

# Embedded System Design for Automotive Applications

*Alberto Sangiovanni-Vincentelli*, University of California, Berkeley

*Marco Di Natale*, Scuola Superiore S. Anna, Pisa

**To optimize the system design and allow for plug-and-play of subsystems, automotive electronic system architecture evaluation and development must be supported with a robust design flow based on virtual platforms.**

**T**oday, though still relatively stable, the roles of carmakers and their suppliers are undergoing a period of stress caused by the increased importance and added value of electronics. The automotive supply chain includes

- car manufacturers—or OEMs—such as GM, Ford, DaimlerChrysler, and Toyota, who provide the final product to the consumer market;
- Tier 1 suppliers—such as Bosch, Contiteves, Siemens, Nippondenso, Delphi, and Magneti Marelli—that provide subsystems such as power train management, suspension control, and brake-by-wire devices to OEMs;
- Tier 2 suppliers—chip manufacturers such as Freescale, Infineon, ST, and Renesas; IP providers such as ARM; and real-time operating system suppliers such as WindRiver and ETAS—who serve both OEMs and Tier 1 suppliers; and
- manufacturing suppliers such as Flextronics and TSMC.

Because of liability issues, automakers generally limit outside manufacturing to non-safety-critical verticals. The standard approach for OEMs is to develop systems by assembling components that have been completely or partly designed and developed by Tier 1 suppliers. However, these suppliers increasingly are shifting toward outsourcing their manufacturing.

The supply process traditionally has been targeted at simple, black-box integrated subsystems in which

requirements capture and OEM-issued specifications consisted of the message interface's periods and general performance requirements, but without a detailed definition of timing and synchronization properties and of the communication protocols' requirements. As a result, the integration of subsystems is done routinely, albeit in a heuristic and ad hoc way. The resulting lack of an overall understanding of the subsystems' interplay, and the difficulties encountered in integrating very complex parts, make system integration a very challenging job. The "Car Electronics Architecture" sidebar provides more information on the complexity of modern architectures.

## CHALLENGES

Novel methods and tools for system-level analysis and modeling are needed not only for predictability and composability when partitioning end-to-end functions at design time (and later, at system integration time), but also for providing guidance and support to the designer in the very early stage where the electronics and software architectures of product lines are evaluated and selected. The critical architecture-evaluation and -selection design-process phase affects profoundly a product line's cost, performance, and quality.

Architecture selection typically is performed years in advance of subsystem development and integration. In this process, models of the functions and possible solutions for the physical architecture must be defined and matched to evaluate quality and select the best possible hardware platform with respect to performance, reliability, and cost metrics and constraints.



Given the high cost of research, training, and possibly license acquisition for system-level design, using a coherent set of models, methods, and tools during a product's or platform's entire lifetime is desirable. This extends from the architecture-analysis stage to system partitioning and design, and includes model-based development, with its automatic middleware and application code generation steps, and the final integration, testing, and validation stages.

Optimizing automotive electronics system design requires standards in the software and hardware domains that allow for plug-and-play of subsystems. The ability to integrate subsystems will then become a commodity item, available to all OEMs. An OEM's competitive advantage will increasingly rely on novel and compelling functionalities. 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,<sup>1</sup> a worldwide consortium of most of the players in the automotive domain electronics supply chain, has this goal clearly in mind.

However, technical and business challenges must first be overcome. In particular, from a technical viewpoint, while sharing algorithms and functional designs seems feasible at this time, the sharing of safety-critical and hard real-time software is difficult, even assuming substantial improvements in design methods and technology. Several issues must be resolved for function partitioning and subsystem integration in the presence of real-time and reliability requirements. These include the following:

- **Time predictability.** This issue relates to the capability of predicting the system-level timing behavior (latencies and jitter) resulting from the synchronization between tasks and messages, as well as from the interplay that different tasks can have at the real-time operating system (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.
- **Dependability.** Deploying functions onto the system engine control units (ECUs) and determining communication and syn-

## Car Electronics Architecture

A typical modern vehicle contains between a dozen and nearly 100 electronic control units (ECUs).<sup>1</sup> Current electronics systems are typically partitioned by domains. There are two main classes of electronic systems: hard-real-time control of mechanical parts and information-entertainment. The first category includes

- chassis control;
- automotive body, including components such as interior air conditioning, dashboard, power windows, and control subsystems;
- powertrain, including the engine, transmission, and emission and control systems; and
- active safety control.

The second category includes information management, navigation, computing, external communication, and entertainment.


Each domain has its own requirements for computation speeds, time scales, reliability, flexibility, and extensibility. Today, powertrain applications pose the most demanding challenge in terms of real-time constraints and computational power, with activation period requirements going down to a few milliseconds at high engine speeds.

New active safety applications, currently planned to execute at slower rates—typically in the range of 20 to 100 ms—at each stage, will pose new challenges because of their high distribution, complexity, and interoperability. The typical power train ECU today relies on a 32-bit microcontroller running at hundreds of MHz, while the rest of the real-time subsystems use a 16-bit microcontroller running at less than 1 MHz, with memory requirements reaching up to 2 Mbytes for a few complex subsystems. The next generation, however, is rapidly moving toward widespread use of 32-bit ECUs, with some running at more than 100 MHz. Multicore ECUs will likely provide the next-generation solution for applications requiring high reliability.

