSAR and the development of a standardized infrastructure for the handling of time specifications is the objective of the ITEA project TIMMO, which started in April 2007 and includes car manufacturers like Audi, PSA, Volvo Technology and Volkswagen, as well as electronics and tool suppliers, including Bosch, Continental, ETAS, Siemens VDO, Symtavision and TTTech.

On a separate context, a discussion of the issues that need to be considered when mapping UML into AUTOSAR (and vice-versa) and the possible gaps and inconsistencies in this transformation can be found in [22].

Components must be characterized by (a set of) behavior requirements and a corresponding internal behavior model. In AUTOSAR, the behavior of AtomicSoftwareComponents is represented by a set of RunnableEntities (Runnables for short) communicating with each other over the ports of the container structural

entities (components). Like in UML, structural and behavioral entities are linked to each other but are kept separated. AUTOSAR provides several mechanisms for Runnables to access the data items for sender/receiver communication and the services of client/server communication, *but the synchronization and timing semantics in the execution of Runnables is only partly specified*. In AUTOSAR, the runtime environments (RTEs) of each ECU are responsible for establishing the communication between the Runnables (local or remote) and triggering their execution using the following events:

- *Timing Event* triggering periodical execution of Runnables.
- *DataReceivedEvent* upon reception on a Sender/Receiver communication.
- *OperationInvokedEvent* for invocation of Client/Server service.
- *DataSendCompleteEvent* upon sending a Sender/Receiver communication.
- *WaitPoint* allows blocking a runnable while waiting for an Event.

Behavioral models are not supported in AUTOSAR, but the standard relies on external behavioral modeling tools like Simulink and ASCET, which brings the issue of the composition of (possibly heterogeneous) models. Therefore, any integration environment (EAST-ADL2 [22] is an example), must define the triggering and execution semantics of functions. This semantic should be deterministic to allow execution verification.

An example of the possible issues in the definition of the execution semantics (and also an example of model translation issues) can be found in Figure 2 (adapted from [22]), in which three models of a control algorithm, respectively in Simulink, UML (activity diagram) and AUTOSAR are represented.

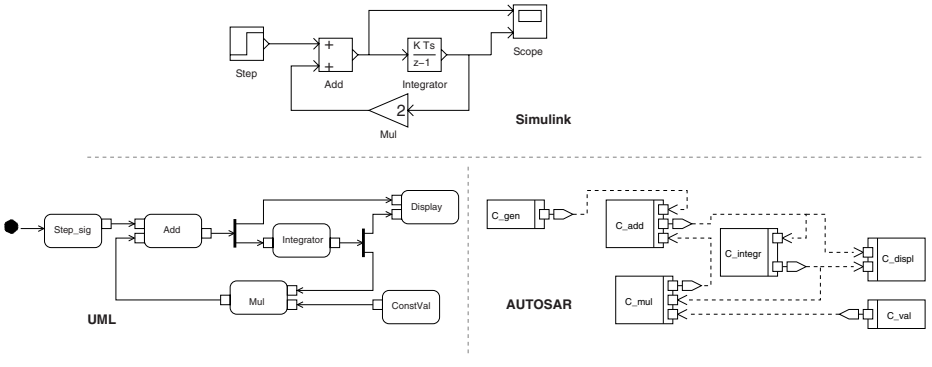


**Fig. 2.** Model-to-model transformation issues

Despite a similarity in their structure, the three models differ in the execution order of the actions. Contrary to UML activity diagrams, in Simulink, blocks are not executed in lexicographic order. In Simulink, blocks for which the output does not depend on the input at any given time, such as the *Integrator* in the Figure, can be executed before the others. Indeed, the simulation behavior of the depicted Simulink model will start with the output of the *Integrator*, and

then continue with the *Mul* and *Add* blocks. In the UML activity diagram, the *Add* action will run before the *Integrator* block. The difference in the execution order may lead to different model behaviors and different simulation results.

