Taxys = Esterel + Kronos
A tool for verifying real-time properties of embedded systems [1] [2]

V. Bertin[3] E. Closse[4] M. Poize[3] J. Pulou[3] J. Sifakis[5] P. Venier[3]
D. Weil[3] S. Yovine[4]

Abstract

The goal of TAXYS is to provide a framework for developing real-time embedded code and verifying its correct behavior with respect to quantitative timing requirements. To achieve so, TAXYS connects France Telecom's ESTEREL compiler SAXO-RT with VERIMAG's model-checker KRONOS. TAXYS has been successfully applied on real industrial telecommunication systems, such as a GSM radio link from Alcatel and a phone prototype from France Telecom.

1 Introduction

Real-time systems are those systems whose correct functioning is subject to and must ensure the fulfillment of strict timing constraints such as tight execution and response times. Timing errors in embedded real-time software are very difficult to detect as its behavior is extremely sensitive to both the underlying execution platform and the patterns of external stimuli. The development of such software needs using rigorous methods and tools for generating executable code that meets the timing requirements at run-time.

Typically, a real-time application is composed of a set of components that interact. The role of the *scheduler* is to dispatch their execution in a way such that the timing requirements imposed on the components (e.g., period, deadline) and the whole system (e.g., synchronization, progress, mutual exclusion) are met. An approach to schedulability that has been thoroughly studied in the real-time systems literature (e.g., [12]) consists in (1) giving a *scheduling policy* (e.g., RMA , EDF [18]), and (2) showing the system is *schedulable* with such a policy. This theory led to the development of real-time operating systems (RTOS) (e.g., [15]), which typically provide preemptive scheduling algorithms based on priorities and implement protocols for avoiding deadlocks (e.g., [21]). The scheduling is ensured at *run-time* by the RTOS scheduler. Correctness is thereby guaranteed by the fulfillment of the schedulability conditions. This approach proved to be useful for hard real-time applications where the behavior of the environment is not predictable. There are, however, real-time systems which interact with *strongly* constrained environments. For such applications, it is desirable to generate *tailor-made schedulers* that make optimal use of the underlying execution hardware and shared resources, guided by the knowledge of all the possible behaviors of the environment. In this case, we can use a *model* of the system, the environment and the requirements to build an application-specific scheduler. Several *model-based* approaches to schedulability have been proposed in the literature, e.g., [19, 6, 16, 1, 17].

In fact, a scheduler can be regarded as a *controller* that prevents the system from performing "undesired" actions that can lead to the violation of a deadline. Thus, the construction of a scheduler from a model can theoretically be rephrased as a *controller synthesis* problem. One well-known approach to controller synthesis in the context of discrete-event systems is the one first proposed in [25], where the actions of the system are classified as *controllable*, which are those under control of the application, or *uncontrollable*, which correspond to the actions of the environment. The role of the controller is to forbid or to force certain controllable actions to ensure the state of the system remains in a desired set. In [2] we have proposed a methodology for model-based schedulability of real-time systems that combines (1) recent results on compositional modeling of timed systems [11] and (2) the controller synthesis algorithm for timed automata [4] developed in [5]. Our approach has been experimented on several case studies that illustrated its interest but also exposed its weaknesses, notably its *high computational cost*. A compositional methodology to try to overcome this problem is investigated in [3].

In this paper, we pursue an alternative direction that consists in focusing on a particular programming lan-

[2] Partially supported by projects RNRT TAXYS ITEA-DESS.
[3] ST Microelectronics, Central R&D, DAIS, 850 rue Jean Monnet, 38926 Crolles, France. valerie.bertin@st.com.
[4] France Telecom R&D, 28 chemin du Vieux Chêne, 38243 Meylan, France. Contact: daniel.weil@rd.francetelecom.com.
[5] VERIMAG, Centre Equation, 2 Ave. Vignate, 38610 Gières, France. Contact: sergio.yovine@imag.fr.

