Mixed Models of Computation in the Design of Automotive Engine Control[1]

Andrea Balluchi† Maria Domenica Di Benedetto‡
Claudio Pinello¶ Alberto Luigi Sangiovanni–Vincentelli†¶

†PARADES, Via San Pantaleo, 66, 00186 Roma, Italy,
balluchi, alberto@parades.rm.cnr.it

‡Dip. di Ingegneria Elettrica, Università dell'Aquila,
Poggio di Roio, 67040 L'Aquila, Italy, dibenede@ing.univaq.it

¶Dep. of Electrical Engineering and Computer Science,
University of California at Berkeley, CA 94720, alberto,pinello@eecs.berkeley.edu

Abstract

In this paper we discuss issues involved in system level design of complex embedded systems, we focus on automotive engine and power–train control applications. We briefly illustrate the problem of capturing a complicated plant and of designing control laws that satisfy multiple requirements in different regions of operation. We propose a unified framework, the Tagged Signal Model (TSM) based on an extended theory of hybrid systems. Our framework can put together engine models that are a combination of finite state machines, discrete event and sequential processes, powertrain models, sensors and actuators, and at last controller models. The synchronous reactive language formalism can be used for the design of the controller so that synthesis procedures can be followed to generate either the hardware or the software.

1 Introduction

System level design is rapidly becoming the focus of attention in electronics design. Time-to-market, safety, power consumption, cost and size are criteria that guide the design problem. In these last few years, we have witnessed an unprecedented increase in complexity that has caused traditional design methodologies based on extensive prototyping almost obsolete. Hence formal approaches are more and more in demand. The essence of a formal approach to system level design is the ability to use mathematical models to capture the requirements and the functions of a design. Design capture is in general performed via a language that expresses in a more or less formal way the functionality of our system. In the past, there has been a number of different languages used for different parts of the system: hardware description languages for the hardware component, C or other low level languages for the software and for the high level functional models of the hardware. Only recently the attention has shifted towards languages with strong semantic properties, i.e., that are based on precise mathematical underlying concepts. In example, synchronous reactive languages, like Esterel and Lustre, have made significant contributions in this direction. However, when we consider the complexity and the richness of the systems we are trying to design, it is clear that these approaches are somewhat limited and that we need an innovative design methodology that might be seen as unnatural for a number of system designers today. We believe much needs to be done in the area of heterogeneous system capture and design. Some approaches have been quite successful in the research domain (e.g., Ptolemy for simulation and capture, Polis for synthesis and estimation) and have started having industrial impact (e.g., Cadence VCC). Still several questions remain unanswered. In particular, how far is it reasonable to go with a fully formal approach? What mathematical models does it make sense to use? Is it possible to capture in a unified framework both the system under design and the environment it is asked to function in?

To illustrate the difficulties that a designer has to face to take a rigorous approach to system design, in this paper we focus on an application domain that is of great industrial interest: automotive engine and power–train control. The problem here is to capture a complicated plant (the entire power train including engine, transmission, sensors and actuators) and to design control laws that satisfy multiple requirements in different regions of operation. In reference [2], we presented a general framework for power–train control based on hybrid models and we illustrated its effectiveness in addressing the synthesis of some control problems. To formally handle the interactions among the parts of different nature in the closed–loop system, we use a modeling framework called the Tagged Signal Model (TSM) [6]

[1]This research has been partially sponsored by PARADES, a Cadence, Magneti-Marelli and SGS-Thomson GEIE, by CNR and by GSRC.

that addresses explicitly time and partial orders among events, and is particularly suited to represent modality of interaction. The capability of reasoning about such a wide variety of mathematical models connected together allows us to take for the first time a holistic view to the problem of embedded controller design.

In this paper, we show how the TSM modeling framework can also be successfully used to represent the functionalities and the behavior of sensors and actuators, as well as to describe the specifications for the implementation of the designed control laws. An effective way to design the controller is by using a synchronous reactive language. This type of description is amenable to synthesis procedures that can flexibly generate either the hardware or the software implementing the controller. Given the strong semantic properties of these languages, formal verification techniques can be used to guarantee adherence to the specifications.

The paper is organized as follows. In Section 2, we introduce the Tagged Signal Model framework. In Section 3, a TSM hybrid model of the engine and of the power–train is described. In Section 4, the design process of the control system is discussed with particular emphasis on the representation of the implementation specifications.

