

# Polychronous Analysis of Timing Constraints in UML MARTE

Huafeng Yu Jean-Pierre Talpin

Loïc Besnard Thierry Gautier

IRISA/INRIA Rennes/CNRS

263, avenue du Général Leclerc, Rennes, France

Email: {firstname}.{lastname}@irisa.fr

Frédéric Mallet Charles André

Robert de Simone

Université de Nice/INRIA Sophia Antipolis

2004 route des Lucioles, BP 93, Sophia Antipolis, France

Email: {firstname}.{lastname}@sophia.inria.fr

**Abstract**—The UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) defines a broadly expressive Time Model to provide a generic timed interpretation for UML models. As a part of MARTE, Clock Constraint Specification Language (CCSL) allows the specification of systems with multiple clock domains as well as nondeterminism. In this paper, we propose to take advantage of Polychrony clock calculus, named hierarchization, to analyze timed systems specified in CCSL, and to generate code for simulation considering determinism. Hierarchization enables to identify the endochrony property in a system that allows code generation ensuring determinism. The presented work is being integrated into the TimeSquare environment dedicated to the simulation of MARTE timed systems.

**Index Terms**—MARTE; Polychrony; CCSL; TimeSquare; clock calculus; simulation;

## I. INTRODUCTION

The Unified Modeling Language (UML) [1] has been widely adopted as a general purpose modeling language. As an extension of UML that aims at system modeling in the domain of real-time and embedded (RTE) systems, the UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [2] has been proposed and was recently adopted as an Object Management Group (OMG) specification. MARTE provides more precise semantics and appropriate concepts in the context of RTE systems than UML.

Time plays a very important role in RTE systems. However, UML only provides a simple (even simplistic) model of time. A richer and more adequate time model has been therefore proposed in MARTE. It is heavily inspired by the Tagged Signal Model [3], synchronous languages [4], and the Globally Asynchronous and Locally Synchronous (GALS) architecture. This time model enables to specify independent clocks, which can be progressively composed with a family of possible time evolutions. It is associated with the Clock Constraint Specification Language (CCSL) [2], [5], which is also defined in MARTE. CCSL supports both synchronous and asynchronous compositions, as well as specifications with nondeterminism. TimeSquare [6] is a software environment dedicated to the design and analysis of CCSL specifications. It is equipped with a constraint solver that enables the simulation for nondeterministic behavior.

CCSL is a specification language that gives a timed causality semantics to UML models. TimeSquare only supports interactive simulation and should rely on analysis-specific environments to provide advanced analysis features, such as identification of determinism or code generation. Behavior determinism with regard to time is very important for the design of embedded systems, particularly safety-critical systems. It also ensures the same system behavior in different contexts such as design, verification and implementation. It therefore simplifies the behavior verification and analysis. It is interesting to find other existing tools to complement this functionality for TimeSquare considering reduced cost and risk.

Polychrony [7] is an integrated development environment for the design of embedded reactive systems. The Polychrony toolset takes Signal as kernel design language. It provides a formal framework for the system modeling at a high level of abstraction, design validation at different levels, as well as simulations for deterministic specifications. The Signal language [8] is based on synchronized data-flow. The Signal formal model provides the capability to describe systems with several clocks (polychronous systems) as relational specifications. Relations are useful as partial specifications and as specifications of nondeterministic devices (e.g., a nondeterministic bus) or external processes (e.g., an unsafe car driver). Deterministic specifications allow code generation that enables simulation, analysis, validation and synthesis. The application domain of Polychrony includes safety-critical systems, such as automotive and avionics.

Simulations without consideration of determinism in the clock constraints is not enough for safety-critical systems. Hence, we propose an approach to address clock determinism analysis of CCSL specifications using existing techniques and tools of Polychrony. This approach is based on the hierarchization technique that allows the identification of determinism in the CCSL specifications. Afterwards, code is generated and simulation is carried out, compared to the nondeterministic one of TimeSquare. Two examples are presented in the paper. The first example is involved in the calculation of Easter in the calendar. The second one is an example of simplified flight warning system. The systems in the two examples are modeled using CCSL clock constraints and analyzed in the framework

of Polychrony to illustrate our proposed approach.

General introductions to CCSL and Polychrony are respectively given in Section II and III. Our main contribution on CCSL clock analysis, together with two examples, are presented in Section IV. Section V summarizes related work. Finally, conclusions are given in Section VI.