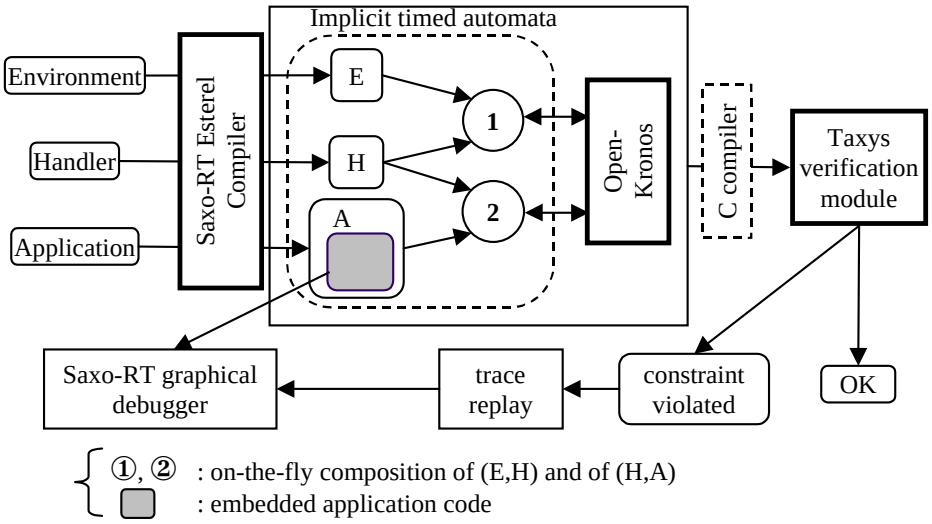

Figure 1: Taxys.

guage. Our goal is to embody specification-driven schedulability analysis in the compiler with the aim of generating *auto-scheduled code at compile time*.

2 Taxys

We use ESTEREL [10] as development language of the application. This language provides powerful constructs for management of parallelism and exceptions. It has rigorously defined semantics. ESTEREL programs run in a single thread on a single processor and can access external data and call routines written in C for complex (numerical) computations. In this context, an embedded real-time program is decomposed into a control part, written in ESTEREL and a data-manipulation part written in C. The program is compiled with the ESTEREL compiler SAXO-RT [24] that generates very efficient sequential C code targeted to mono-processor hardware architectures. The order of execution of the specified parallel modules is resolved at compile time. Such an implementation is (functionally) correct modulo the so-called *synchrony* assumption, which states that the program always reacts faster than its environment [7]. However, it may not satisfy the real-time requirements which depend on the execution speed of the target machine and the timing constraints imposed by the environment. In practice, validating the synchrony assumption amounts to show that the environment does not take too much lead over the application. This requires the use of a "realistic" synchrony assumption strongly depending on the application and its interactions with the environment. To interface the real-time system with its environment, we need to use an external event-handler which keeps track of the history of the environment and triggers computation steps of the application.

To check whether the program meets the timing requirements, we have developed the tool TAXYS [9, 13], which integrates SAXO-RT and the verification techniques implemented in the tool KRONOS [14]. The basic underlying idea of TAXYS consists in producing a formal model (actually, a timed automaton [4]), that captures the real-time behavior of the whole application composed of the program, its execution constraints, and the external environment. The ESTEREL source code is annotated with timing constraints of two kinds: (a) time intervals associated with the execution times of the external C functions, and (b) the requirements to be satisfied. TAXYS produces timed model of the self-sequenced executable code of the application. This model is composed with a timed model of the external environment. The global model is analyzed to validate timing requirements. Whenever a property is not satisfied, TAXYS generates a source-level error trace. In many cases, the violation of a timing constraint is indeed caused by the wrong execution order of some computations triggered by independent events within a reaction (or synchronous step). Such problems can be remedied by automatically re-arranging the sequential code at compile time. To do so, the information derived from the error trace can be used as a compiler directive to make SAXO-RT generate a correctly scheduled sequential implementation of the ESTEREL program.

TAXYS actual design flow is shown in Fig. 1. The overall system is made up from three components: (1) the ESTEREL program A, (2) the environment E, and (3) the external event handler H. The environment E is also an ESTEREL program. Since synchronous programs are deterministic, we added the statement

npause to the ESTEREL language to be able to model non-deterministic behaviors. Timing constraints are specified as pragmas in the ESTEREL code of A and E. The event handler H determines the way external events are taken into account by the program to generate an image of the state of the environment which is consistent with the synchronous semantics. Typically, external events are stored in a FIFO queue. The behavior of H is specified in terms of the size of the buffer and the incoming events. These can be specified to be *cumulative* or *separator*. All the occurrences of a cumulative event are stored in the buffer. It is assumed that between two successive such occurrences there exists a reaction of the program that consumes it. The occurrence of a separator events marks the beginning of a new synchronous instant. This information is used to determine the events that are consumed by the program at each reaction.