2 Background on models of computation

We use the theory behind the notion of models of computation to identify the role of the interfaces among the different components of closed–loop systems described as hybrid systems. In particular, we adopted the tagged-signal model (TSM) formalism proposed by Lee and Sangiovanni-Vincentelli [6]. We recall here only some of the most important aspects of this formalism. The reader is referred to [6] for a detailed description. The TSM formalism defines a semantic framework within which models of computation can be studied and compared. It is abstract—describing a particular model of computation involves imposing further constraints that make it more concrete. The fundamental entity in the TSM is an event: a value/tag pair (v, t). Tags are often used to denote temporal behavior. A set of events is a signal. Processes are relations on signals, expressed as sets of tuples of signals. A particular model of computation is distinguished by the order it imposes on tags and the character of processes in the model.

2.1 Concurrency and Communication

In general, systems will typically contain several coordinated concurrent processes. At the very least, such processes interact with an environment that evolves independently, at its own speed. It is also common to partition the overall model into tasks that also evolve more or less independently, occasionally (or frequently) interacting with one another. This interaction implies a need for coordinated communication. Communication between processes can be *explicit* or *implicit*. Explicit communication implies forcing an order on the events, and this is typically realized by designating a *sender* and a *receiver* process. Implicit communication implies the sharing of a common time scale and a common notion of state.

Basic time. In classical transformational systems, such as personal computers, the correct result is the primary concern—when it arrives is less important (although *whether* it arrives *is* important). By contrast, embedded systems are usually real-time systems, where the time at which a computation takes place is very important. Different models of time become different order relations on the set of tags T in the tagged-signal model. Implicit communication generally requires totally ordered tags (*timed processes*), usually identified with physical time. The tags in a *metric timed process* have the notion of a "distance" between them, much like physical time. Two events are *synchronous* if they have the same tag (the distance between them is 0). Two signals are synchronous if each event in one signal is synchronous with an event in the other signal and vice versa.

Treatment of time in processes. A *synchronous process* is one in which every signal in the process is synchronous with every other signal in the process. An *asynchronous process* is a process in which no two events can have the same tag. Hence, in our framework, asynchronous processes are a subset of non synchronous processes. If tags are totally ordered, the process is *asynchronous interleaved*, while if tags are partially ordered, the process is *asynchronous concurrent*. In an asynchronous concurrent process events may not have a precedence relation, so they may represent or be the result of independent activities that might be executed concurrently[2]. Synchronous/reactive languages (see e.g. [5]) deserve special mention. They have an underlying synchronous model in which the set of tags in any behavior of the system denotes a global "clock" for the system. However, to make this MOC synchronous in the sense of the TSM, we need to assume that every signal conceptually has an event at every tag (this can be always obtained by introducing a value denoting the absence of an event, when necessary). At each clock tick, each process maps input values to output values.

Implementation of concurrency and communication. Concurrency in physical implementations of processes implies a combination of *parallelism*, which employs physically distinct computational resources, and *interleaving*, which means sharing of a common physical resource. Parallel physical systems naturally

[2]For asynchronous processes concurrency and interleaving are, to a large extent, interchangeable, since interleaving can be obtained from concurrency by partial order embedding, and concurrency can be reconstructed from interleaving by identifying "untimed causality".

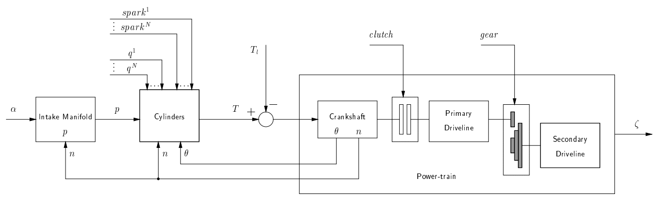

Figure 1: Power-train decomposition.

share a common notion of time, according to the laws of physics. Physically interleaved systems also share a natural common notion of time: one event happens before another. A variety of mechanisms for managing the order of events, and hence for communicating information between processes, exists. In the TSM *communication* is embodied in the event, which is a two-component entity whose value is related to function and whose tag is related to time. That is, communication between two processes is implemented by two operations:
1. the transfer of values (function; TSM event value),
2. the determination of the relationship in time (time; TSM event tag).