## II. THE MARTE TIME MODEL AND TIMESQUARE

### A. Time model in MARTE

In UML, the time notion is not clearly defined for the design of RTE systems. In comparison, MARTE presents time in a more precise and clear manner. Both discrete and dense times are handled in MARTE. Clocks, which can be chronometric or logical, are used to access time structure. A chronometric clock implicitly refers to physical time and a logical clock mainly addresses concrete instant ordering. The MARTE time model allows multiform/polychronous time modeling, which is inspired by synchronous languages [4].

We address logical clocks in this paper. A *clock* is a finite or infinite set of *instants*. A clock may represent a timed event and instants are its occurrences. A clock has a unit and the instants can have a label. These instants in a clock are totally ordered for discrete time clocks, thus they can be indexed by natural numbers. A time structure is composed of a set of clocks with the *precedence* relation between them. Precedence is a binary relation on clocks [9], and from this relation, we can derive the following new relations: coincidence, strict precedence, independence, and exclusion.

### B. CCSL

MARTE introduces a new stereotype of UML Constraint, through which a MARTE timed system can be specified. CCSL is used to express the clock constraints based on these constraint stereotypes. It is a non normative language annexed to MARTE (Annex C), and it is independent of any existing language. A comprehensive informal description of CCSL has previously been presented in [5] and a formal semantics for a kernel of CCSL can be found in [9]. A CCSL specification consists of clock relations and clock expressions. *alternatesWith* and *isPeriodicOn* are instances of clock relations, *delayedFor* and *sampledOn* are clock expressions. In this paper, only these four constraints are presented and illustrated with examples in Section IV.

### C. TimeSquare

TimeSquare is a software environment dedicated to modeling and analysis of timed systems specified with clock constraints using the CCSL language [6]. It is composed of a set of Eclipse plugins and has been integrated into the OpenEmbeDD platform [10]. TimeSquare has the following functionalities: interactive clock-related specifications; clock constraint checking; generation of a solution for clock constraints; visualization and exploration of the simulation results through waveforms.

## III. THE POLYCHRONY FRAMEWORK

### A. Signal and Polychrony

Polychrony is an integrated development environment based on the Signal language for design of safety-critical systems. Signal is a synchronous language, which is based on synchronized data-flow (flows + synchronization). Variables are called *signals* in the Signal language. Each signal (e.g.,  $x$ ) represents an infinite typed sequence, which is mapped onto the *logical time* indexed by natural numbers, i.e.,  $x$  is actually  $(x_\tau)_{\tau \in \mathbb{N}}$ . The symbol  $\perp$ , which represents the absence of the signal at certain instant on the logical time, expands the domain of signal. A signal has an associated clock indicating the set of instants where the signal is present. A *process* is considered as a program that is composed of a system of equations over signals and an interface.

Signal allows the specification of *multiclock/polychronous* systems, in which a process can be deactivated while other processes are still activated. Two kinds of operators are defined in Signal: monochronous and polychronous. The former operates on signals with the same clock, i.e., signals that are always present at the same time. The latter handles signals with different clocks. In addition, the Signal formal model allows partial and nondeterministic specifications. The model also supports a design methodology which goes from specification to implementation, from synchrony to asynchrony.

The Polychrony toolset is composed of the Signal batch compiler, a Graphical User Interface (GUI), and the Sigali tool. The compiler provides a set of functionalities, which include program transformations, optimizations, formal verification, separate compilation, code generation, simulation, temporal profiling, etc. The Sigali tool is used to build associated formal systems for formal verification and controller synthesis [20].

### B. Hierarchization

Several techniques enable static analysis, code generation, formal verification, code distribution, etc., in the compilation of Signal programs. In this paper, we concentrate on the hierarchization technique [11], which helps to achieve our objective of analyzing CCSL clock constraints. The main objective of hierarchization is to exploit clock relations based on the clock *inclusion* relation. This relation indicates that all the instants of a clock are included in another one. A tree structure is then built according to this exploitation. Only *Clock partition* and *tree fusion* are briefly presented here.