For communications, a typical vehicle today contains two or three controller area network buses, with rates from 25 to 500 Kbytes, two or three lower-speed local interconnect network buses, and, optionally, some dedicated high-speed links for infotainment. Experimental vehicles now being developed have up to 10 CAN buses, with additional buses almost invariably providing 500-Kbps links. A further increase in the number of buses is unlikely because of the additional gateways and consequent increased latencies and jitter. This is why FlexRay, aside from being a possible solution for future highly reliable communication needs, is already required for high-speed, highly deterministic communication.

### Reference

1. J.A. Cook et al., "Control, Computing and Communications: Technologies for the Twenty-First Century Model T," *Proc. IEEE*, special issue on automotive power electronics and motor drives, vol. 95, no. 2, 2007, pp. 334-355.



chronization policies must be done with a view to meeting dependability targets. A system-level design tool should integrate support for design patterns suited to the development of highly reliable systems with fault containment at both the functional and timing levels. Such tools should also support the automatic construction of fault trees to compute the probability of a hazard occurrence or simply the causal dependencies that link it to subsystem-level or even atomic component faults based on the deployment choices.

- *Composability and extensibility versus efficiency.* The timing of software tasks depends on the presence or absence of other tasks, and a similar reasoning applies to messages. A scheduling policy that could prevent timing variability in the presence of dynamically changing task characteristics can be conceived, but it will carry at least some overhead. Further, no commercially available RTOS supports this kind of policy.

The previous situation shows the standard tradeoff between efficiency and reliability, but with more important business implications than usual. If software from different sources must be integrated on a common hardware platform—in the absence of composition rules and formal verification of the composed systems' properties—who will be responsible for the final product's correct functioning?

Whoever takes responsibility for subsystem specification and later integration will need a strong methodology and iron fist to make suppliers and partners comply with it. This may not be enough, in the sense that software characteristics are hard to pin down. Even with the best intentions, in the presence of foreign components, developers might not be able to guarantee functional and timing behavior and reliability targets.

The constant growth of embedded systems design complexity makes manual analysis and design impractical and error prone. The ideal approach would automatically map a set of tasks onto the platform, guaranteeing the correct functionality and timing with optimal resource utilization. This approach should take the design description at the pure functional level—including performance and other constraints, as well as the platform architecture—and produce correct settings for the middleware, RTOS, and optimized application-level code.

## MODEL-BASED DESIGN

Software content in vehicles has grown steadily over the years. Conceivably, by 2010 more than 100 million lines of code will be present in even low-end vehicles. Manufacturers increasingly adopt model-based design

methodologies for improving the quality and reusability of these software artifacts. A model-based environment allows development of control and dataflow applications in a graphical language familiar to control engineers and domain experts. Defining components at higher abstraction levels and with well-defined interfaces permits separation of concerns and improves modularity and reusability. Further, the use of virtual prototyping tools during development allows verification by simulation of the system behavior.

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 have several shortcomings:

**The constant growth of embedded systems design complexity makes manual analysis and design impractical and error prone.**

- *Lack of separation between the functional and architecture model.* Such a separation is fundamental for exploring different architecture options with respect to functionality and for reusing an architecture platform with different functions.
- *Lack of support for defining the task and resource model.* Most model-based design flows support the transition from the functional model directly to the code implementation. The designer has limited control when generating the task set and can barely address the task and resource model. Placement of tasks in a distributed environment is typically performed at the code level. The specification of task and message design, and of resource allocation policies, is necessary to evaluate the system's timing and dependability properties. Modeling languages often do not consider the definition of end-to-end deadlines and jitter constraints, which results in insufficient support for the specification of timing constraints and attributes.
- *Lack of modeling support for the analysis and back-annotation of scheduling-related delays.* Most tools support the functional model's simulation and verification, which developers typically base on an assumption of zero communication and computation delays. Deployment on a given architecture allows analysis of the delays caused by resource sharing. In a sound design flow, tools should support this analysis, and the communication and scheduling delays should be back-annotated into the model to verify the function's performance on a given architecture solution.
- *Lack of sufficient semantics preservation.* When generating code from a starting model description, developers do not always preserve the original semantics. Designers and developers must understand under what conditions the code-generation

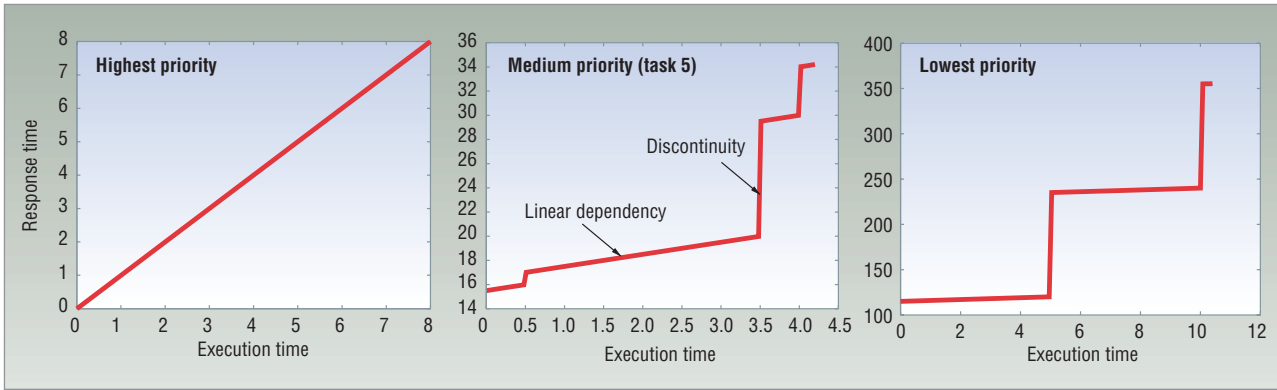


Figure 1. Worst-case response times of the highest-priority, a medium-priority (fifth task), and the lowest-priority task in the sample set of eight tasks of Table 1, when their computation times are increased from 0 to the maximum value that ensures completion within the designated period.