In UML, in fact, the triggering order is defined when operations are called, but the execution order is undefined in the case of communication by data (streams) received on ports. SysML tries to define the semantics of data reception on ports, but the bindings between behavior parameters, and either the flow properties or the containing block properties are a semantic variation point [22]. In conclusion, for triggering semantics that differ from the loose UML standard definition, designers are required to explicitly define their own semantics by introducing stereotypes (specializing generic UML concepts by additional constraints and tagged values) in a dedicated profile .

However, execution order is not the only problem with our example. In Simulink, all blocks react at the same time and produce outputs in zero time (according to the Synchronous Reactive semantics), which leads to possible problems when the model has algebraic loops (instantaneous cyclic dependencies of signals from themselves). In this case, the system may have a fixed point solution or the model may be simply not correct. The definition of a Synchronous Reactive semantics in UML is probably possible by leveraging the Marte profile, but it would require the adoption of a stereotyped (discrete) time model. Additional diagrams are probably required to synchronize triggers and/or enforcing the correct execution order (possibly state diagrams). Finally, in case other types of timing constraints on end-to-end computations exist, an additional sequence diagram (and a stereotyped notation for timed events) would be required as well.

Finally, the AUTOSAR specification is based on the OSEK specification for Operating Systems. In an OSEK system, tasks are executing concurrently with priorities and subject to preemption. Hence, special care must be taken in the code generation stage, when the structural and behavioral part of the specification are mapped into concurrent tasks using automatic code generation techniques. Runnables and functional blocks must be executed by tasks in such a way to ensure data consistency of the variables implementing the communication links, and also time determinism in the execution of blocks. Furthermore, the implementation must guarantee the enforcement of the set of partial orders in the execution of blocks, as determined by the model semantics.

## 3 Timing Predictability, Timing Isolation and Standards

The automotive domain has been traditionally receptive to methods and techniques for timing predictability and time determinism. The standard Controller Area Network (CAN) bus [6] for communication is based on the concept of a deterministic resolution of the contention and on the assignment of priorities to messages. The OSEK standard for real-time operating systems [14] not only supports predictable priority-based scheduling [10], but also bounded worst-case blocking time through an implementation of the immediate priority ceiling protocol [17] and the definition of non-preemptive groups [18] for a possible further

improvement of some response times and to allow for stack space reuse. In the absence of faults, and assuming that the worst-case execution time of a task can be safely estimated, these standards allow the designer to predict the worst-case timing behavior of computations and communications.

Priority-based scheduling of tasks and messages fits well within the traditional design cycle, in which timing properties are largely verified a-posteriori and applications require conformance with respect to worst-case latency constraints rather than tight time determinism. Furthermore, control algorithms are designed to be tolerant with respect to small changes in the timing behavior and to the nondeterminism in time that possibly arises because of preemption and scheduling delays [7], or even possibly to overwritten data or skipped task and message instances because of temporary timing faults. Finally, although formally incorrect, there is a common perception that small changes in the timing parameters (decreased periods and/or wrong estimates of the computation times) typically only result in a graceful degradation of the response times of tasks and messages and that such degradation will in any case preserve the high priority computations.

These assumptions can be misleading and faulty. The worst-case response times of tasks and messages, scheduled on priority-based systems, such as those defined by the OSEK and CAN standards can be computed using a fixed point formula. For a periodic task $\tau_i$, activated with period $T_i$ and worst-case computation time $C_i$, the worst-case response time $r_i$ is given by (in case $r_i \leq T_i$, the general case is discussed in [11])

$$r_i = B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil C_j \tag{1}$$

Where $j \in hp(i)$ means all the indexes of the generic tasks $\tau_j$ with a priority higher than $\tau_i$ and $B_i$ is the worst-case blocking time in which the task cannot execute because of an activity (typically a critical section or an interrupt handler) executed on behalf of a lower priority task. The worst-case latency of a CAN message can be upper bound as shown in [9], where the factor $B_i$ is the largest transmission time of any message frame.

$$\begin{aligned} w_i &= B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i}{T_j} \right\rceil C_j \ (w_i > 0) \\ r_i &= w_i + C_i \end{aligned} \tag{2}$$

In the face of the development of larger and more complex applications, which are deployed with a significant amount of parallelism on each ECU and consist of a densely connected graph of distributed computations, and of new safety-critical functions, which require tight deadlines and guaranteed absence of timing faults, a new rigorous science needs to be established. A number of issues need to be considered with respect to the current standards and the use of priority based scheduling of tasks and messages.