A clock  $C_2$  dominates a clock  $C_1$  (left part of Fig. 1) if the clock  $C_1$  is computed as a function of  $C_2$  (using Boolean sampling, clock intersection, union and difference operators) or as a function of clocks recursively dominated by  $C_2$ . Boolean sampling is the basic pattern of the hierarchy. Let  $V$  be a Boolean signal whose clock is specified by  $\hat{C}$ .  $\hat{C}$  is partitioned into  $[C]$  and  $[-C]$ .  $[C]$  indicates a subclock of  $V$  when the values of  $V$  are *true*, whereas  $[-C]$  indicates another complementary subclock of  $V$  when  $V$  is *false*. The subtree that expresses the relation of these three clocks can be built and it is illustrated by the right part of Fig. 1.

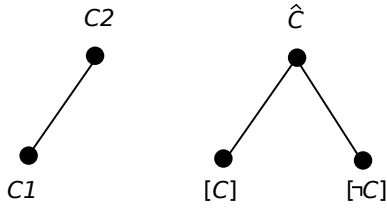


Fig. 1. An example of partition tree for clock  $C$ .

A Signal program is expressed by a set of clock trees similar to the ones in Fig. 1. Fusion of these trees should be carried out so that a concise yet global clock relation is exhibited. The tree fusion is also carried out according to clock inclusion. Fig. 2 presents an example of clock fusion.  $C3 = C1 < op > C2$  is an equation, where  $< op >$  indicates clock union, intersection or exclusion.  $C$  is the parent of both  $C1$  and  $C2$ . Hence these two trees can be fused because  $C$  can also be considered as the parent of  $C3$ .

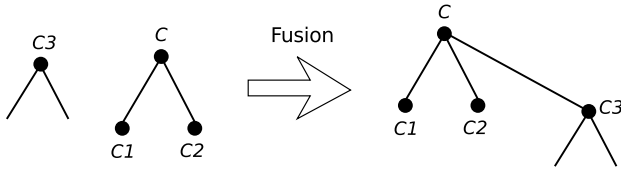


Fig. 2. An example of clock tree fusion.

### C. The endochrony property and code generation

The compilation of Signal programs partly relies on hierarchization, particularly the generation of control structure for the code. *Endochrony* is an important property for the compilation of Signal programs. An Endochronous Signal process can reconstruct synchronous clock relations from untimed but ordered input signals with the help of signal relations and states defined in the process. So it is possible to execute a Signal program even without knowing the absence status of the input signals. The application of tree fusion on all trees leads to two results, a clock hierarchy with one or multiple roots. The one root hierarchy implies the endochrony property in the Signal program. This program can be compiled into code in *if-then-else* structure according to the clock hierarchization. Fig. 3 illustrates the example of Fig. 2. In comparison, a multi-root hierarchy signifies nondeterminism in the system. As a result, code generation is not possible without adding complementary information.

```

if C then
  if C1 then ... endif
  if C2 then ... endif
  if C1 <op> C2 then ... endif
endif

```

Fig. 3. The control structure of code generated from the example in Fig. 2.

## IV. HIERARCHIZATION OF CCSL CLOCK CONSTRAINTS

This paper mainly addresses the integration of the Polychrony clock calculus into TimeSquare. The MARTE time model/CCSL aims at providing a general time model with regard to clock relations. However it is not yet supported by many tools until now. On the other hand, Polychrony consists of plenty of analysis tools for clock relations. Hence it is a promising approach to benefit from the various tools of Polychrony to enhance the analysis capacity of TimeSquare. In addition, behavior determinism with regard to time is a very important property for safe design of safety-critical systems. It simplifies design validation and also ensures the coherence between design and implementation. TimeSquare allows simulation for timed system specified in CCSL, however, it is not adequate for safety-critical systems.

Our approach has been proposed to satisfy previous demands. Polychronous clock analysis is necessary when code generation, in consideration of deterministic simulation, is expected from a timed system. Using the existing and sophisticated hierarchization technique of Polychrony helps to reduced design cost and risk. Nevertheless, the expressivity of the two languages, i.e., Signal and CCSL, is different as they are not faced with the same problems. The main differences between TimeSquare/CCSL and Polychrony/Signal are first summarized here:

- CCSL aims at providing a more generic time model than Signal. Both dense time and discrete time are supported in CCSL, whereas only logical time is allowed in Signal. Hence, it is necessary to map dense and discrete time of CCSL onto the logical time of Signal. For instance, the dense time is mapped onto the discrete time through sampling or discrete observation. Then, the discrete time is mapped onto the logical time in a natural way.
- CCSL allows the specification of clock relations with numerical properties. Some of them can be also specified in Signal. However, some numerical properties, e.g., duration, are not well supported by the code generation of Signal programs. On the other hand, Signal arithmetic operations on numbers are not supported in CCSL.
- Asynchronous clock constraints are more easily specified and addressed in CCSL than in Signal. TimeSquare provides a constraint solver that addresses these constraints in a nondeterministic way, thus it allows nondeterminism in the simulation. Signal also allows the specification of asynchronous clock constraints, whereas the Signal compiler refuses direct code generation for these non-deterministic constraints. Hence, a valid specification of TimeSquare is not always accepted by the Signal compiler for code generation.

