Towards Validated Real-Time Software*

Valérie BERTIN†, Michel POIZE, Jacques PULOU

France Télécom - Centre National d'Etudes des Télécommunications
28 chemin du Vieux Chêne - BP 98 - 38243 Meylan cedex - France
{valerie.bertin, michel.poize, jacques.pulou}@cnet.francetelecom.fr

Joseph SIFAKIS

VERIMAG - Centre Equation
2 avenue de Vignate
38610 Gières - France
Joseph.Sifakis@imag.fr

Abstract

We present a tool for the design and validation of embedded real-time applications. The tool integrates two approaches, the use of the synchronous programming language ESTEREL for design and the application of model-checking techniques for validation of real-time properties. Validation is carried out on a global formal model (timed automata) taking into account the effective implementation of the application on the target hardware architecture as well as its external environment behavior.

Keywords : *real-time computing, embedded applications, real-time constraints, compiling, synchronous languages, validation, timed automata, model-checking, external environment modelling.*

1 - Introduction

The correct behavior of real-time applications depends not only on the correctness of the results of computations but also on the times at which these results are produced. Their development requires rigorous methods and tools to reduce development costs and "time-to-market" while guaranteeing the quality of the produced code (in particular, respect of the temporal constraints).

The above requirements motivated the development of the TAXYS tool, dedicated to the design and validation of real-time telecommunications software.

One of the major goal of the TAXYS tool is to produce a formal model that captures the temporal behavior of the whole application which is composed of the embedded computer and its external environment. For this purpose we use the formal model of timed automata from Alur and Dill [8]. The choice of this model allows the use of results, algorithms and tools available [9], [10]. We use the KRONOS model checker [8] for model analysis.

From the source code of the application, an ESTEREL program annotated with temporal constraints, the TAXYS tool produces on the one hand self-sequenced executable code and on the other hand a timed model of the application. This model is again composed with a timed model of the external environment in order to obtain a global model which is statically analyzed to validate timing constraints. This validation should notably shorten design time by limiting tedious test and simulation sessions.

We present the TAXYS tool and the associated approach. In section 2, we expose the general architecture of TAXYS and the different steps of the development of an application with this tool. In section 3, we provide two simple examples illustrating the approach.

2 - TAXYS architecture

The use of synchronous languages [3] for the development of real-time reactive applications relies on a "synchrony assumption" meaning that the application reacts infinitely fast with respect to its environment. This assumption, very convenient in practice, must be validated for a given implementation on a target machine. In this validation, dynamic features of the machine and of the external environment must be taken into account. The speed of the machine determines duration of atomic actions executed in synchronous program computation steps. In practice, validating the synchrony assumption amounts to show that the environment does not take too

* This work is supported in part by the RNRT project TAXYS.
† Alcatel Business Systems - Colombes - France

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.

2.1 - Structure of a TAXYS application

To interface the real-time system with its environment, we use an External Events Handler (EEH) which keeps track of the history of the environment and triggers computation steps of the application.

The EEH receives **events** generated by the external environment and delivers sets of **signals** called **stimuli** to the Polling Execution Structure (PES) (see figure 1). The PES is a cyclic process, driven by stimuli, which executes a single procedure, called REACT.

2.1.1 - External Events Handler (EEH)

The correspondence between stimuli and history of external events depends on the way significant external environment state changes are taken into account by the application. Correctness of the implementation strongly depends on the policy of the EEH. Among the different policies to build stimuli, we distinguish two extreme ones :

- The external events history is structured as a FIFO queue and each execution of REACT consumes the stimulus, corresponding to a single event, placed on top of the queue (FIFO policy).
- The external events history is not structured ; when REACT procedure starts its execution, all the events in the buffer are taken to be in the same stimulus (Greedy Policy).

In this model we assume that there is no busy waiting : external events are recorded by interrupt routines which will be considered to be timeless as a first approximation ; their duration is negligible with respect to execution times of treatments in the application (a more accurate model can be obtain by adding this overhead to these execution times). The stimuli are recorded in a buffer.