- Priority-based scheduling can lead to discontinuous behavior in time and timing anomalies. The dependency of the response time of a lower priority task or message with respect to the computation time (or period) of a higher priority task is not linear and not even continuous. Furthermore, especially in distributed systems, it may even be possible that shorter computation times result in larger latencies [16]. A recently developed branch of worst-case timing analysis is focusing on sensitivity analysis [5][16] as a means for understanding which computation and communication loads are critical for the preservation of deadlines.
- Variability of the response times between the worst-case and the best case scenario, together with the possible preemptions, can lead to the violation of time-deterministic model semantics in the implementation of software models by priority scheduled tasks and messages [4].
- Extensibility and (to some degree) tolerance with respect to unexpectedly large resource requirements from tasks and messages that is allowed by priority-based scheduling comes at the price of additional jitter and latency and lack of timing isolation.
- Future applications, including safety critical (x-by-wire) and active safety need shorter latencies and time determinism (reduced jitter) because of increased performance. The current model for the propagation of information, based on *communication by periodic sampling*, among non-synchronized nodes has very high latency in the worst-case and a large amount of jitter between the best case and the worst-case delays. Even if communication-by-sampling can be formally studied and platform implementations can be defined to guarantee at least some fundamental properties of the communication flow (such as data preservation), time determinism is typically disrupted and the application must be able to tolerate the large latencies caused by random sampling delays.
- The deployment of reliable systems requires timing isolation in the execution of the software components, and protection from timing faults. One of the major downsides of priority-based scheduling of resources is that faulty high priority computation or communication flows can easily obtain the control of the ECU or the bus, subtracting time from lower priority tasks or messages. For example, an excessive request of computation time from any high priority task impacts the response time of lower priority tasks on the same ECU. Timing protection is even more important in the light of AUTOSAR, when components from Tier1 suppliers are integrated into the same ECU, leveraging the standardization of interfaces, and faulty behaviors (functional and temporal) need to be contained and isolated.
- The development of future applications will also require the enforcement of composability and compositionality not only in the functional domain but also for para-functional properties of the system, including the timing behavior of the components and their reliability. (see next section)

Time-based schedulers, including those supported by the FlexRay and OSEK-Time [13] standards force context switches on the ECUs and the assignment of

the communication bus at predefined points in time, regardless of the outstanding requests from the tasks for computation and communication bandwidth. Therefore, they are better suited to provide temporal protection, except that the enforcement of a strict time window for the execution and communication requires a much better capability of the designer in predicting the worst case execution times of tasks so that the execution window can be appropriately sized, and guardians are needed to ensure that an out-of-time transmission will not disrupt the communication flow on the bus.

## 4 Platform-Based Design for Architecture Selection

Platform-based design requires/entices the identification of clear abstraction layers and a design interface that allows for the separation of concerns between the refinement of the functional architecture specification and the abstractions of possible implementations. The application-layer software components are thus decoupled from changes in microcontroller hardware, ECU hardware, I/O devices, sensors, actuators, and communication links. The basic idea is captured on the left side of Figure 3. The vertex of the two cones represents the combination of the functional model and the architecture platform. Decoupling the application-layer logic from dependencies on infrastructure-layer hardware or software enables the application-layer components to be reused without changes across multiple vehicle programs. A prerequisite for the adoption of the platform-based