3 Engine and Power–train Hybrid Model

In this section, we recall a hybrid model for a power–train with a N–cylinder 4–stroke engine that has been proposed in reference [2]. The overall system is described with the TSM formalism and is composed of three main interacting blocks, namely the *intake manifold*, the *cylinders* and the *power-train* (Fig. 1).
The manifold pressure p is controlled by the throttle valve, whose position is denoted by α. The mass of air loaded in the cylinders depends on the pressure p and on the crankshaft revolution speed n. The torque T produced by the engine is given by $\Sigma_{i=1}^{N} T^i$, where T^i is the torque generated by the i–th cylinder, which is determined by the mass of loaded air m^i, the mass of fuel q^i injected in the cylinder, and the $spark^i$ ignition command[3]. The timing sequence of the four strokes of each cylinder is determined by the continuous motion of the crankshaft denoted by θ. The discrete signal $spark^i$ models spark ignition, while the continuous-valued signal q^i represents the amount of fuel injected into the cylinder. Finally, the power-train dynamics and the crankshaft revolution speed n, controlled by the generated torque T, are subject to the sum of a number of load torques, T_l, and depend on the position of the *clutch* and the selection of the *gear*.

[3]From this point on, we use the superscript i to indicate variables related to the i–th cylinder.

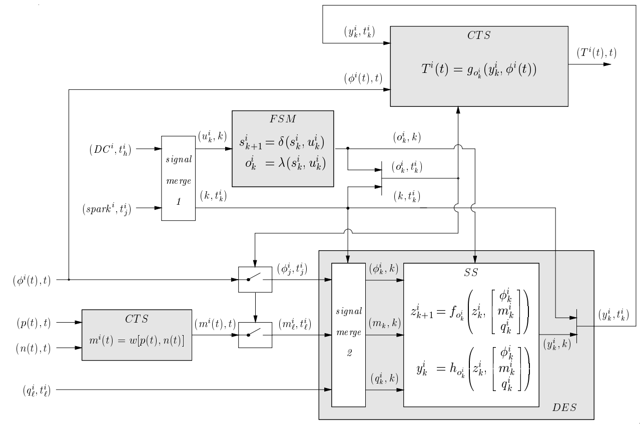

Figure 2: TSM hybrid model of the i–th cylinder.

The intake manifold. The evolution of the manifold pressure p is represented by the nonlinear dynamics $\dot{p}(t) = a_p[n(t), p(t)]\, n(t)p(t) + b_p[p(t)]\, s[\alpha(t)]$.
The i-th cylinder. The torque T^i generated by each piston at each cycle depends on:
- the phase of the engine cycle the cylinder is in;
- the piston position $\phi^i(t)$, which can be expressed in terms of the crankshaft angle $\theta(t)$;
- the mass m^i of air loaded into the cylinder;
- the amount of fuel q^i injected;
- the spark ignition timing (t^i_j) which is usually expressed in terms of the spark advance (φ^i).

The model of a single cylinder consists of four communicating processes of different MOCs (see Fig. 2):
- An FSM, modeling the 4-stroke engine cycle and spark ignition. The FSM is depicted in Fig. 3 and has input, state and output[4] u_k, s_k and o_k.
- A memory-less CTS, $m^i(t) = w[p(t), n(t)]$, modeling the air intake process.
- A DES obtained by combining a SS synchronized with the FSM transitions and two processes to keep track of the times at which transitions occur. The DES is modeling the discrete delay between the time at which the mixture is loaded, the spark is given and the time at which the corresponding torque is generated: $z^i_{k+1} = f_{o^i_k}(z^i_k, v^i_k)$, $\quad y^i_k = h_{o^i_k}(z^i_k, v^i_k)$ where the input v^i_k collects the variables (m^i, q^i, ϕ^i), and the output y^i_k provides the delayed values (m^i, q^i, φ^i).
- A memory-less CTS, $T^i(t) = g_{o^i_k}[y^i_k, \phi^i(t)]$, modeling the profile of the generated torque, which depends on the piston position $\phi^i(t)$, the FSM's output o^i_k and the DES output $(m^i, q^i, \varphi^i) = y^i_k$.

Fig. 2 shows the (non trivial) interactions among the MOCs modeling the different phenomena in the cylinder behavior. Using the TSM framework, we can un-

[4]For simplicity, we dropped superscript i from index k.