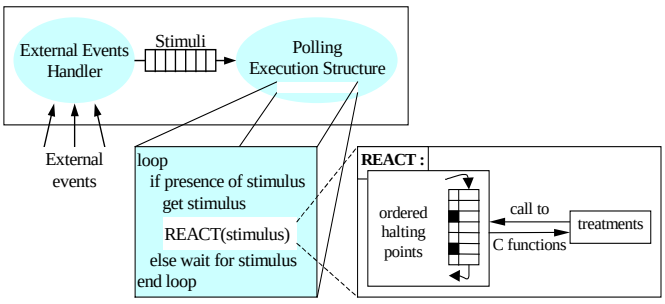


Figure 1. Structure of a TAXYS application

2.1.2 - Polling Execution Structure (PES)

We use ESTEREL [4] as development language of the application. This language provides powerful constructs for management of parallelism and exceptions. It has rigorously defined semantics. ESTEREL programs can refer to external data and routines written in C.

We have extended ESTEREL with pragmas (compiling directives) intended to specify temporal constraints of applications. A temporal constraint corresponds to a pair of locations in the ESTEREL code (location of triggering of the constraint, location of test of the constraint).

We use the SAXO-RT compiler [5] to generate from an ESTEREL source file a PES intermediate C code in the form of a cyclic process that gets a stimulus from the EEH and calls the REACT procedure. The latter uses an array of **halting points** associated with C functions.

Halting points are control locations of the ESTEREL source code where processing may stop waiting for some internal or external signals. External signals belong to stimuli provided by the EEH as mentioned previously. Internal signals can be either explicit (e.g. the declared ESTEREL internal signals) or implicit (e.g. synchronous termination of parallel tasks). At run time, a halting point can be either active or inactive.

Each control state of a program corresponds to a single set of active halting points. If a halting point is active and sensitive to the current stimulus, then the associated C function is executed. The execution consists in modifying data and computing the next control state.

The REACT procedure scans the halting points in a pre-calculated order guaranteeing the respect of causality constraints imposed by the ESTEREL semantics. This order, which realizes the interleaving of concurrent tasks, is chosen at compile time.

We use the SAXO re-targetable optimized compiler [2] to generate executable code from the PES intermediate C code and for the EEH definition. SAXO is dedicated to embedded applications on irregular processing architectures such as Digital Signal Processors (DSP). It produces, from the functional description of the application (set of C files) and from the description of the hardware architecture, a memory mapped machine code executable as stand-alone code. No software kernel is needed.

SAXO takes into account the features of the hardware to optimize the generated code. This allows a precise evaluation of execution times of functions, used for temporal validation.

2.2 - Modeling of a TAXYS application

To check synchrony assumption and temporal constraints, we build a global timed model by composition of the two timed automata representing the embedded code behavior, and the external environment

coupled by the finite automaton model representing the EEH. The timed automaton representing embedded code behavior is generated automatically from the PES intermediate code produced by the SAXO-RT compiler. The automata modeling the environment and the EEH are provided separately.

The validation is carried out by using the KRONOS tool. KRONOS [8] is a symbolic model checker for real-time applications modeled as a timed automaton. It allows in particular, to check whether a real-time application satisfies a property specified as a formula of the real-time temporal logic TCTL [8].

Timed automata [1] are automata extended with continuous variables called **clocks** that measure the time elapsed at states and can be tested and reset at transitions. KRONOS can be used to check logical properties – such as mutual exclusion, reachability, inevitability or lack of deadlock – as well as temporal constraints such as bounds on the time between the execution of two transitions in the timed automaton. If a property is not met, KRONOS generates some diagnostics for error detection and correction.

The properties characterizing the synchrony assumption depend on the way the EEH establishes the correspondence between external events and stimuli. They can be expressed as constraints of the form $| t_e - t_o | < D$ where t_o is the time of occurrence of an external event and t_e the termination time of the treatment triggered by the occurrence of the event (see figure 2).