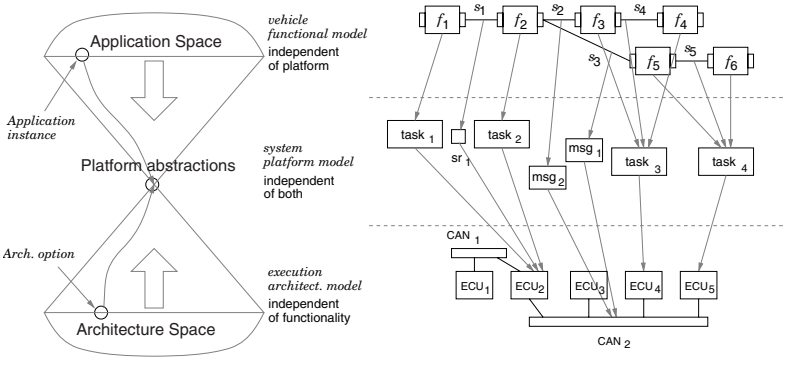


**Fig. 3.** Platform-based design and models

design and of the meet-in-the-middle approach is the definition of the right models and abstractions for the description of the functional platform specification and for the architecture solutions at the top and the bottom of the hourglass of Figure 3. The platform interface must be isolated from lower-level details but, at the same time, must provide enough information to allow design space exploration with a fairly accurate prediction of the properties of the implementation. This model may include size, reliability, power consumption and timing; variables that are associated to the lower level abstraction.

Design space exploration consists of seeking the optimal mapping of the system platform model into the candidate execution platform instances. The mapping must be driven by a set of methods and tools providing a measure of the fitness of the architecture solutions with respect to a set of feasibility constraints and optimization metric functions. This work focuses on timing constraints and metrics. In Section 4.2, we discuss the possibility for the automatic selection of part of the platform configuration by software tools. The technology, however, is not mature yet for a full synthesis of the task and message design and the definition of the architecture mapping. The approach that is currently viable is a what-if analysis where different options are selected as representatives of the principal platform options and evaluated according to measurable metrics.

**Functional Model**
The starting point for the definition of ECS based vehicle architecture is the specification of the set of features that the system is expected to provide. A feature is a very high level description of a system capability, such as an active-safety function. The software component of each feature is further developed by control engineers who devise algorithms fulfilling the design goals. Typically, these algorithms are captured by a hierarchical set of block diagrams produced with commercial tools for control design. The functional model(s) are created from the decomposition of the feature in a hierarchical network of components encapsulating a behavior, within a provided and required interface, expressed by a set of ports or by a set of methods with the corresponding signature. This view abstracts from the details of the functional behavior and models only the interface and the communication semantics, including the specification of the activation signal for each functional block, be it periodic, sporadic, or arriving, together with the incoming data, from one of its input ports. The functional description is further endowed with the required constraints. For example, timing constraints are expressed in the context of the functional architecture by adding end-to-end deadlines to the computation paths, maximum jitter requirements to any signal and time correlation constraints between any signal pair.

**Architecture Model**
The model of the architecture is hierarchical and captures the logical topology of the vehicle network, including the communication buses, such as CAN [6] and time triggered links, the number of processors for each ECU and the resource management policies that control the allocation of each ECU and BUS. At this stage, the hardware and software resources that are available for the execution of the application tasks and the resource allocation and scheduling policies must also be specified.

**Platform Model**
The system platform model is where physical concurrency and resource requirements are expressed. The system platform model(s) are a representation of the mapping process. Tasks are defined as units of computation processed concurrently in response to environment stimuli or prompted by an internal

clock. Tasks cooperate by exchanging messages and synchronization or activation signals and contend for use of the processing and communication resource(s) (e.g., processors and buses) as well as for the other resources in the system. The system platform model entities are, on one hand, the implementation of the functional model and, on the other hand, are mapped onto the target hardware. The mapping phase consists of allocating each functional block to a software task and each communication signal variable to a virtual communication object (right side of Figure 3). The task activation rates are derived from the functional blocks activation rates. As a result of the mapping of the platform model into the execution architecture, the entities in the functional models are put in relation with timing execution information derived by worst-case execution time analysis or back-annotations extracted from physical or virtual implementation. Given a mapping, it is possible to determine which signals are local (because the source and destination functions are deployed onto the same ECU) and which are remote and, hence, need to go over the network. Each communication signal is therefore mapped to a message, or to a task private variable or to a protected shared variable. Each message, in turn, is mapped to a serial data link. The mapping of the threads and message model into the corresponding architecture model and the selection of resource management policies allows the subsequent validation against timing constraints.