Due to these differences as well as the objective of clock determinism analysis, we take advantage of the hierarchization technique. As the presented work uniquely addresses behavior with regard to time in a system, only clock expressions, Boolean expressions and CCSL logical clocks without consideration of values and numerical characteristics are involved

here, similar to the TimeSquare constraint solver. Typical CCSL clock relations and expressions are analyzed with the hierarchization technique separately, based on which a system of clock constraints is analyzed. The result of the analysis assists code generation with consideration of determinism, which is finally used for simulation in the framework of Polychrony.

### A. Hierarchization analysis

This work partly relies on the translation of typical CCSL relations into Signal [12]. In this paper, we focus on the hierarchization analysis of these CCSL clock relations, which helps to identify the potential determinism in the application.

1) **Clock relation *alternatesWith***: it indicates the alternate occurrences of instants in two clocks. More precisely, *A alternatesWith B* implies the first occurrence is *A*, then between two successive occurrences of *A*, there is one and only one occurrence of *B*. A concrete example is a reliable asynchronous communication: data sending is always before data receiving. Obviously, the two clocks *A* and *B* are not synchronous if there is no other relation specified between them. In this case they are only constrained by an asynchronous relationship. This relationship implies nondeterminism. Fig. 4 shows an example of *alternatesWith* between the two clocks *A* and *B*. Clocks *A* and *B* are in different trees in the hierarchization so they are placed in different triangles. Furthermore, the dotted line tagged with *alternatesWith* indicates a constraint relation. There are two forms of *alternatesWith*: weak form and strict form. As these two forms have the same hierarchization, they will not be detailed here.

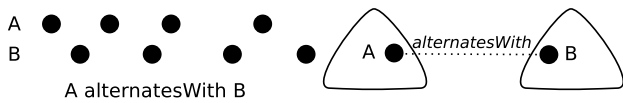


Fig. 4. An example and the hierarchization of *alternatesWith*.

2) **Clock relation *isPeriodicOn***: it can be used to build a subclock from a given clock. For instance, *B isPeriodicOn A period = p offset = o* builds a subclock *B* from the clock *A* according to a periodicity *p* and an offset *o*. The time index of *A* has a linear relation with that of *B*. The left part of Fig. 5 illustrates an example of this relation. As there is an explicit inclusion relation between *A* and *B*, hierarchization of *A* and *B* is comparably simple, as illustrated in Fig. 5. A special case of *isPeriodicOn* is when *p = 1* and *o = 0*, *A* and *B* are completely synchronous, i.e., *A = B*.

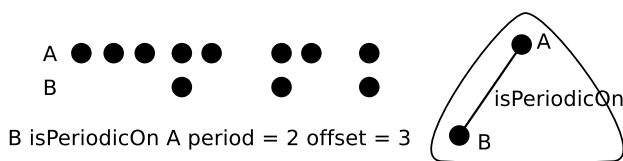


Fig. 5. An example and the hierarchization of *isPeriodicOn*.

In addition, a more general clock relation *filteredBy* has also been defined in CCSL to filter a clock and build a subclock by selecting some instants (not necessarily periodical). Using the clock relation *filteredBy* the relation *isPeriodicOn* can be written  $A = B \text{ filteredBy } 0^o(1.0^{p-1})$  (*o* indicates offset and *p* implies period), where  $0^o(1.0^{p-1})$  is the filter pattern based on the instants of a clock [9]. The number 1 indicates the instant selection and 0 indicates disregard. They are separated by “.”. The power of *o* and *p-1* means the repetition number of 0 or 1. () implies the infinite repetition of it inside pattern. Both *filteredBy* and *isPeriodicOn* have the same form in hierarchization.