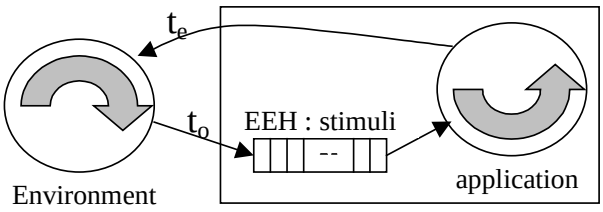


Figure 2. Temporal behavior of a TAXYS application

2.2.1 - Timed model of the environment

We consider that the environment generates a set of **events** $S\uparrow=\{s_1\uparrow, s_2\uparrow, \ldots, s_n\uparrow\}$. The event sequences of the environment are modeled as a timed automaton whose transitions are labeled with events, elements of $S\uparrow$.
Let $S=\{s_1, s_2, \ldots, s_n\}$ be the set of input **signals** of the ESTEREL source program. These signals are in bijection with the set $S\uparrow$ of external environment events. The event $s_i\uparrow$ is a state change of the environment that makes s_i = true.

The automaton for the environment is described in ESTEREL extended with functions that manipulate time and clocks. This description is then translated in a timed automaton by the tool Env2TM.

2.2.2 - The model of the EEH

The EEH determines the way external events are taken into account by the application to generate an image of the state of the environment which is nothing but the application's view on the environment. An external event $s\uparrow$ is taken into account when a stimulus containing the signal s is produced. The EEH can be modeled as an automaton with two types of transitions : input transitions that are triggered by external events (elements of $S\uparrow$) and output transitions labelled by stimuli (subsets of S).

From any state, there exist successors for any possible input event but a unique output transition labeled with a stimulus. The automaton modeling the EEH can be considered as a transducer from external event flow to stimuli sequence. The EEH is described in a specific language and is then translated in the aforementioned automaton by the tool EEH2A.

The language allows to characterize the EEH in terms of the size of its buffer, the set of the input events and their attributes. These attributes are used to specify precisely the different policies for taking external event flow into account. According to the type of the application, it may be important to generate as many occurrences of a signal s as the number of the occurrences of the event $s\uparrow$ ($s\uparrow$ is **cumulative**) or to consider that the events have no cumulative effect ($s\uparrow$ is **coalescent**).

For a cumulative event it is assumed that between two successive occurrences there exists a system reaction consuming it. We show later how such a property can be checked : for example, an alarm event is not cumulative in general, as the system must react as soon as possible.

Another attribute is the **separator** attribute. If an event is a separator then after its occurrence a new stimulus is constructed. As a consequence, two distinct occurrences of a separator event appear in distinct stimuli.

```
Handler =
       { buffer_size=4
       event S1 : separator
       event S2 : cumulative
       event S3 :
       event S4 :
%default : S3 is coalescent as S4
       }
```

Example : For a FIFO policy, all events must be separators while for a greedy policy no event is a separator.

2.2.3 - Timed model of the application

The timed model of an application is generated from the intermediate code by the tool IC2TM. It is a timed automaton with two kinds of states :

- **waiting** states corresponding to configurations of halting points. Transitions issued from them are labeled by stimuli.

- **computation** states corresponding to execution of C functions associated with halting points. Transitions from computation states are labeled by elements of the set $S\downarrow=\{s_1\downarrow, s_2\downarrow, \ldots, s_n\downarrow\}$. The label $s_i\downarrow$ is used to denote the completion of the computation triggered by the external event $s_i\uparrow$.

From a waiting state (set of active halting points) and a given stimulus, the application automaton takes the transition sensitive to the stimulus and reaches a computation state. From this state, it computes the set of C functions associated with the active halting points of the waiting state and sensitive to the consumed stimulus. The automaton will stay at the computation state for some time corresponding to the execution time of these functions. Then, it will get to the next waiting state by a transition labeled with $s_i\downarrow$, indicating completion of the computation of functions triggered by external events $s_i\uparrow$.

To distinguish between termination times of functions associated with a computation state, the latter can be split up into a sequence of computation states. Transitions between these states are labeled by elements of the set $S\downarrow$ and correspond to the termination of individual functions.

2.3 - TAXYS general architecture

SAXO-RT, SAXO, Env2TM, EEH2A, IC2TM and KRONOS are integrated into the TAXYS environment as shown in figure 3. TAXYS takes as inputs the description of the application, of the EEH and of the environment, the constraints that have to be verified and the features of the hardware target architecture.

From ESTEREL source code annotated with temporal constraints and from C functions, the SAXO-RT tool produces an intermediate code. This intermediate code is compiled by the SAXO tool in order to generate self-sequenced code directly executable by the target architecture.