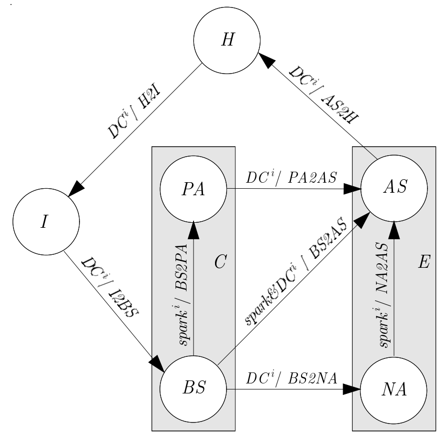

Figure 3: FSM describing the i–th cylinder's behavior.

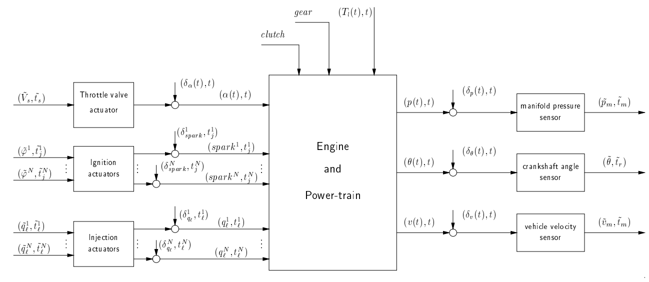

Figure 4: Representation of the plant equipped with sensors and actuators in the TSM formalism.

derline the importance of sequencing and timing of events in the torque generation process and, hence, be more effective in the synthesis of control algorithms. Note that the FSM uses only the information about the sequencing of events, not their exact timing. However, torque generation does need time information since it feeds a continuous time system (the model of the power–train).

The engine and power–train model. The overall hybrid model for vehicles with 4–stroke N–cylinder gasoline engine is obtained by combining N cylinder hybrid models and it is composed of the following parts:
• N subsystems as in Fig. 2;
• two CTS's modeling, respectively, the power-train and intake manifold dynamics;
• a process that synchronizes the cylinders models by generating the piston position angles ϕ^i from the crankshaft angle θ and the events DC^is corresponding to dead centers.

4 Control System Design

As it is common to many control applications (see e.g. [3]), also in the engine control problem a number of regions of operation or modes are identified. Each region of operation is characterized by an optimization problem that includes a set of constraints related to driving performance or emissions, and a cost function that identifies the desired behavior of the controlled system. The transitions are determined by the action of the driver or by engine conditions. A set of basic control laws is derived to address each sub-problem. The resulting global control law selects the set of control actions depending on the current region of operation.
The goal of control system design is to accomplish the system specifications providing a solution at the minimum cost. The design process of the control system is carried out by addressing the following issues: choice of sensors and actuators, synthesis of the control algorithms, implementation of the control system. In all these design steps, the TSM formalism can be used to obtain a unified representation of the entire closed–loop system, so to provide the designer with: a clear representation of the specifications for the part under design; a formalism for describing the designed part; a complete description of both the plant and the already designed parts; a framework for analyzing the transitions from one region to the next. In [2] the development of hybrid engine control algorithms has been investigated. We discuss here the issues related to the choice of the controller-plant interfacing elements and the implementation of the controller.

4.1 Sensors and actuators

The choice of sensors and actuators is in the sphere of competence of the control engineer and is part of the design of the control system. Nonetheless the stringent cost constraints typical of automotive applications often lead the car manufacturers to fix the class and often the type of sensors/actuators to be used. However, this reduces the degrees of freedom of the control engineer in the exploration of the possible solutions and may lead to a no–minimum cost design of the control system.

As far as the choice of a sensor is considered, the first decision to be taken is whether the use of a sensor to make available a system variable required for feedback is cheaper than using an algorithm to reconstruct this information. Indeed, it is often possible to use either estimators or dynamic observers that, based on other measurements available, can produce the evolution of the needed variable. On the other hand, quantities available to measurement through sensors are typically processed to filter-out noise and/or compensate for non–linearities in the sensors. Hence, the cost of this algorithm should also be considered along with the cost of the sensor itself. It should be noticed that the management of sensors cost is not straightforward. Indeed, it is often the case that the measurement from a same sensor is used by several control loops, so that the cost of the sensor has to be considered coupled with the cost of different control algorithms that have been designed for achieving the system specifications.