stage can preserve the model semantics. They must also realize the implications of an incorrect implementation.

### TIMING PREDICTABILITY AND ISOLATION

Traditionally, the automotive domain has been receptive to methods and techniques for timing predictability and time determinism. Developers based the standard controller area network (CAN) bus<sup>2</sup> on a deterministic resolution of the contention between messages and on the assignment of priorities to them. The OSEK standard for RTOSs ([www.osek-vdx.org](http://www.osek-vdx.org)) not only supports predictable priority-based scheduling,<sup>3</sup> but also bounded worst-case blocking time through an implementation of the immediate priority ceiling protocol.<sup>4</sup> OSEK also defines nonpreemptive groups<sup>5</sup> for a possible further improvement of some response times and to allow for stack space reuse. In the absence of faults, and assuming that a task's worst-case execution time can be safely estimated, these standards allow predicting the worst-case timing behavior of computations and communications.<sup>6,7</sup>

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. Further, developers design control algorithms to be tolerant of both small changes in the timing behavior and the nondeterminism in time. This can arise because of preemption and scheduling delays<sup>8</sup> or possibly because of overwritten data or skipped task and message instances caused by temporary timing faults.

Finally, although formally incorrect, there is a common perception that small changes in the timing parameters, such as decreased periods or wrong computation-time estimates, typically result only in a graceful degradation of the tasks' and messages' response

Table 1. Sample task set.

Task	$C_i$	$T_i$	$r_i$
$\tau_1$	1	8	1
$\tau_2$	3	10	4
$\tau_3$	2	12	6
$\tau_4$	3.5	20	15.5
$\tau_5$	3	30	19.5
$\tau_6$	2	60	40
$\tau_7$	7	120	115
$\tau_8$	8	300	238

times. Further, developers believe that such degradation will in any case preserve the high-priority computations.

This is only partly true, however. Figure 1 shows the worst-case response times of the highest-priority, a medium-priority, and the lowest-priority task in a sample set of eight tasks.<sup>3</sup> As Table 1 shows, these tasks have nominal computation time and period values that ensure completion within the designated period. However, when their computation times are increased, their response time depends linearly on the computation time only in limited portions of the graphs. For all tasks except the highest-priority one, there exist points of discontinuity, where the increased number of preemptions adds the entire execution of one or more task instances to the response time.

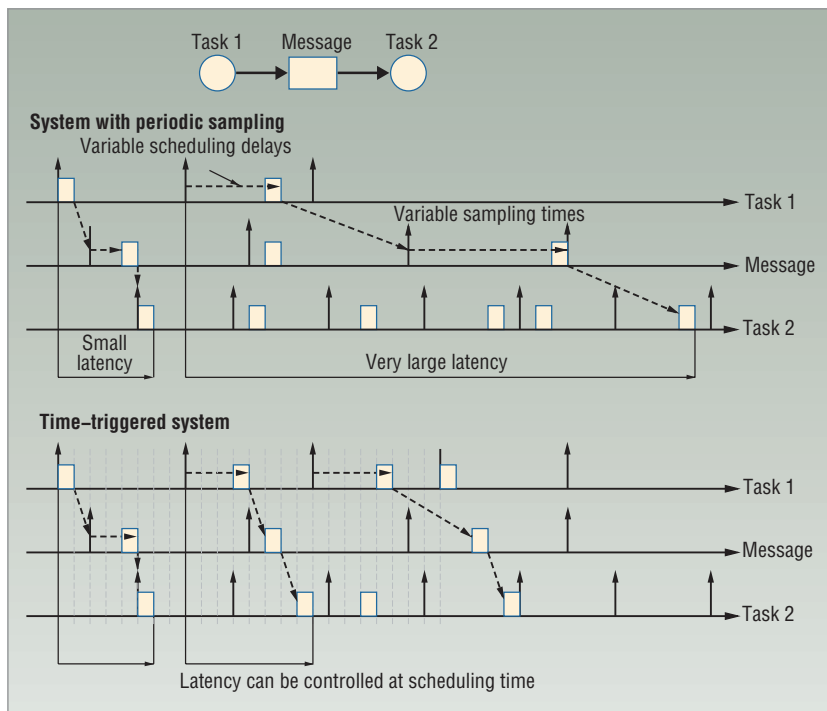
Development of larger and more complex applications—deployed with significant parallelism on each ECU, consisting of a densely connected graph of distributed computations and new safety-critical functions that require tight deadlines and the guaranteed absence of timing faults—makes previous assumptions no longer trustworthy. A new rigorous science must be established. Several issues must be considered regarding current standards and the use of priority-based task and message scheduling:



- Priority-based scheduling can lead to discontinuous behavior in time and timing anomalies. The dependency of a lower-priority task or message's response time on the computation time of a higher-priority task is nonlinear and not even continuous. A small additional high-priority load can lead to a sudden increase in the response time on some computation paths.<sup>9</sup> Further, especially in distributed systems, timing anomalies are possible, and shorter computation times may result in larger latencies.<sup>10</sup>
- Variability of the response times between worst- and best-case scenarios, together with the possible pre-emptions, can lead to violation of time-deterministic model semantics in the implementation of software models by priority scheduled tasks and messages.<sup>11</sup>
- Extensibility and, to some degree, tolerance with respect to unexpectedly large resource requirements from tasks and messages allowed by priority-based scheduling comes at the price of additional jitter, latency, and lack of timing isolation.
- Future applications, including safety-critical and active-safety ones, need shorter latencies and time determinism—*reduced jitter*—to increase performance. The current model for propagating information, based on communication by periodic sampling among nonsynchronized nodes,<sup>12</sup> has very high latency in the worst case and significant jitter

between the best- and worst-case delays. Even if communication-by-sampling can be formally studied and platform implementations defined to guarantee at least some fundamental communication-flow properties, such as data preservation,<sup>12</sup> time determinism is typically disrupted and the application must tolerate the large latencies caused by random sampling delays. Figure 2 shows how communicating information by periodic sampling and shared variables can result in large latencies and an equally large jitter between the best- and worst-case end-to-end latency. In a time-triggered system, task and message scheduling can be arranged to reduce latencies and jitter.

- Deployment of reliable systems requires timing isolation in software-component execution and protection from timing faults. Timing protection is even more important in light of AUTOSAR, which integrates components from Tier 1 suppliers into the same ECU, requiring containment and isolation of faulty functional and temporal behaviors.
- The development of future applications will also require the enforcement of composability and compositionality, not only in the functional domain but also for parafunctional system properties, including the components' timing behavior and reliability.



**Figure 2. Periodic sampling model versus a time-triggered system. Communicating information by periodic sampling and shared variables can result in large latencies and an equally large jitter between the best- and worst-case end-to-end latency. In a time-triggered system, task and message scheduling can be arranged to reduce latencies and jitter.**

Priority-based resource scheduling has the major downside of allowing faulty high-priority computation or communication flows to easily take control of the ECU or bus, subtracting time from lower-priority tasks or messages. For example, an excessive request for computation time from any high-priority task affects the response time of lower-priority tasks on the same ECU.

In this case, additional control layers—consisting of runtime guards that monitor the timing assertions—avoid the propagation of timing faults. In the future, application tasks from multiple Tier 1 suppliers will integrate into the same ECU—leveraging the standardization of interfaces allowed by AUTOSAR—and it will be necessary to protect the tasks of each IP from other IPs' timing errors. Timing isolation is therefore required to provide additional separation of concerns and protection.

Time-based schedulers, including those supported by the FlexRay and OSEKtime standards, force context switches on the ECUs and the assign-

ment of the communication bus at predefined points, regardless of the outstanding requests for computation and communication bandwidth. They are thus better suited to provide temporal protection, except that the enforcement of a strict time window for execution and communication requires that the designer have a much better ability to predict the worst-case task execution times<sup>13</sup> to allow sizing the execution window appropriately. Further, guardians must be used to ensure that an out-of-time transmission will not disrupt the bus's communication flow.

### COMMUNICATION AND DISTRIBUTED SYSTEMS

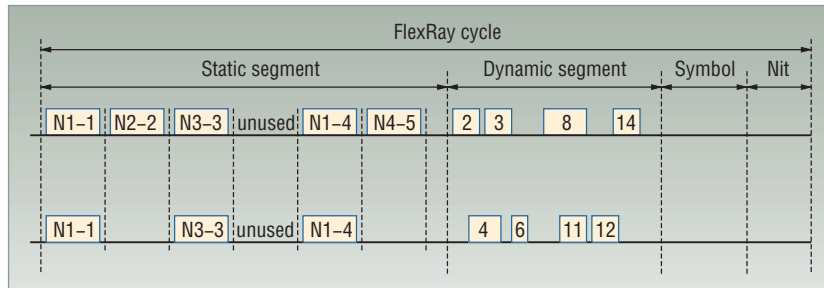
Motivations for the upcoming FlexRay communication standard for highly deterministic and high-speed communication include development of new by-wire functions with stringent requirements for determinism and short latencies, as well as innovative active safety functions. These are characterized by large volumes of data traffic, generated by 360-degree sensors positioned around the vehicles.

A consortium that includes BMW, DaimlerChrysler, General Motors, Freescale, NXP, Bosch, and Volkswagen/Audi as core members is developing the FlexRay standard ([www.flexray.com](http://www.flexray.com)). The consortium seeks to support cost-effective deployment of distributed by-wire controls.

The currently available CAN standard<sup>2</sup> is limited to a speed of 500 Kbps and imposes a protocol overhead of more than 40 percent, given that the maximum payload of each frame is 64 bits and the protocol overhead consists of at least 47 bits for the standard format. In CAN, a contention phase assigns the shared bus immediately before each message's transmission. At each contention, the message with the lowest identifier gets the right to transmit.