A timed model of the implementation of the application is then produced from the intermediate code by the IC2TM tool. The environment is also modeled as a timed automaton by the Env2TM tool. The EEH is modeled as an automaton by the EEH2A tool. The three automata are then composed, in order to obtain a global model, and analyzed by the KRONOS tool which generates diagnostics if the timing constraints are not satisfied.

3 - Two simple examples to illustrate the methodology

3.1 - Scheduling of two tasks

3.1.1 - Specifications

Consider an application reacting to two periodic events, $s_1\uparrow$ and $s_2\uparrow$, each one requiring the execution of a dedicated task, *task1* and *task2*, respectively. The tasks are completed within intervals $[C_1,C_1']$ and $[C_2,C_2']$, respectively. Each task must terminate before the occurrence of the next request of its execution.

The events $s_1\uparrow$ and $s_2\uparrow$ are generated by the environment and are characterized by their period T_i, $i \in \{1,2\}$, and their relative shift δ.

Notice that our example follows closely assumptions about applications in RMA or EDF scheduling [6] except that we forbid preemption.

3.1.2 - Timed model of the application

The ESTEREL application code with temporal constraints is given in figure 4(a).

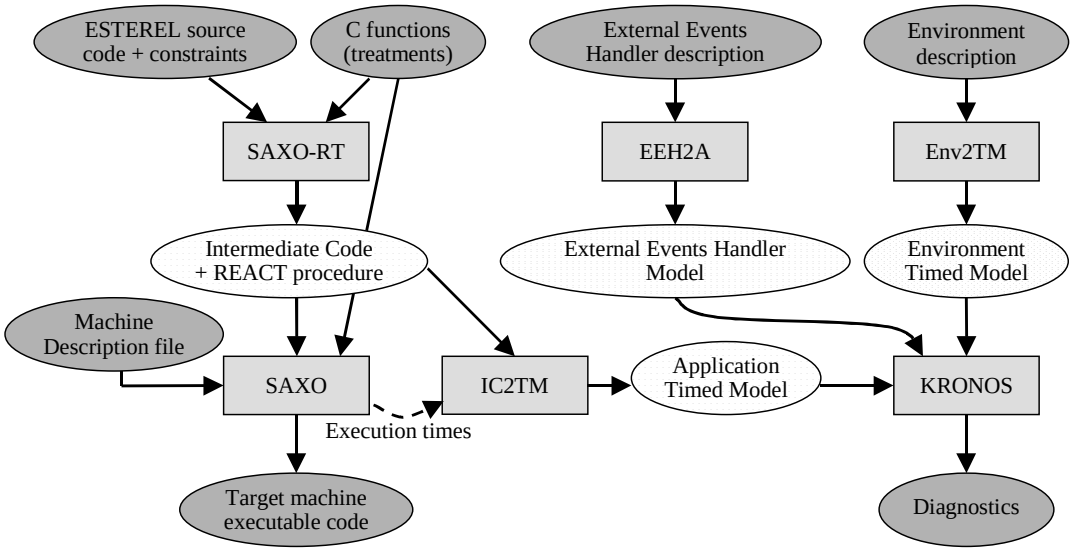
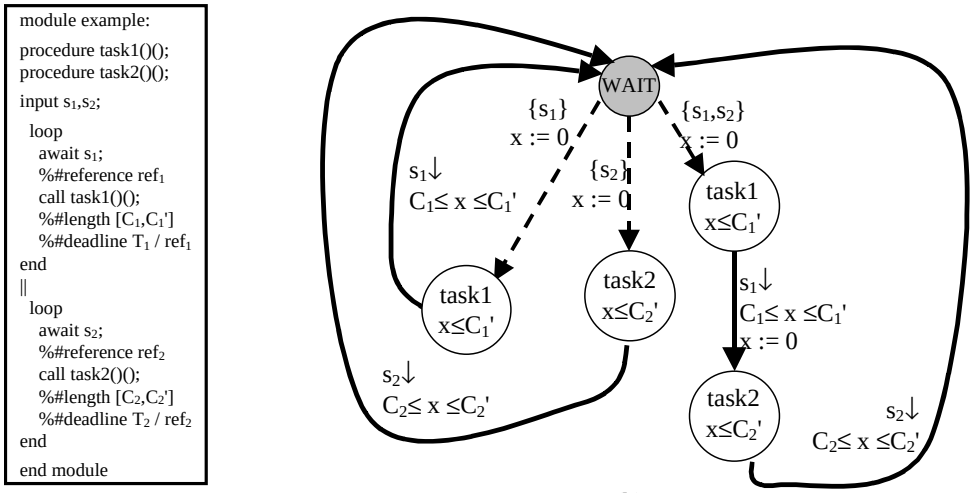