Actuators control some energy transfers to and from the system and can thus modify the evolution of the state. The cost of actuators is strongly related to the power they have to manage. Simple actuators can be operated by writing digital data in some register whenever a new value is desired. The controller could achieve its goals by setting these values at appropriate times. Often it is more convenient to use a decentralized control scheme where multiple local control loops are fed with higher-level commands. Consider in example the actuation of the throttle valve. The actuator is usually a DC motor and the controller can set the voltage supply across it. However it is usually simpler to develop engine control algorithms that abstract away the motor and throttle dynamics and simply assign the desired throttle position. One possible configuration would then be a local loop to track the throttle position. The corresponding controller/actuator/sensor group would then act as a logical actuation block for the higher-level controller, to which an overall cost is assigned.

In Fig. 4, an example of a representation in the TSM formalism of sensors and actuators connected to the plant hybrid model described in Section 3 is given. The blocks representing the actuators and the sensors would contain a description of the dynamics of the devices including filtering effects, non–linearities, and delays. The effects of disturbances on the plant input and output variables are explicitly represented on the domain of the physical signals. It is worth noting that sensors and actuators may have to provide an interface between signals expressed in different models of computation. The TSM formalism is particularly fit to rigorously specifying the interactions among models of computation, hence easing the task of sensors and actuators modeling. Consider the plant outputs represented in Fig. 4. The manifold pressure $(p(t), t)$ is a continuous–time variable assuming values on a continuous range, while the output $(\tilde{p}_m, \tilde{t}_m)$ available from the manifold pressure sensor is a quantized variable read at fixed–frequency sample times $\tilde{t}_m$. Furthermore, to measure the crankshaft angle $(\theta(t), t)$ — a continuous–time variable taking value on the reals — an incremental encoder connected to the flywheel is commonly used. The sensor output $(\tilde{\theta}, \tilde{t}_r)$ is then a two–value piecewise signal. Such signal needs to be processed to reconstruct an estimation of the actual crankshaft position signal $(\theta(t), t)$. The algorithm that handles the crankshaft angle sensor has first to recognize the zero value of the crankshaft position, by catching a missing tooth in the flywheel encoder, and then has to integrate the impulses $(\tilde{\theta}, \tilde{t}_r)$ and filter the obtained signal. Since all the actions related to the engine cycle, such as fuel injection and spark ignition, require a precise measure of the crankshaft angular position the reconstruction

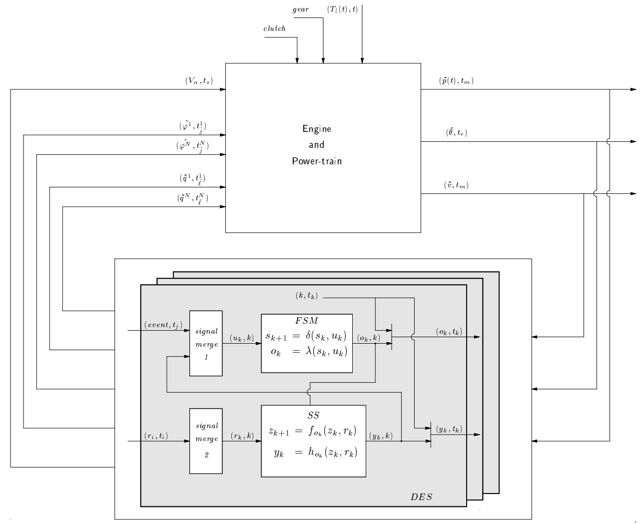

Figure 5: Representation of the closed–loop system.

of this signal is very critical for the performances of the closed–loop system. The formal description of the crankshaft angle sensor and its output signal elaborations given in the TSM formalism is instrumental to the correct design and verification of this important element of the control system. A similar analysis regards the plant inputs. The throttle valve actuator is driven by a discrete–time quantized signal $(\tilde{V}_s, \tilde{t}_s)$ produced at sample times $\tilde{t}_s$. The actuator output is the throttle position $(\alpha(t), t)$, a continuous–time signal. The model of the actuator contains the motor dynamics. The reference signals for the spark ignition actuators are the spark advance angles $(\tilde{\varphi}^i, \tilde{t}^i_j)$, generated at each engine cycle. The actuators produce the spark signals modeled by events $(spark^i, t^i_j)$.

In conclusion, the TSM description allows us to specify in a formal way the perimeter of the digital controller, assigning precise requirements to the interfacing blocks.

4.2 Control system implementation

The controller is modeled as a combination of control flow (typically a FSM) that reacts to events coming from the plant and the driver, and data flow (e.g. a DES) that, based on the values from the acquisition blocks, computes the desired values for the actuation blocks.