3) **Clock relation *sampledOn***: it is based on sampling.  $C = B \text{ sampledOn } A$  indicates *C* is a subclock of *A* and each instant of *C* corresponds to an instant of *A* that closely follows an instant of *B*. There are two forms of the *sampledOn* relation: weak form and strict form, which defines the exact occurrence of *C* with regard to *A*. As only clock inclusion relation is involved in this paper, they will not be further distinguished as these two forms have the same hierarchization result. Fig. 6 presents an example of the *sampledOn* relation and the hierarchization result: *C* is a subclock of *A*; *A* and *B* are polychronous clocks.

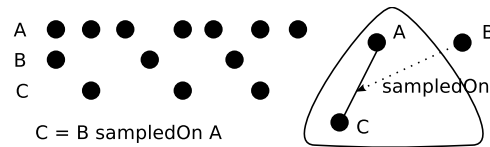


Fig. 6. An example and the hierarchization of *sampledOn*.

4) **Clock relation *delayedFor***: A watchdog can be modeled by the relation *delayedFor*. A watchdog is composed of a *starter*, a *delay* number and a *timer*. A *timeout* is emitted when the starter occurs after a period of the delay time according to the timer. If there is another starter that occurs during this period, the time counter will not be reset. Fig. 7 illustrates the relation *delayedFor* with an example and its hierarchization result. *A* is not synchronous with *B*, thus they are polychronous. Clock *C* (timeout) is a subclock of *A* (timer) but not *B* (starter).

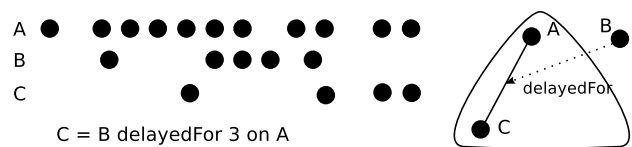


Fig. 7. An example and the hierarchization of *delayedFor*.

### B. Code generation and simulation

One of the main objectives of the hierarchization technique is to determine an endochronous clock system *E* from the analysis of an original Signal program *S*. An endochronous system implies a unique root node in the hierarchization tree.

This endochronous system ensures the deterministic scheduling of arriving events only according to the internal signal state and structure of  $S$ . In Polychrony,  $E$  can be used to generate code, for instance, the control structure of code in Fig. 3 is generated from the example in Fig. 2. Once the code is generated, it can be used for simulation, which is a deterministic one compared to that of TimeSquare. Traces in VCD format [13] can be generated for the visualization of simulation results through graphical VCD viewers [6].

However it is not always obvious to build an endochronous system from  $S$  as  $S$  can be polychronous, i.e., there are independent clocks. These clocks can be completely independent, i.e., no communication occurs between processes. These clocks can also have constraints between them, but no synchronous relationships specified between them. For instance, clock  $A$  and  $B$  in the relations *sampledOn* and *delayedFor*. In this case, it leads to nondeterminism in the system. In TimeSquare, the user can choose amongst a set of possible simulation policies (random, as soon as possible, priority-based) to select one solution out of the many possible ones. However, in Polychrony, code cannot be generated due to the nondeterminism in the specification. It therefore results in a Signal compilation issue.

There are several solutions available in Polychrony to obtain the deterministic behavior. One of the solutions relies on adding supplementary clocks to endochronize polychronous clocks, i.e., these clocks should be mapped onto the added supplementary clocks, through which the original clocks are endochronized. For instance, Fig. 8 illustrates a possible solution of endochronizing polychronous clocks. Clock  $A$  and  $B$  are two polychronous clocks.  $C$  is a supplementary clock that is added to endochronize  $A$  and  $B$ .  $C$  has two subclocks  $C1$  and  $C2$ . The endochronization of  $A$  and  $B$  can be achieved by synchronizing  $A$  with  $C1$  and  $B$  with  $C2$  respectively. A concrete example is presented here. Let  $A$  *alternatesWith*  $B$ . A Boolean signal  $C$  is a supplementary clock, which has alternate *true* and *false* as its values.  $C1$  and  $C2$  are subclocks of  $C$  indicating  $C$  is true and false respectively. Then the clock  $A$  is synchronized with  $C1$ , and  $B$  is synchronized with  $C2$ . This approach endows the endochrony property to the system so code generation in the framework of Polychrony is possible.

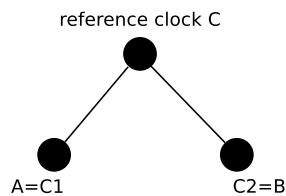


Fig. 8. Examples of endochronizing polychronous clocks.