Figure 3. TAXYS general architecture



(a) (b)

Figure 4. (a) ESTEREL application code - (b) Timed model of the application

The corresponding timed automaton is represented in figure 4(b). It has only one waiting state (WAIT) while all the other states are computation states. Edges labeled by $s_1\downarrow$ (respectively $s_2\downarrow$) correspond to the end of the execution of *task1* (respectively *task2*) and can be considered as acknowledgments of event $s_1\uparrow$ (respectively $s_2\uparrow$). In the case where the stimulus $\{s_1, s_2\}$ is present, the SAXO-RT compiler gives priority to *task1*. This priority is an implementation choice as SAXO-RT takes into account only causality preservation rules when it sequentializes by interleaving the parallel tasks. In this case, our model captures the fact that the two tasks do not terminate at the same time.

The clock x is used to control execution times. On the model, thick edges correspond to completion of tasks and dashed edges correspond to beginning of execution cycles (REACT procedure).

3.1.3 - Timed model of the environment

Each one of the two events $s_1\uparrow$ and $s_2\uparrow$ is generated by a timed automaton A_i, $i \in \{1,2\}$, using a clock Z_i (see figure 5).

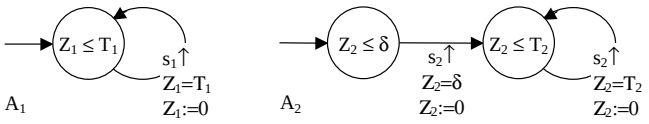


Figure 5. Timed automata for events $s_1\uparrow$ and $s_2\uparrow$

The global environment behavior is modeled as a timed automaton which is the synchronized product of the two independent timed automata A_1 and A_2 (see figure 6). In this model, $s_1s_2\uparrow$ denotes the simultaneous occurrence of $s_1\uparrow$ and $s_2\uparrow$.

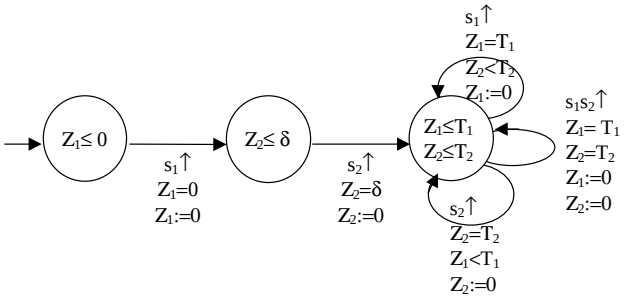


Figure 6. Timed model of the environment

3.1.4 - Model of the EEH

We consider two EEH for this application. The first one has a buffer of size one and both input events are non-separators and cumulative, that is :

- All events which occur in the external environment during a reaction cycle are consumed simultaneously in the next reaction cycle.
- If an event occurs whereas the last occurrence of the same event has not been consumed by the application, then an error condition is reported (represented in the model by an error state).

This EEH is specified below and its automaton model is shown in figure 7. It is used to build the global model analyzed in section 3.1.5.

```
Handler ={
       buffer_size=1
       event S1 : cumulative
       event S2 : cumulative
}
```

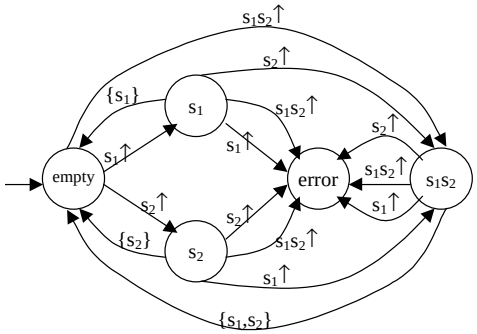


Figure 7. Model of the EEH – example 1

The second EEH has a buffer of size two and both input events are separators and cumulative, that is :

- The events are consumed one by one in the order of their occurrence (except when they occur exactly at the same time in the environment).
- If an event occurs whereas the last occurrence of the same event has not been consumed by the application, then an error condition is reported (represented in the model by an error state).