FlexRay defines the communication speed at 10 Mbps. The bus time is assigned according to a time-triggered pattern, with time divided into communication cycles. Each cycle contains up to four segments: static, dynamic, symbol, and nit. Clock synchronization for communication has been embedded in the standard using part of the nit segment and therefore incurs no additional cost.

Of the communication segments, the static part allows transmission of time-critical messages according to a periodic cycle in which the system always reserves a time slot of fixed length at a given position on the same node. The dynamic segment allows for flexible communication. Identifier priority arbitrates message transmission in the dynamic part, with the lowest identifier messages transmitted first, similar to CAN.



**Figure 3. FlexRay's dual-channel bus. Dual-channel configurations allow replicating messages on both channels, which facilitates safety-critical communications that leverage physical redundancy.**

FlexRay includes a dual-channel bus specification for increased reliability. Including bus guardians at the node- and star-level in the upcoming specification will in turn offer increased reliability and timing protection. In a dual-channel configuration, messages can be replicated on both channels, as Figure 3 shows for the messages from node N1. This facilitates safety-critical communications that leverage physical redundancy. The slots can also be assigned independently, in which case the system doubles the communication bandwidth.

FlexRay's time-triggered model not only allows for much better time determinism, but developers also consider it a better paradigm for composability and extensibility. Each node only needs to know the time slots for its outgoing and incoming communications. The specifications of these time slots reside in local scheduling tables. No global description exists and each node executes with respect to its own synchronized clock. As long as the local tables are kept consistent, no timing conflicts or interferences arise. Slots left free in the virtual global table resulting from the local tables' composition can be used for future extensions. Reserving time slots guarantees time protection and isolation from timing faults, while guardians avoid that node transmit outside the allocated time window.

Clock synchronization and time determinism on the communication channel allow implementation of end-to-end computations in which the data generation, data consumption, and communication processes align temporally, avoiding sampling delays. Also, system-level time-triggered schedules allow the semantics-preserving implementation of distributed control models—including models with a synchronous reactive semantics, like those that popular commercial tools such as Simulink from Mathworks ([www.mathworks.com](http://www.mathworks.com)) produce. To achieve these goals, the time-triggered communication model must be propagated to the computation layers, using a time-triggered scheduler and careful coordination of the communication and computation schedules, so that the schedule becomes global. However, although the OSEKtime standard is a suitable candidate for a time-triggered RTOS, current stan-



dards barely address synchronization of the communication and RTOS layers.

Finally, with respect to reliability, although FlexRay has a powerful error-detection mechanism, the foreseen error-management scheme instructs the receiver to discard a corrupted frame. Because the standard does not provide support for an acknowledgment mechanism (which does exist in CAN), if an application needs a reliable communication mechanism, an acknowledgment must be implemented at the application level.

However, the communication cycle's fixed structure would probably require preallocating a communication slot specifically for acknowledging each transmission. Since the system uses fixed-size static slots, this can imply a significant loss of bandwidth and, even in the best case, the transmitter must wait for its next communication cycle before attempting a retransmission.

In CAN, however, faults usually have limited consequences. All receivers discard error frames and attempt retransmission immediately, without the application's intervention. Similarly, CAN offers some limited protection against byzantine faults, although most serial data designers and users probably are not aware of this. Again, such protection must be planned for and explicitly implemented at FlexRay's application level. Taking all these factors into account, the potential 20 times speedup for FlexRay with respect to 500 Kbps CAN communication will probably be much less than expected.<sup>14</sup>

### COMPOSABILITY AND AUTOSAR

The increasing complexity of software implementations parallels increasing supply-chain complexity. Software developers design their components based on requirement definitions from the OEMs or Tier 1 suppliers, who are later responsible for their integration. The AUTOSAR development partnership,<sup>1</sup> which includes several OEM manufacturers, Tier 1 suppliers, and tool and software vendors, has been created to develop an open industry standard for automotive software architectures. To achieve the technical goals of modularity, scalability, transferability, and function reusability, AUTOSAR provides a common software infrastructure based on standardized interfaces for the different layers.

The current version of AUTOSAR includes a reference architecture and interface specifications. Also, the AUTOSAR consortium recently acknowledged that the specification lacks a formal model of components for design time verification of their properties and development of virtual platforms. As a result, the development partnership started defining the AUTOSAR metamodel.

The AUTOSAR project has focused on the concepts of location independence, interface standardization, and code portability. Although these goals are extremely important, their achievement will not necessarily be sufficient for improving the software systems' quality. As with most other embedded systems, car electronics are characterized by functional and nonfunctional properties, assumptions, and constraints. In complex systems, component-based design can provide encapsulation and separation of concerns, thereby improving reuse if information hiding is implemented so that the component model allows the following properties<sup>6</sup>:

- *composability*, which guarantees preservation of a component property across integration; and
- *compositionality*, which allows deduction of the composed object's global properties from its component properties; this property enables correctness-by-construction.