### C. An example of Easter days

An example of Easter calculation that has been presented in [14] is taken here to illustrate the usage of hierarchization. *Easter Day is the first Sunday after the 14th day of the lunar month that falls on or after March 21st (nominally the day of*

*the vernal equinox*). In comparison to [14], which presented the example with UML and MARTE models, only its CCSL specification is shown here (starting from the 1st of March 2008).

```

sunday isPeriodicOn days period 7 offset 1;
newMoon = days filteredBy 06(1.029);
vEquinox = days filteredBy 020(1.0365);
fullMoon = newMoon delayedFor 14 on days;
easterMoon = vEquinox weakly sampledOn fullMoon;
easter = easterMoon strictly sampledOn sunday;
  
```

Fig. 9. The Easter example specified in CCSL.

Fig. 9 illustrates the CCSL specification of the Easter example. Important dates, such as day, Sunday, new moon, full moon, vernal equinox, and Easter are modeled as pure clocks (without definition of values). These days are defined as days, sunday, newMoon, fullMoon, vEquinox, and easter in CCSL respectively. According to the calendar, the relations of these dates are modeled by clock constraints in CCSL (Fig. 9).

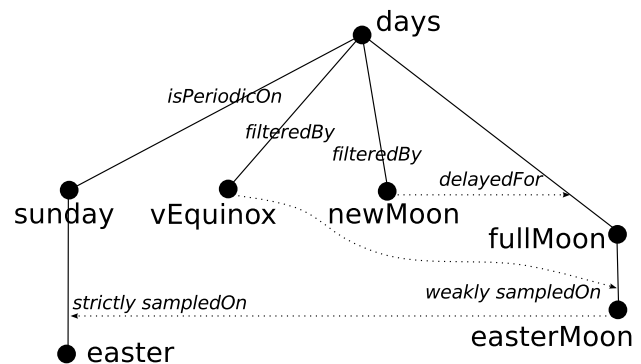


Fig. 10. Hierarchization result of the Easter example.

Hierarchization of the Easter example is illustrated in Fig. 10. It implies an endochronous system, i.e., all the clocks have a unique root days. The Signal compiler succeeded in code generation when the CCSL specification was translated into Signal. The generated code in C has a similar control structure as presented in 3 and implies deterministic behavior.

### D. Another example of a flight warning system

Another example of a simplified flight warning system is also considered here. This system has been proposed by the Aerospatiale Company (France) for Airbus A340 aircraft and was presented as a case study in [15]. The system is used to decide when and how to emit warning signals in case of an anomaly. It is composed of two processes: an *alarm manager* and an *alarm notifier*. An alarm manager receives an alarm  $a$  in the system, and  $a$  is changed to a confirmed alarm after a given period of time or removed according to its present or absent status. Confirmed alarms are sent to the

alarm notifier. Once the alarm notifier receives the alarm, it emits a corresponding warning signal  $w$ .

Fig. 11 illustrates the CCSL specification of this example. The alarm manager and alarm notifier are concurrent and they have different clocks:  $\text{clk\_am}$  and  $\text{clk\_an}$ . The confirmed signals  $a1$  and  $w1$  are based on the clocks of  $\text{clk\_am}$  and  $\text{clk\_an}$  respectively. The communication between the alarm manager and notifier is asynchronous: the data sending and receiving is modeled by an `alternatesWith` relation.

```

a1 = clk_am filteredBy 0k(1.0n);
a = a1;
w1 alternatesWith a1;
w1 = clk_an filteredBy 0k(1.0n);
w = w1;

```

Fig. 11. The flight warning system example specified in CCSL.

Fig. 12 shows the hierarchization result, which indicates the system is polychronous, so code generation is not direct. Supplementary clocks should be added to endochronize  $\text{clk\_am}$  and  $\text{clk\_an}$  for the code generation (not for distribution), similar to the approach presented in Fig. 8. For distributed code generation, [16] proposed to use *clock-less* memory so that the memorized signal value can be available whenever it is required without consideration of their different clocks.

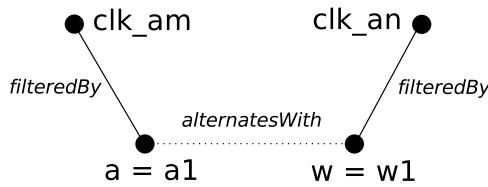


Fig. 12. Hierarchization result of the flight warning system example.