This EEH is specified below and its automaton model is shown in figure 8.

```
Handler ={
       buffer_size=2
       event S1 : separator
       event S2 : separator
}
```

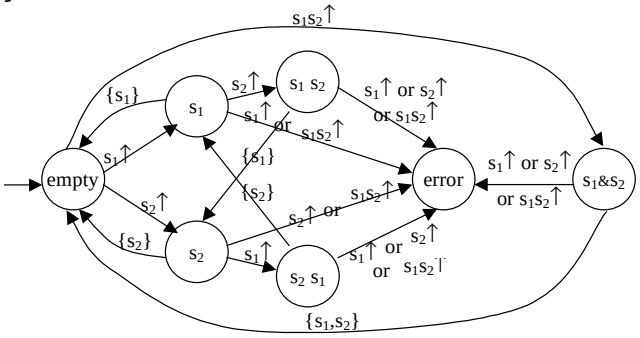


Figure 8. Model of the EEH – example 2

3.1.5 - Global timed model and results

For this example, the global timed model obtained as the composition of the three timed models has 25 states (including a unique error state representing all the global states containing the "error" state of the EEH), 94 transitions and 3 clocks.

We used the KRONOS tool to check correctness of the implementation for different values of the time constants δ, T_1, T_2, C_1, C_2 (see table 1). Correctness is characterized by the following properties :

- non-reachability of the error state.
- the maximum delay between $s_1\uparrow$ and $s_1\downarrow$ (respectively between $s_2\uparrow$ and $s_2\downarrow$) is bounded by T_1 (respectively T_2).

δ, T_1, T_2	0, 20, 50						3, 20, 50				0, 20, 53			
C_1	10	10	11	11	15	15	11	11	15	15	11	11	15	15
C_2	20	21	18	19	10	11	16	17	10	11	10	11	6	7
correct	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no	yes	no

Table 1. Correctness of some implementations

Figure 9 shows the surfaces representing the pairs (C_1, C_2) for which the implementation is correct, for given environment parameters (δ, T_1, T_2). The shape of the surfaces suggests that when no preemption is allowed, there is no simple analytical correctness condition similar to the one by Liu and Layland [6].

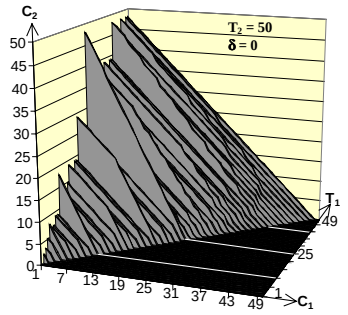


Figure 9. Correctness regions

3.2 - Communication mode of a GSM mobile terminal

This example is inspired from the communication mode of a GSM mobile terminal [7]. A small part of the application is presented in this section.

3.2.1 - Specifications

The application is described as an annotated ESTEREL program consisting of two tasks in parallel (see figure 10).

The first task executes successively the treatment *prepar-RF* (which consists in preparing the radio front end of the mobile terminal in order to receive data) when event $s_{prepar}\uparrow$ occurs then *demodulation* followed by *decoding* when event $s_{receipt}\uparrow$ occurs. The second task executes the treatment *frequency_computation* (which consists in calculating the frequencies on which the data are going to be received) whenever event $s_{freq}\uparrow$ occurs.

The treatments are subject to temporal constraints concerning their duration, release or deadline :

- the treatment *prepar-RF* (*RF* in figure 11) has a duration of 50 time units exactly and must be finished before 80 time units after the latest occurrence of event $s_{prepar}\uparrow$;

- the treatment *demodulation* (*DEM* in figure 11) has a duration between 80 and 100 time units, must begin at least 10 time units after the latest occurrence of event $s_{receipt}\uparrow$ and must be finished before 150 time units after the latest occurrence of event $s_{receipt}\uparrow$;
- the treatment *decoding* (*DEC* in figure 11) has a duration of 20 time units ;
- the treatment *frequency_computation* (*FC* in figure 11) has a duration of 40 time units and must be finished before 90 time units after the latest occurrence of event $s_{freq}\uparrow$.