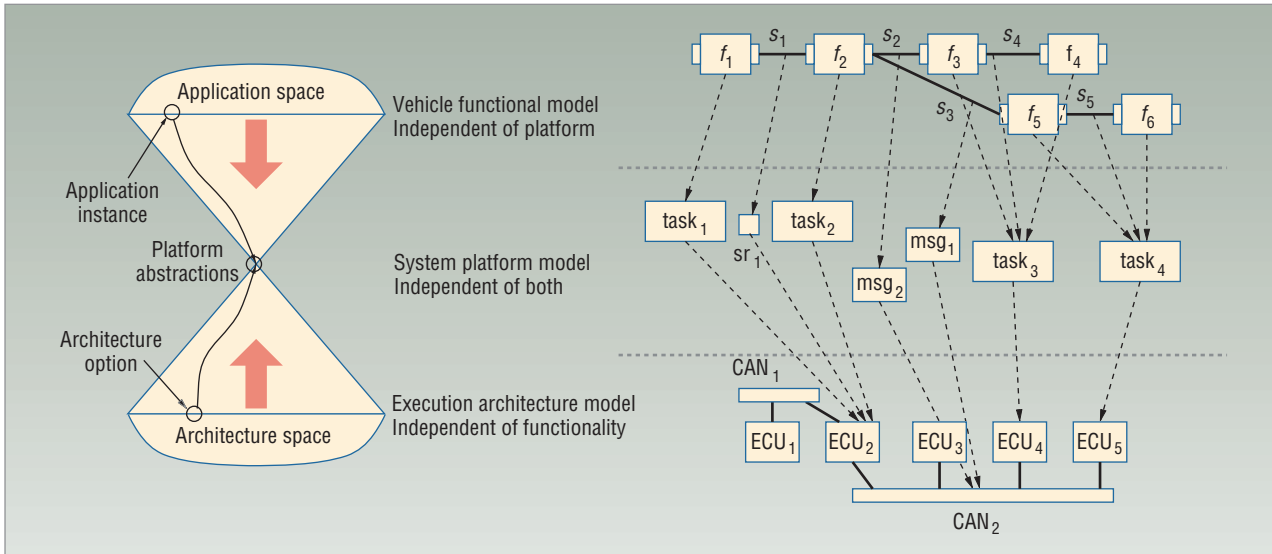
**The AUTOSAR development partnership has been created to develop an open industry standard for automotive software architectures.**

The current specification has at least two major shortcomings that prevent achieving the desired goals. The AUTOSAR metamodel, as of now, lacks a clear and unambiguous communication and synchronization semantics and a timing model. Similar to UML—not surprisingly, considering that UML 2.1 inspired the specification, which by its very

description is provided in the form of UML diagrams—the AUTOSAR metamodel is sufficiently mature in its static or structural part, but offers an incomplete behavioral description. Developers plan to remedy this with significant updates in the upcoming AUTOSAR revision, however.

Further, none of the standard's several layers address issues related to timing and performance, which thus underestimates the complexity of current and future applications. These applications' component interactions generate a variety of timing dependencies due to scheduling, communication, synchronization, arbitration, blocking, and buffering. Ensuring component reuse is not simply a matter of compile-time guarantees that the provided and required functions be accessible regardless of the software module's location and that formal parameters and actual parameters match. It also demands that the behavior of the reused components can be predicted in the new configuration and the result of the composition can be analyzed for timing faults. If developers fail to address this problem, the composition will eventually lead to possibly transient timing problems, including missed deadlines, task and message skipping, or overwriting and buffer overflows.

The definition of a timing model for AUTOSAR, and the development of a standardized infrastructure for the



**Figure 4. Platform-based design and models.** The vertex of the two cones represents the combination of the functional model and architecture platform. Decoupling the application-layer logic from dependencies on infrastructure-layer hardware or software allows reusing application-layer components across multiple vehicle programs without changes.

handling of time specifications, is the objective of the European Union ITEA 2 (Information Technology for European Advancement) project TIMMO (timing model). Started in April 2007, the project includes car manufacturers like PSA, Volvo Technology, and Volkswagen/Audi. The project targets networked automotive real-time systems with the goal of providing

- a description language for time aspects in the development of automobile control units and networks,
- a methodology for cross-company usage of this description language, and
- a validation of the language by means of prototypical demonstrators.

AUTOSAR also encompasses electronics and tool suppliers, including Bosch, Continental, ETAS, Siemens VDO, Syntavision, and TTTech.

### PLATFORM-BASED DESIGN FOR ARCHITECTURE SELECTION

Platform-based design<sup>15-17</sup> requires and elicits clearly identified abstraction layers and a design interface that allows for separation of concerns between refinement of the functional architecture specification and the abstractions of possible implementations. This approach decouples application-layer software components from changes in microcontroller hardware, ECU hardware, I/O devices, sensors, actuators, and communication links. The left side of Figure 4 demonstrates the basic idea. 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 soft-

ware allows reusing the application-layer components across multiple vehicle programs without changes.

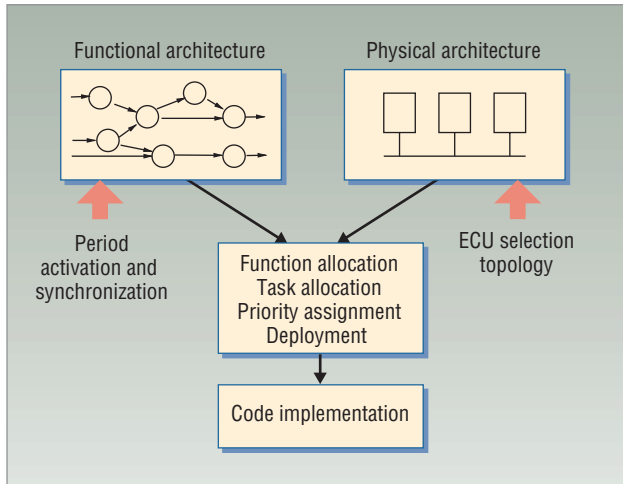
A prerequisite for adopting the platform-based design and meet-in-the middle approach is the definition of the right models and abstractions for the functional platform specification's description and the architecture solutions at the top and bottom of the hourglass in Figure 4. The platform interface 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 implementation's properties.