SAXO-RT generates three C modules which compute A, H and E transition functions. KRONOS explores the system state-space by composing on-the-fly the three models. The C code is indeed an instrumented version of the actual embedded code. Thus, the model of the system is produced by running the (instrumented) embedded code itself. The on-the-fly analysis avoids the state explosion problem and only computes the reachable states for the given environment model. If a timing constraint is violated, a trace leading to this error is generated. This trace can be re-executed step by step on the SAXO-RT graphical debugger to provide to the user more precise diagnostics.

3 Timing analysis

We make the assumption that the execution time of the program is spent by the C functions called from the ESTEREL program. This hypothesis is true for many reactive applications if the embedded code has been compiled efficiently as with SAXO-RT [24]. The execution times of the C functions can be estimated by profiling or static analysis techniques. The ESTEREL code is annotated with pragmas specifying this information. TAXYS checks two kinds of timing requirements, namely *throughput* and *deadline* constraints. A throughput constraint expresses the fact that the program reacts *fast enough* for the given environment model. The violation of a throughput constraint corresponds to an overflow of H which is checked by TAXYS by default. A deadline constraint is imposes a maximum delay between a given input event from the environment and a the corresponding reaction from the program. Deadlines are specified by annotations on the ESTEREL code.

We illustrate the approach with a simple example depicted in Fig. 2. The program is composed of two parallel modules that control some physical device. The filter F is triggered by the external event `SensorData` cyclically emitted by a sensor with a minimum delay of 65ms and a maximum of 70ms between each occurrence. The role of F is to perform some computation using the data and store the result in a shared variable. This computation takes between 20ms and 25ms. The computed value is then used by the controller C to approximate the state of the device at that moment and apply the desired control, called a pulse. The controller C is periodically triggered by the event `PulsePeriod`. Computing the pulse takes between 10ms and 15ms. Pulses should be applied every 100ms with an admissible delay of at most 40ms. The value used to compute the pulse must be less than 85ms old.

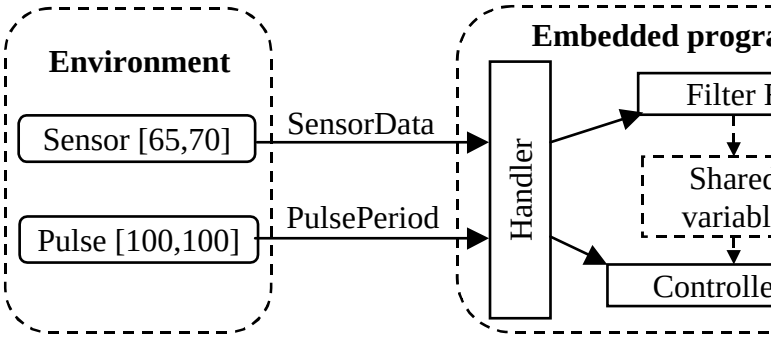

Figure 2: Simple example.

The ESTEREL code of the program is depicted in Fig. 3. The comments between `%{` and `}%` are the annotations carrying the timing information. The execution times are given as intervals which are actually interpreted as being closed. Variables `SD` and `PP` are clocks as in the timed automata theory, that is, they are continuous variables that progress at equal rate. The assignment `SD:=age(SensorData)` sets `SD` to the time elapsed since the occurrence of the event `SensorData` consumed in the current reaction. Notice that this is not necessarily the *last* occurrence of the event which may still be in