### 4.1 What-If Analysis

The procedure for architecture selection and evaluation is a what-if iterative process. First, the set of metrics and constraints that apply to the design is defined. Then, based on the designer's experience, a set of initial candidate architecture configurations is produced. These architectures are evaluated and, based on the results of the quantitative analysis, a solution can be extracted from the set as the best fit. If the designer is not satisfied with the result, a new set of candidate architectures can be selected. The iterative process continues, until a solution is obtained. The intervention of the designer is required in two tightly related stages of the exploration cycle. The designer must provide the initial set of architecture options. After the options have been scored and annotated by the analysis and simulation tools, the designer must understand the results of the analysis and select the architecture options that are the best fit to the exploration goals and (more importantly) understand the results of the analysis to add other options to the next set of configurations that needs to be evaluated. The set of analysis methods that are available for architecture evaluation are:

- Evaluation of end-to-end latency and schedulability against deadlines for chains of computations spanning tasks and messages scheduled with fixed priority [19].
- Sensitivity analysis for tasks and messages scheduled with fixed priorities and sensitivity analysis for resources scheduled with fixed priorities [5].

– Evaluation of message latencies in CAN bus networks [8].
– System level simulation of time properties and functional behaviors (based on the Metropolis engine [3]).
– Analysis of fault probability and cutsets (conditions leading to critical faults) based on fault trees.

### 4.2 Automatic Configuration of the SW Architecture

The mapping of the functional model into the execution platform is part of the platform-based design referred in the previous sections and of the Y-chart design flow [2] shown in Figure 4, where the application description and architectural description are joined in an explicit mapping step. The mapping definition and the creation of the task and resource models can be performed in several ways. In single processor systems, the problem is usually very simple, and often subsumed by the code generation phase. In distributed architectures, the design of the software architecture is a more complex task and it is very often delegated to the experience of the designer. When a software implementation is not feasible because of resource constraints, design iterations may be triggered and the functional model itself or the architecture configuration may be modified.

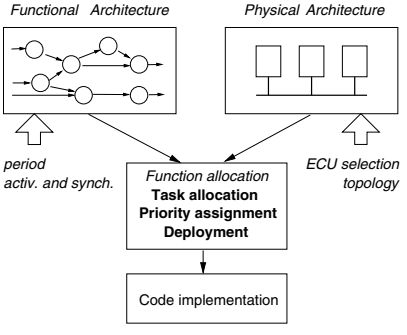


**Fig. 4.** Design flow stages and period synthesis

Once the function and the architecture are defined, there are several possible options for the intermediate layer, and automated tools can provide guidance in the selection of the optimal configuration with respect to the timing constraints and a performance-related metric function.

The mapping consist of the following stages: function to task mapping; task to ECU deployment and signal to message mapping, and, finally, of the assignment of priorities to tasks and messages. When iterations are required on the functional model, a different selection of the execution periods of the functions, or different synchronization and communication solutions may be explored.

We defined solutions based on mixed integer linear programming (MILP) and geometric programming (GP), respectively, for the problem of optimizing the activation mode of tasks and messages [19] and the selection of task periods [8].

The effectiveness of these approaches has been demonstrated by application to on an experimental vehicle system case. We are currently exploring approximated solutions for the selection of a feasible mapping of tasks to ECUs and signals to messages and the assignment of priorities to tasks and messages.