3.2.2 - Timed model of the application

This application is modeled as the timed automaton of figure 11. In this timed automaton, the SAXO-RT compiler schedules the treatments *prepar-RF*, *frequency_computation*, *demodulation*, *decoding* taking into account causality rules between halting points (see section 2.1.2). Duration and release constraints are imposed on the model by using clocks X and C. The clock X is used to control treatments duration and C is a clock used to impose release constraints with respect to $s_{receipt}\uparrow$ which is generated by the environment. Deadline constraints are requirements to be verified on the global model.

3.2.3 - Timed model of the environment

The environment generates three periodic events $s_{prepar}\uparrow$, $s_{receipt}\uparrow$ and $s_{freq}\uparrow$ (see figure 12). $s_{prepar}\uparrow$ and $s_{receipt}\uparrow$ have the same period T and are separated by a phase shift of δ. $s_{freq}\uparrow$ has a period T' and has a phase shift δ' at the origin.

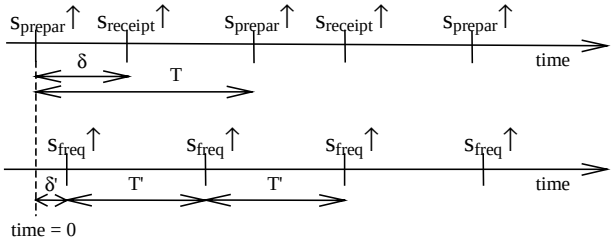


Figure 12. Behavior of the environment

```
module example:

procedure prepar-RF()();
procedure demodulation()();
procedure decoding()();
procedure frequency_computation()();

input sprepar, sreceipt, sfreq;

 loop
  await sprepar;
      % #reference REFprepar
  call prepar-RF()();
      % #length [50,50]
      % #deadline 80 / REFprepar
  await sreceipt;
      % #reference REFreceipt
  call demodulation()();
      % #length [80,100]
      % #release 10 / REFreceipt
      % #deadline 150 / REFreceipt
  call decoding()();
      % #length [20,20]
 end
||
 loop
  await sfreq;
      % #reference REFfreq
  call frequency_computation()();
      % #length [40,40]
      % #deadline 90 / REFfreq
 end

end module
```