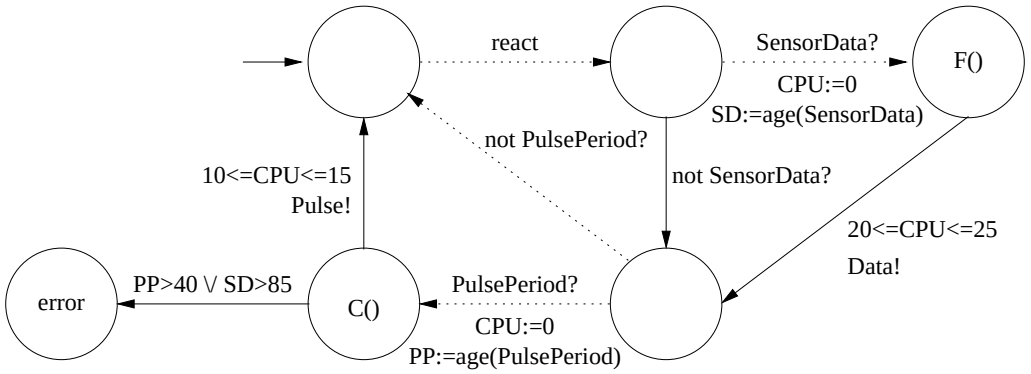

Figure 5: Program's model.

```
loop
  await SensorData %{SD:=age(SensorData)}%;
  call F() %{(20,25)}%;
  emit Data
end loop
||
loop
  await PulsePeriod %{PP:=age(PulsePeriod)}%;
  call C() %{(10,15), PP≤40 ∧ SD≤85}%;
  emit Pulse
end loop
```

Figure 3: Program.

```
emit SensorData;
loop
  npause %{65≤X≤70, X:=0}%;
  emit SensorData
end loop
||
loop
  npause %{Y=100, Y:=0}%;
  emit PulsePeriod
end loop
```

Figure 4: Environment.

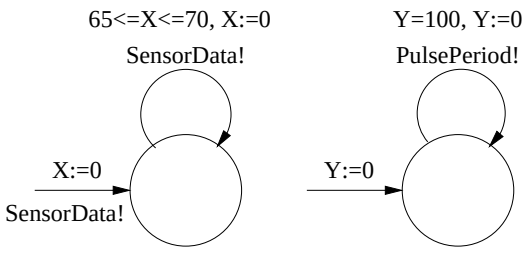

Figure 6: Environment's model.

the queue. The constraint PP≤40 ∧ SD≤85 expresses the requirements: the deadline for a pulse is 40ms and the data used to compute the pulse has been obtained at most 85ms before. The code of the environment is depicted in Fig. 4. Variables X and Y are clocks that are reset to 0 each time the environment emits SensorData and PulsePeriod, respectively.

The corresponding (extended) timed automata models are depicted in Fig. 5 and Fig. 6. Dotted arrows represent eager transitions, that must happen as soon as they become enabled. The "react" transition is the beginning of a reaction that starts as soon as there is an event in the buffer of the handler. In this example, the handler delivers all the events currently present. Notice that, between the two possible orderings which are consistent with the semantics of ESTEREL, the sequential code generated by the compiler SAXO-RT schedules the filter F first when both events are simultaneously present.

TAXYS runs in less than a second, generates about 300 symbolic states and concludes that the timing requirements are satisfied. A symbolic state is composed of the state of the program (i.e., a valuation of the signals SensorData and PulsePeriod as present or not, and a control point), the state of the event handler (i.e., a configuration of the queue), the state of the environment (similar to the program), and a constraint on the clocks (i.e., a DBM structure used by KRONOS [14].)

4 Experimental results

4.1 Communication mode of a GSM terminal

This case study is inspired from the communication mode of a GSM mobile terminal [20]. The complete design and verification of the radio link of a GSM terminal developed by Alcatel is reported in [8]. We describe here a small part of the application [9] which is composed of two modules (Fig. 7). When the event

```
loop
  await Prepar %{X:=age(Prepar)}%;
  call RF() %{(50,50), X≤80}%;
  await Receipt %{X:=age(Receipt)}%;
  call demodulation() %{(80,100), X≤150}%;
  call decoding() %{(20,20)}%
end loop
||
loop
  await Freq %{Y:=age(Freq)}%;
  call frequency_comp() %{(40,40), Y≤90}%
end loop
```

Figure 7: Simplified GSM communication mode.