Regions of operation. The controller must recognize the transitions from one region to the next in order to activate the proper set of control algorithms.The corresponding FSM is extended from to the one used in the system behavior specification

Operating point. Within each region, some of the control actions and models may depend on the operating point of the engine and/or vehicle (e.g. engaged gear). Other control actions may be triggered or enabled by thresholds on engine temperature or revolution speed.

Synchronization with the crankshaft. In engine control applications most actions are synchronized with the crankshaft position. A FSM reacting to the dead-center events can easily capture this.

Synchronization with acquisition and actuation blocks. The acquisition and actuation blocks may be performing autonomous activities. In example there may be observers and local control loops. Other acquisition and actuation blocks may be triggered by a command issued by some higher-level controller (e.g. spark ignition actuation block).

The correct sequencing of actions is performed by the set of interacting FSMs each governing some of the controller blocks and exchanging signals with the others and with the supervising FSMs for the specific region of operation. Some of the control activities are common to multiple regions. However most of the transitions between regions identify a re-initialization of some of the algorithms even when the same algorithms are used in the two regions.

The resulting model of the FSM–DES interaction is depicted in Fig. 5. Like in Fig. 2 additional processes are used for keeping track of physical time and to obtain synchronous signals[5].

While the hybrid model of the plant is modeling an existing system, the one in the controller needs to be designed and implemented. The data flow part is typically well implemented in C. Synchronous languages like Esterel can be conveniently used to implement the FSMs. The potentially high number of FSMs can then be composed synchronously into a single FSM by the Esterel compiler. Since the component FSMs are not running as separate tasks, the behavior of the product FSM is largely independent of implementation details like the scheduling policy. Alternatively the FSMs can be composed asynchronously by having them implemented as separate tasks communicating via the real time operating system. This second mechanism may be less deterministic but may be used to reduce problems related to state explosion. The type of composition may also be dictated by considerations on the execution time of the dataflow part. If this time cannot be neglected with respect to the interval between transitions in other FSMS, the synchronous hypothesis cannot be satisfied for the product FSM. This kind of analysis has similarities with the one performed in POLIS [1] when some CFSMs (co-design FSMs) can be combined to a single one (synchronous composition) and others communicate asynchronously. This model of composition is called GALS (globally asynchronous locally synchronous).

After the composition, the corresponding product FSMs select the proper control actions as the region of operation and the inputs from the driver change including any initialization action necessary to run a new set of algorithms. Also in [3, 7, 4] controllers are modeled as a combination of transformational blocks (typically C functions) and FSMs coordinating them.

[5]See [2] for more details on the FSM-DES interaction.

5 Conclusions

In this paper we illustrate the use of the Tagged Signal Model (TSM) in the system level design of automotive engine controllers. In [2] the TSM was used to build accurate models of complicated plants and to capture the specifications. We illustrate here how, operating on signals of different nature, the TSM is particularly effective in modeling sensors and actuators. Moreover we propose its use to model the controller as a coordinated collection of control flow and data flow. Finally the closed loop system can be modeled to analyze global properties across regions of operation.

References

[1] F. Balarin, E. Sentovich, M. Chiodo, P. Giusto, H. Hsieh, B. Tabbara, A. Jurecska, L. Lavagno, C. Passerone, K. Suzuki, and A. Sangiovanni-Vincentelli. *Hardware-Software Co-design of Embedded Systems – The POLIS approach.* Kluwer Academic Publishers, 1997.

[2] A. Balluchi, L. Benvenuti, M. D. Di Benedetto, C. Pinello, and A. L. Sangiovanni-Vincentelli. Automotive engine control and hybrid systems: Challenges and opportunities. *Proceedings of the IEEE*, 88, "Special Issue on Hybrid Systems" (invited paper)(7):888–912, July 2000.

[3] K. Kapellos D. Simon, R.P. Gibollet and B. Espiau. Synchronous composition of discretized control actions: design, verification and implementation with orccad. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, pages 158–65, 1999.

[4] M. Gunzert and A. Nagele. Component-based development and verification of safety critical software for a brake-by-wire system with synchronous software components. In *Proceedings International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 134–45, 1999.

[5] N. Halbwachs. *Synchronous Programming of Reactive Systems.* Kluwer Academic Publishers, 1993.

[6] E. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, December 1998.

[7] J. Pulou V. Bertin, M. Poize and J. Sifakis. Towards validated real-time software. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 157–64, 2000.