This model can include size, reliability, power consumption, and timing—variables associated with lower-level abstraction. Design space exploration consists of seeking the system platform model's optimal mapping into the candidate execution platform instances. This mapping must be driven by a set of methods and tools that provide a measure of the architecture solutions' fitness for providing a set of feasibility constraints and optimization-metric functions. This work focuses on timing constraints and metrics. In the currently available approach, a what-if analysis, developers select different options as representatives of the principal architecture solutions and evaluate them metrically.

### What-if analysis

A what-if iterative process drives architecture selection and evaluation. First, developers define the set of metrics and constraints that apply to the design. Then, based on the designer's experience, the team produces a set of initial candidate architecture configurations. The team evaluates these configurations and, based on the results of the quantitative analysis, extracts a solution from the set as the best fit. If the designer is dissatisfied





**Figure 5. Design flow stages and period synthesis. This chart's application and architectural descriptions are joined in an explicit mapping step. Both the mapping definition and the task- and resource-model creation can be performed in several ways.**

with the result, the team can select a new set of candidate architectures. The iterative process continues until a solution is obtained.

The designer must intervene in two tightly related stages of the exploration cycle. First, the initial set of architecture options must be provided. Second, once the analysis and simulation tools have scored and annotated the options, the designer must assess the analysis's results and select the architecture options that best fit the exploration goals. More importantly, the designer must evaluate the analysis results to add other options to the next set of architecture configurations for evaluation. The currently available set of analysis methods, including those not related to timing constraints and metrics, are as follows:

- evaluation of end-to-end latency and schedulability against deadlines for computation chains spanning tasks and messages scheduled with fixed priority;<sup>6,7</sup>
- sensitivity analysis for tasks and messages scheduled with fixed priorities, and sensitivity analysis for resources scheduled with fixed priorities;<sup>9</sup>
- evaluation of message latencies in CAN bus networks;<sup>4</sup>
- system-level simulation of time properties and functional behaviors, based on the Metropolis engine;<sup>18</sup>
- analysis of fault probability and cutsets (conditions leading to critical faults) based on fault trees;<sup>19</sup> and
- product-line cost analysis.

Going beyond the purely designer-driven what-if approach, our current research includes algorithms and methods for the automatic configuration of the software architecture or at least some of its attributes.

## Automatic SW architecture configuration

Mapping the functional model into the execution platform is part of the platform-based design and Y-chart design flow.<sup>20,21</sup> As Figure 5 shows, this chart's application and architectural descriptions are joined in an explicit mapping step. The mapping definition and creation of the task and resource models can be performed in several ways. In single-processor systems, the problem is usually simple and often subsumed by the code-generation phase. In distributed architectures, designing the software architecture presents a more complex task that often must be delegated to the most experienced designer. When resource constraints make a software implementation infeasible, design iterations might be triggered, and the functional model itself, or the architecture configuration, might be modified.

Once developers define the function and architecture, they have several options for the intermediate layer. At least conceptually, automated tools could provide guidance in selecting the optimal configuration's timing constraints and performance-related metric function. The mapping consists of the following stages: function-to-task mapping, task-to-ECU deployment, signal-to-message mapping, and assignment of priorities to tasks and messages. When the functional model requires iterations, a different selection of the functions' execution periods or synchronization and communication solutions could be explored.

In past research, we defined solutions based on mixed-integer linear programming and geometric programming, respectively, to optimize the tasks and messages activation mode<sup>22</sup> and the selection of task periods.<sup>23</sup> We demonstrated the effectiveness of these approaches by applying them to an experimental vehicle system case. We are currently exploring approximated solutions for the selection of a feasible mapping of tasks to ECUs, signals to messages, and assignment of priorities to tasks and messages.

Using this methodology, we envision the availability of an intermediate platform layer in which developers map the functions into the architecture option and evaluate the result with respect to parafunctional metrics and constraints related to timing, dependability, and cost. This will be of the highest importance in supporting the evolution of automotive standards and ensuring the feasibility of a correct and robust design flow based on a virtual platform. ■

## References

1. AUTOSAR Consortium Web page; [www.autosar.org](http://www.autosar.org).
2. R. Bosch, *CAN Specification*, v2.0, Stuttgart, 1991.
3. M.G. Harbour, M. Klein, and J. Lehoczky, "Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems," *IEEE Trans. Software Eng.*, vol. 20, no. 1, 1994, pp. 13-28.