`Prepar` occurs, the first module calls the C function `RF()`, which takes 50ms to prepare the radio front end of the mobile terminal in order to receive data. Then, when the event `Receipt` occurs, it sequentially executes `demodulation()`, that takes between 80ms and 100ms, followed by `decoding()`, that finishes in 20ms. The second module is triggered by the event `Freq` and executes the C function `frequency_comp()`, which consists in calculating the frequencies on which the data are going to be received and completes in 40ms. The computations are subject to the timing constraints annotated in the code.

The code of the full application developed by Alcatel consists in 815 lines of ESTEREL and 48000 lines of C. The application was validated for 62 test environments provided by Alcatel. 4 scenarios were found to lead to deadline violations caused by a wrong scheduling of calls. These errors were corrected by slightly changing the ESTEREL code.

4.2 ISDN prototype phone

In [13] we presented the results obtained on a digital phone prototype carrying simultaneously voice and data produced by a graphic tablet, implemented on a 32 MIPS Digital Signal Processor (Fig. 8). The prototype has an audio input channel sampled at 8kHz that is connected to the microphone, an rs232 input channel carrying data from the graphic tablet, and an input channel sampled at 8 kHz to retrieve audio and graphic data sent by the network (TNR). Processing audio data consumes 3900 CPU cycles over the 4000 CPU cycles available every 125μs. Graphic data are compressed by a vectorization algorithm which consumes sporadically between 15000 and 20000 CPU cycles.

Clearly, there is a risk of buffer overflow if graphic data samples arrive too often. We have used TAXYS to analyze the relationship between the size of the input buffers and the arrival rate of graphic data. We carried out 6 experiments with the same ESTEREL code for the program but with different environment models and handler buffer sizes. **ISDN**$_1$ and **ISDN**$_2$ with an environment model composed of two strictly periodic and independent tasks (the first carrying audio data at 8kHz and the second the graphic tablet data at 100Hz). **ISDN**$_3$ and **ISDN**$_4$ with the second task being aperiodic and emitting bursts at rates varying in a non-deterministic manner between 25 and 100Hz. **ISDN**$_5$ and **ISDN**$_6$ with a third additional periodic task modeling switching between several audio modes. In all cases, the program consists of 3000 C lines and 258 ESTEREL lines, and the environment of 120 ESTEREL lines. Results presented in table 1 show that a buffer size of at least 6 is necessary for absorbing the sporadic task. We observe that the number of symbolic states explored by KRONOS increases exponentially with the "degree" of non-determinism of the environment. Therefore, to cope with state explosion due to environment non-determinism, it is necessary to find appropriate environment model approximations preserving the verified properties.

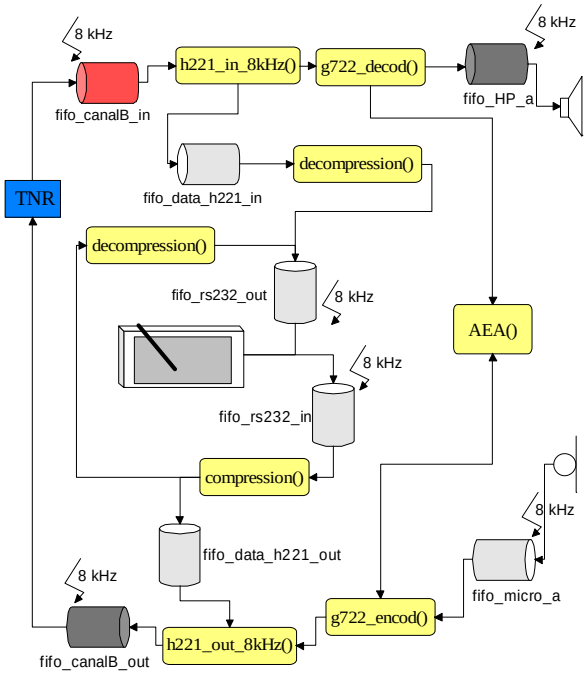

Figure 8: ISDN phone prototype.

Name	Buff. size	Symb. States	Verif. Time	Diagnostic
ISDN$_1$	5	2 200	1.27 s	overflow
ISDN$_2$	6	10 849	5 s	OK
ISDN$_3$	5	15 894	6.29 s	overflow
ISDN$_4$	6	633 472	10 mn 47 s	OK
ISDN$_5$	5	22 695	13.6 s	overflow
ISDN$_6$	6	$> 10^7$	?	aborted

Table 1: Experimental results for the phone prototype.