## 5 Conclusions

The structure of the automotive electronic industry and the state-of-the-art of automotive electronics design methodology was summarized. Issues on model-based design, composability and timing protection and a quick look at the opportunities and the limitations of the existing standards were also discussed. We concluded with a proposed methodology for architecture exploration, based on virtual platforms and the separation of functional and physical architecture models. We envision the availability of an intermediate platform layer in which the functions are mapped into the architecture option and the result is evaluated with respect to para-functional metrics and constraints related to timing, dependability and cost. It will be of highest importance to support the evolution of the automotive standards to ensure the feasibility of a correct and robust design flow based on virtual platform.

## References

1. AUTOSAR. Consortium web page, `www.autosar.org`
2. Balarin, F., et al.: Hardware-Software Co-Design of Embedded Systems – The Polis Approach. Kluwer Academic Publishers, Dordrecht (1997)
3. Balarin, F., Lavagno, L., Passerone, C., Watanabe, Y.: Processes, interfaces and platforms. Embedded software modeling in Metropolis. In: Proc. of the 2nd ACM EMSOFT, Grenoble, France (October 2002)
4. Baleani, M., Ferrari, A., Mangeruca, L., Sangiovanni-Vincentelli, A.: Efficient embedded software design with synchronous models. In: Proc. of the 5th ACM EMSOFT. ACM Press, New York (2005)
5. Bini, E., Natale, M.D., Buttazzo, G.: Sensitivity analysis for fixed-priority real-time systems. In: Euromicro ECRTS, Dresden, Germany (June 2006)
6. Bosch, R.: Controller area network specification, version 2.0. Stuttgart (1991)
7. Caspi, P., Benveniste, A.: Toward an approximation theory for computerised control. In: Sangiovanni-Vincentelli, A.L., Sifakis, J. (eds.) EMSOFT 2002. LNCS, vol. 2491, pp. 294–304. Springer, Heidelberg (2002)
8. Davare, A., Zhu, Q., Natale, M.D., Pinello, C., Kanajan, S., Sangiovanni-Vincentelli, A.: Period optimization for hard real-time distributed automotive systems. In: Design Automation Conference, San Diego, CA (June 2007)
9. Davis, R.I., Burns, A., Bril, R.J., Lukkien, J.J.: Controller area network (can) schedulability analysis: refuted, revisited and revised. Real-Time Systems 35, 239–272 (2007)
10. Harbour, M.G., Klein, M., Lehoczky, J.: Timing analysis for fixed-priority scheduling of hard real-time systems. IEEE Transactions on Software Engineering 20(1) (January 1994)

11. Lehoczky, J.P., Sha, L., Ding, Y.: The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proc. of the 10th RTSS, Santa Monica, CA (December 1989)
12. Mathworks. The Mathworks Simulink and StateFlow User's Manuals, http://www.mathworks.com
13. OSEK. OSEK/VDX Steering Committee: Time-Triggered Operating System, http://www.osek-vdx.org
14. OSEK. OS vers. 2.2.3 specification (2006), http://www.osek-vdx.org
15. DSpace TargetLink product page, http://www.dspaceinc.com
16. Racu, R., Ernst, R.: Scheduling anomaly detection and optimization for distributed systems with preemptive task-sets. In: 12th RTAS, San Jose (April 2006)
17. Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: An approach to real-time synchronization. IEEE Transactions on computers 39(9), 1175–1185 (1990)
18. Wang, Y., Saksena, M.: Scheduling fixed priority tasks with preemption threshold. In: Proc. of the RTCSA Conference (December 1999)
19. Zheng, W., Natale, M.D., Pinello, C., Giusto, P., Sangiovanni-Vincentelli, A.: Synthesis of task and message activation models in real-time distributed automotive systems. In: Proc. of the DATE conference, Nice, April 15-18 (2007)
20. Object Management Group MARTE profile: Modeling and Analysis of Real-time and Embedded systems, http://www.omgmarte.org/
21. Object Management Group UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms, http://www.omg.org/cgi-bin/doc?ptc/2006-12-02
22. ATESST Advanced Traffic Efficiency and Safety through Software Technology Deliverable 3.2 Report on behavior modeling with the EAST-ADL 2.0 (July 12, 2007)