Figure 10. ESTEREL source code with timing constraints

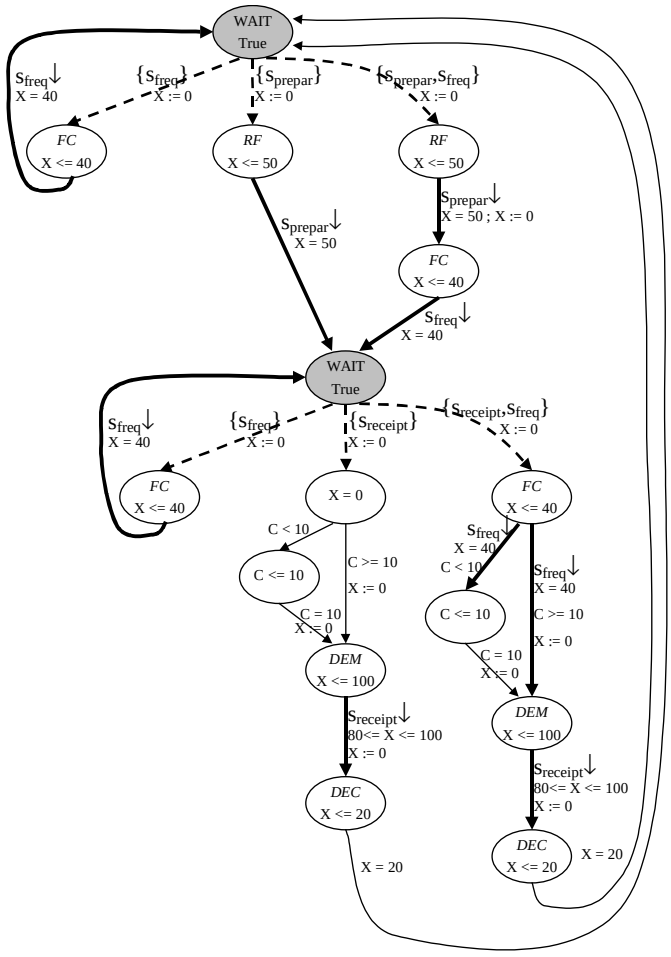


Figure 11. Timed model of the application

T, δ, T', δ'	210,60,210,60	240,60,320,30	240,60,320,60	240,60,330,60	240,60,160,60	240,60,160,85	240,120,160,40
deadline of *RF*	OK	OK	OK	OK	OK	overrun	overrun
deadline of *DEM*	OK	OK	OK	OK	OK	OK	OK
deadline of *FC*	OK	overrun	OK	overrun	OK	overrun	OK

Table 2. TAXYS diagnostics

The environment is modeled as a timed automaton, product of two independent timed automata, one for events $s_{prepar}\uparrow$ and $s_{receipt}\uparrow$ (because they have the same period) and one for event $s_{freq}\uparrow$. It has 6 states, 18 transitions and 2 clocks.

3.2.4 - Model of the EEH

We consider in this example, that the EEH has a buffer of size three. The three external events are separators and cumulative.

3.2.5 - Global timed model and results

KRONOS is used to check whether deadlines are respected or not. According to the periods of the external events (T and T') and to their initial phase shifts (δ and δ'), none, one or more deadline(s) can be overrun. The TAXYS diagnostics are presented in the table 2.

4 - Conclusions and future work

We have presented an approach for the design and validation of real-time applications. This approach combines the synchronous ESTEREL language for design and model-checking for validation.

The work reported is part of a more general project which aims automatic low level code generation from high level specifications of an embedded real-time application.

Current work is developed in the following directions :

- Mastering state explosion, an inherent limitation of model-checking techniques, by using "on-the-fly" symbolic analysis to avoid explicit construction of the global model. Such techniques have already been implemented in the KRONOS tool and are currently under evaluation on industrial applications.
- Increase of readability of diagnostics and traceability of the results. In particular, help the user identifying possible causes of an error.
- Generation of optimal schedulers, at compile time. The SAXO-RT compiler considers one arbitrary order of parallel tasks, provided it respects the causality requirements. We study scheduling techniques taking into account global dynamic features of the application as well as "realistic" synchrony assumptions.

References

[1] R. Alur, D. Dill, "Automata For Modeling Real-Time Systems", Proceeding of the 17th ICALP, Lecture Notes In Computer Sciences 443, Springer Verlag, pp. 322-335, 1990.

[2] J.C. Bauer, E. Closse, E. Flamand, M. Poize, J. Pulou, P. Venier, "SAXO, a re-targetable Optimized Compiler for DSPs", Proceedings of the 8th international Conference on Signal Processing Applications & Technology, San Diego, California, USA, pp. 1032-1036, September 14-17, 1997.

[3] G. Berry, A. Benveniste, "The Synchronous Approach to Reactive and Real-Time Systems", Another Look at Real-Time Programming, Proceedings of the IEEE, vol. 79, pp. 1270-1282, 1991.

[4] G. Berry, G. Gonthier, "The ESTEREL Synchronous Programming Language : Design Semantics, Implementation", Science of Computer Programming, vol. 19-2, pp. 87-152, 1992.

[5] V. Bertin, M. Poize, J. Pulou, "Une Nouvelle Méthode de Compilation pour le Langage ESTEREL [a New Method for Compiling the ESTEREL Language]", Proceedings of GRAISyHM-AAA, Lille, France, March 1999 (in French).

[6] C.L. Liu, J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", J. of the ACM, Vol. 20-1, pp. 46-61, January 1973.

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

[8] S. Yovine, "Model Checking Timed Automata", in Embedded Systems, G. Rozemberg anf F. Vaan Drager EDS, Lecture Notes in Computer Sciences, 1998.

[9] T.A. Henzinger, P. H. Ho, H. Wong-Toi, "HYTECH: a Model-Checker for Hybrid Systems", Journal of Software Tools for Technology Transfert, vol. 1-1/2, pp 110-122, 1997.

[10] K.G. Larsen, P. Petterson, Wang Yi. "UPPAAL in a nutshell", Journal of Software Tools for Technology Transfert, vol. 1-1/2, pp 134-152, 1997.