5 Conclusion

We have presented an original approach for specifying, designing and validating real-time embedded systems. Compared to other approaches where verification is done on an abstract model, in TAXYS the embedded code itself is effectively executed during verification. Timing analysis is done on-the-fly during the execution of the appropriately instrumented C code generated by the compiler. Instrumentation allows the verifier to observe the execution time of the application code specified in the pragmas. This approach is fully implemented in an entirely automated tool that demonstrated to be applicable to mid-size applications from the telecommunications industry. Recently, we started using TAXYS in other application domains, such as vehicle control software [23].

References

[1] Y. Abdeddaïm and O. Maler. Job-shop scheduling using timed automata. In *CAV'01*. LNCS 2102, Springer, 2001.

[2] K. Altisen, G. Goessler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *IEEE RTSS'99*, Phoenix, AZ, USA, December 1999. IEEE Computer Society Press.

[3] K. Altisen, G. Gößler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. To appear in *Journal of Real-Time Systems, special issue on control-theoretical approaches to real-time computing*, 2001.

[4] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994. Elsevier Science.

[5] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid System II*. LNCS 999, Springer, 1995.

[6] H. Ben-Abdallah, J.-Y. Choi, D. Clarke, Y. Kim, I. Lee and H.-L. Xie. A process algebraic approach to the schedulability analysis of real-time systems. *Real-Time Systems*, 15, 1998.

[7] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.

[8] V. Bertin. Validation d'applications temps-réel par analyse de programmes synchrones temporisés. PhD. Thesis, INPG 2000.

[9] V. Bertin, M. Poize, J. Pulou, J. Sifakis. Towards Validated Real-Time Software. In *12th Euromicro Conference on Real-Time Systems*, Stockholm, Sweden, June 2000

[10] G. Berry, G. Gonthier, The ESTEREL Synchronous Programming Language : Design, Semantics, Implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[11] S. Bornot and J. Sifakis. An algebraic framework for urgency. *Information and Computation*, 163:172–202, 2000. Academic Press.

[12] A. Burns. Scheduling hard real-time systems: a review. *Software Engineering Journal*, p. 116–128, May 1991.

[13] E. Closse, M. Poize, J. Pulou, J. Sifakis, P. Venier, D. Weil, and S. Yovine. TAXYS: a tool for the developpment and verification real-time embedded systems. In *CAV'01*. LNCS 2102, Springer, 2001.

[14] C. Daws, A. Olivero, S. Tripakis and S.Yovine. The tool KRONOS. In *Hybrid Systems III, Verification and Control*, LNCS 1066, Springer, 1996.

[15] IEEE Standard P1003.4. Real-time extensions to POSIX. 1991.

[16] H. Kwak, I. Lee, A. Philippou, J. Choi, and O. Sokolsky. Symbolic schedulability analysis of real-time systems. In *IEEE RTSS'98*, Madrid, Spain, December 1998. IEEE Computer Society Press.

[17] K. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: efficient cost-optimal reachability for priced timed automata. In *CAV'01*. LNCS 2102, Springer, 2001.

[18] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973. ACM.

[19] A. K. Mok, D. C. Tsou and R. C. M. Rooij. The MSP.RTL Real-Time Scheduler Synthesis Tool. In *RTSS'96*, Washington, DC, USA, December 1996. IEEE Computer Society Press.

[20] M. Mouly and M.B. Pautet. The GSM System for Mobile Communication. Cell&Sys, 1992.

[21] R. Rajkumar. *Synchronization in real-time systems: a priority inheritance approach*, Kluwer Academic Publishers, 1991.

[22] S. Tripakis. Software Architecture of PATH's Automated Vehicle Control. Technical report UCB/ERL M01/6, 2001.

[23] S. Tripakis and S. Yovine. Timing Analysis and Code Generation of Vehicle Control Software using Taxys. *ENTCS*, 55(2):174–183, 2001. Elsevier.

[24] D. Weil, V. Bertin, E. Closse, M. Poize, P. Venier, J. Pulou. Efficient Compilation of ESTEREL for Real-Time Embedded Systems. In *CASES'2000*, San Jose, November 2000.

[25] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3):637–659, May 1987.