4. L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Computers*, vol. 39, no. 9, 1990, pp. 1175-1185.
5. Y. Wang and M. Saksena, "Scheduling Fixed-Priority Tasks with Preemption Threshold," *Proc. IEEE Int'l Conf. Real-Time Computing Systems and Applications*, IEEE Press, 1999, pp. 328-337.
6. M. Joseph and P.K. Pandya, "Finding Response Times in a Real-Time System," *The Computer J.*, vol. 29, no. 5, 1986, pp. 390-395.
7. K. Tindell, A. Burns, and A.J. Wellings, "Calculating Controller Area Network (CAN) Message Response Times," *Control Eng. Practice*, vol. 3, no. 8, 1995, pp. 1163-1169.
8. P. Caspi and A. Benveniste, "Toward an Approximation Theory for Computerized Control," *Proc. 2nd Int'l Conf. Embedded Software (EMSOFT 02)*, Springer-Verlag, 2002, pp. 294-304.
9. E. Bini, M.D. Natale, and G. Buttazzo, "Sensitivity Analysis for Fixed-Priority Real-Time Systems," *Proc. Euromicro Conf. Real-Time Systems*, IEEE Press, 2006, pp. 10-20.
10. R. Racu and R. Ernst, "Scheduling Anomaly Detection and Optimization for Distributed Systems with Preemptive Task-Sets," *Proc. 12th IEEE Real-Time and Embedded Technology and Applications Symp.*, IEEE Press, 2006, pp. 325-334.
11. M. Baleani et al., "Efficient Embedded Software Design with Synchronous Models," *Proc. 5th ACM Int'l Conf. Embedded Software (EMSOFT 05)*, ACM Press, 2005, pp. 187-190.
12. A. Benveniste et al., "The Synchronous Languages 12 Years Later," *Proc. IEEE*, vol. 91, Jan. 2003, pp. 64-83.
13. C. Ferdinand et al., "Reliable and Precise WCET Determination for a Real-Life Processor," *Proc. 1st Int'l Conf. Embedded Software (EMSOFT 01)*, Springer-Verlag, 2001, pp. 469-485.
14. T.M. Forest, "The FlexRay Communication Protocol and Some Implications for Future Application," *Proc. Convergence*, SAE Press, 2006.
15. A.L. Sangiovanni-Vincentelli, "Quo Vadis, SLD? Reasoning about Trends and Challenges of System Level Design," *Proc. IEEE*, vol. 95, no. 3, pp. 467-506.
16. A.L. Sangiovanni-Vincentelli, "Defining Platform-Based Design," *EETimes*, Feb. 2002.
17. A.L. Sangiovanni-Vincentelli et al., "Benefits and Challenges for Platform-Based Design," *Proc. Design Automation Conf.*, 2004, ACM Press, pp. 409-414.
18. F. Balarin et al., "Processes, Interfaces and Platforms: Embedded Software Modeling in Metropolis," *Proc. 2nd Int'l Conf. Embedded Software*, Springer-Verlag, 2002, pp. 407-416.
19. C. Pinello, L. Carloni, and A. Sangiovanni-Vincentelli, "Fault-Tolerant Deployment of Embedded Software for Cost-Sensitive Real-Time Feedback-Control Applications," *Proc. DATE Conf.*, IEEE CS Press, 2004, pp. 1164-1169.
20. F. Balarin et al., *Hardware-Software Co-Design of Embedded Systems—The Polis Approach*, Kluwer Academic Publishers, 1997.
21. B. Kienhuis et al., "A Methodology to Design Programmable Embedded Systems: The Y-Chart Approach," *Embedded*

- Processor Design Challenges: Systems, Architectures, Modeling, and Simulation—SAMOS, LNCS 2268, E.F. Deprettere, J. Teich, and S. Vassiliadis, eds., Springer, 2002, pp. 18-37.
22. W. Zheng et al., "Synthesis of Task and Message Activation Models in Real-Time Distributed Automotive Systems," *Proc. IEEE/ACM DATE Conf.*, IEEE Press, 2007, pp. 93-98.
23. A. Davare et al., "Period Optimization for Hard Real-Time Distributed Automotive Systems," *Proc. ACM Design Automation Conf.*, ACM Press, 2007, pp. 278-283.

*Alberto Sangiovanni-Vincentelli holds the Edgar L. and Harold H. Buttner Chair of EECS at the University of California, Berkeley. His research interests include system design, methodologies and tools, hybrid system control, and innovation. Sangiovanni-Vincentelli received a Dr. Ing. in electrical engineering and computer sciences from Politecnico di Milano. Contact him at [alberto@eecs.berkeley.edu](mailto:alberto@eecs.berkeley.edu).*

*Marco Di Natale is currently an associate professor at the Scuola Superiore S. Anna of Pisa, Italy. This work was performed while he was a staff researcher at General Motors Research and Development. His research interests include the modeling, design, and timing evaluation of embedded systems and the evaluation of distributed architectures. Di Natale received a PhD in computer engineering from the Scuola Superiore S. Anna of Pisa. Contact him at [marco@sssup.it](mailto:marco@sssup.it).*



## REACH HIGHER

**Advancing in the IEEE Computer Society can elevate your standing in the profession.**

**Application to Senior-grade membership recognizes**

- ✓ **ten years or more of professional expertise**

**Nomination to Fellow-grade membership recognizes**

- ✓ **exemplary accomplishments in computer engineering**

**GIVE YOUR CAREER A BOOST ■ UPGRADE YOUR MEMBERSHIP**

**[www.computer.org/join/grades.htm](http://www.computer.org/join/grades.htm)**