### E. Discussions

The objective of our proposed approach is to identify behavior determinism with regard to time in systems, particularly those who are specified using the MARTE CCSL. The MARTE time model implies a general time model, which is heavily inspired by the Tagged Signal Model, synchronous languages, and the Globally Asynchronous and Locally Synchronous (GALS) architecture. Based on this time model, the presented work has a broad application domain, and it does not aim at certain specific technologies.

Our work is considered as a continuation of [12] with regard to behavior determinism analysis. In [12], the translation of certain main CCSL clock relations has been presented. Formal semantics of these relations were exhibited with the help of this translation, and comparisons of relevant CCSL and Signal operators are also given. However, systematic clock analysis, particularly time-related determinism analysis, is absent. In this paper, this aspect has been addressed. The

advantage of our approach is that it enables to identify the behavior determinism with regard to time in a system. Thus a deterministic simulation for a safety-critical system is possible.

A polychronous system, which is not endochronous, implies nondeterministic behavior with regard to time. This can be simulated by TimeSquare. TimeSquare does not reject deterministic simulation. The TimeSquare simulation implies a nondeterministic choice of a solution from solution space. In comparison, Polychrony only accepts deterministic simulation, hence a polychronous system should be firstly endochronized as presented previously. The endochronization implies a deterministic choice of a specific solution from the solution space. However it is not always possible to find appropriate supplementary clocks for endochronization in consideration of the determinism requirement. In this case, clock constraints of the system are expected to be improved.

## V. RELATED WORK

*Affine clock systems* are considered as a Signal clock refinement [17]. They are based on the affine relation of time indexes of signals, from which the synchronizability of these signals is analyzed. Periodic clocks can be easily specified in CCSL, however they need to be specified with the help of counters in Signal. Moreover their synchronizability is difficult to be analyzed. Affine clocks allow the specification of periodic clocks in an efficient way, and analyze the synchronizability between periodic clocks. Hierarchization only takes clock inclusion relation in consideration now. But it is promising to consider affine relation and clock synchronizability.

As endochrony is not well situated to address issues of compositionality [18], asynchronous clock relations, etc., weakly endochronous systems [19] have been proposed for the Globally Asynchronous Locally Synchronous (GALS) architecture. They aim at meeting the requirements of building deterministic asynchronous implementations from polychronous specifications. Weak endochrony enables identical execution results of synchronous specifications in any asynchronous environment. So weak endochrony is a good complement to address asynchronous composition while preserving deterministic system behavior. Therefore, it is a complementary technique to hierarchization for the analysis of asynchronous clock relations.

Hierarchization is a powerful mechanism to determine the endochrony property of a given system specified in CCSL. Code generation for polychronous systems has always been a difficult problem. Several solutions have been presented previously, however it is still interesting to explore the existing work on code distribution in the framework of GALS via Polychrony. For instance, consider each endochronous sub system and the overall polychronous system as local systems and the global system in the GALS architecture respectively. These endochronous sub systems are interconnected with asynchronous communications. Thus, the presented work makes its extensions in the framework of GALS possible, particularly weak endochrony [19].

In addition to the clock calculus presented previously, controller synthesis [20] is another approach that enables to

restrain system behavior with regard to certain expected properties, such as invariance and reachability. The controllers to be synthesized in the system ensure a deterministic scheduling of system compared to the construction of a deterministic system a posteriori via clock calculus. In Polychrony, Sigali is an associated tool to achieve controller synthesis.

## VI. CONCLUSIONS

This paper presents an approach to identify the determinism in a timed system, specified with the MARTE Clock Constraint Specification Language (CCSL). This approach is based on hierarchization, the clock calculus provided by Polychrony. Deterministic behavior identification via clock analysis is a desired feature of TimeSquare, a software environment dedicated to the simulation of MARTE timed systems. Polychrony and its hierarchization technique is a good complement to TimeSquare.

Hierarchization of typical CCSL clock relations is first presented, which can be recursively applied on a system of clock constraints. Two kinds of systems can be identified through hierarchization: endochronous and polychronous. An endochronous system implies synchronization of all clocks is completely defined, hence it has a deterministic behavior. Whereas a polychronous system does not ensure the synchronization of all clocks, so it remains nondeterministic, which complicates the system design, validation, implementation, etc. In order to make them deterministic, a specific solution to endochronize clocks, i.e., adding synchronization clocks, has been presented. In addition, weak endochronous systems, affine clock systems, and controller synthesis are briefly discussed as they are related to our work and are also good candidates to improve our work to some extent.

## REFERENCES

- [1] Object Management Group (OMG), “UML 2.2 Superstructure and Infrastructure,” <http://www.omg.org/spec/UML/2.2>, February 2009, formal/2009-02-04.
- [2] —, “Modeling and Analysis of Real-time and Embedded systems (MARTE), v1.0,” <http://www.omgarte.org/Documents/Specifications/08-06-09.pdf>, November 2009, document number: formal/2009-11-02.
- [3] E. Lee and A. Sangiovanni-Vincentelli, “A Framework for Comparing Models of Computation,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1217–1229, December 1998.
- [4] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, “The Synchronous Languages Twelve Years Later,” *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, January 2003.
- [5] C. André, F. Mallet, and R. D. Simone, “Modeling Time(s),” in *ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS/UML’07)*, ser. LNCS 4735. TN, USA: Springer, October 2007, pp. 559–573.
- [6] INRIA AOSTE team, “TimeSquare,” [http://www-sop.inria.fr/aoste/dev/time\\_square](http://www-sop.inria.fr/aoste/dev/time_square), 2009.
- [7] INRIA ESPRESSO team, “Polychrony V4.15.10,” <http://www.irisa.fr/espresso/Polychrony>, December 2008.
- [8] L. Besnard, T. Gautier, P. Le Guernic, and J.-P. Talpin, “Compilation of polychronous data flow equations,” in *Correct-by-Construction Embedded Software Synthesis: Formal Frameworks, Methodologies, and Tools*, S. Shukla and J.-P. Talpin, Eds. Springer, 2010.
- [9] C. André, “Syntax and Semantics of the Clock Constraint Specification Language (CCSL),” INRIA, Research Report RR-6925, 2009, 37 pages. [Online]. Available: <http://hal.inria.fr/inria-00384077/en/>

- [10] “OpenEmbeDD,” <http://www.openembedd.org>, 2009.
- [11] P. Amagbegnon, L. Besnard, and P. Le Guernic, “Implementation of the Data-flow Synchronous Language Signal,” in *Proceedings of the ACM Symposium on Programming Languages Design and Implementation (PLDI’95)*. ACM, 1995, pp. 163–173.
- [12] F. Mallet and C. André, “On the Semantics of UML/MARTE Clock Constraints,” in *12th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2009)*. Tokyo, Japan: IEEE Computer Society, March 2009, pp. 305–312.
- [13] “IEEE Standard for Verilog Hardware Description Language,” 2006, IEEE Std 1364 -2005.
- [14] F. Mallet and C. André, “UML/MARTE CCSL, Signal and Petri nets,” INRIA, Research Report RR-6545, May 2008, 23 pages.
- [15] N. Lopez, M. Simonot, and V. V. Donzeau-Gouge, “A Methodological Process for the Design of a Large System: Two Industrial Case-studies,” *FMICS’02, Electronic Notes in Theoretical Computer Science*, vol. 66, no. 2, pp. 84–103, 2002.
- [16] A. Gamatié and T. Gautier, “The Signal Synchronous Multi-Clock Approach to the Design of Distributed Embedded Systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. 1, pp. 1045–9219, 2009.
- [17] I. Smarandache, T. Gautier, and P. Le Guernic, “Validation of Mixed Signal-Alpha Real-Time Systems through Affine Calculus on Clock Synchronisation Constraints,” in *World Congress on Formal Methods (FM’99)*, ser. LNCS, vol. 1709. Springer, 1999, pp. 1364–1383.
- [18] A. Benveniste, B. Caillaud, and P. L. Guernic, “Compositionality in dataflow synchronous languages: Specification and distributed code generation,” vol. 163, pp. 125–171, 2000.
- [19] D. Potop-Butucaru, B. Caillaud, and A. Benveniste, “Concurrency in synchronous systems,” in *Formal Methods in System Design*, vol. 28, March 2006, pp. 111–130.
- [20] H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic, “Synthesis of Discrete-Event Controllers based on the Signal Environment,” *Discrete Event Dynamic System: Theory and Applications*, vol. 10, no. 4, pp. 325–346, October 2000.