Clocks vs. Instants Relations: Verifying CCSL Time Constraints in UML/MARTE Models

Judith Peters[1] Nils Przigoda[2,3] Robert Wille[4,3] Rolf Drechsler[2,3]
[1]Department of Satellite Ground Systems, OHB System AG, 28359 Bremen, Germany
[2]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[3]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[4]Institute for Integrated Circuits, Johannes Kepler University Linz, 4040 Linz, Austria
judith.peters@ohb.de {przigoda,drechsler}@informatik.uni-bremen.de robert.wille@jku.at

***Abstract*—The specification of non-functional requirements, e. g., on timing forms an essential part of modern system design. Modeling languages such as MARTE/CCSL provide dedicated description means enabling engineers to formally define the ticking of the clocks to be implemented in terms of clock constraints and the actually intended timing behavior in terms of instant relations. But thus far, instant relations have only been utilized in order to monitor the correct execution of the clock constraints. In this work, we propose a methodology which, for the first time, verifies clock constraints against the given instant relations. To this end, the timing behavior is represented in terms of an automaton followed by its verification through satisfiability solvers. A case study illustrates the application of the proposed methodology.**

I. Introduction

The design of today's computing devices (including embedded and cyber-physical systems) is one of the most complex problems *Electronic Design Automation* (EDA) is currently facing. In order to handle the ever increasing complexity, designers constantly introduce higher levels of abstraction. While the design evolved from the *Register Transfer Level* (RTL) to the *Electronic System Level* (ESL) in the past, new trends include the exploitation of modeling languages as a bridge between the initial (textbook) specification as well as a first (formal) model. This eventually led to the design at the *Formal Specification Level* (FSL, [1]). Here, modeling languages such as the *Systems Modeling Language* (SysML, [2]) or the UML profile *Modeling and Analysis of Real-time and Embedded systems* (MARTE, [3]) find great attention.

However, while they allow for a precise (i. e., formal) description of the functional behavior of the system to be implemented, the specification of non-functional requirements, e. g., of timing forms another essential part of today's designs flows. In fact, even a system which is functionally correct may fail in application or may be rejected by the user, if it does not realize the expected timing behavior. Particularly safety-critical systems heavily rely on such requirements. As an example, an airbag is not implemented correctly if it is just released, but only if it is released at the exact moment.

In order to address these needs, modeling languages have been enriched with description means focusing on the specification of timing constraints. As one of the most prominent examples, MARTE offers the *Clock Constraint Specification Language* (CCSL, [3]) for this purpose. CCSL equips designers and engineers with powerful description means in order to precisely specify and, afterwards, implement the timing behavior of the system to be implemented. Moreover, due to its formal nature also efficient tool support is available that assists them in these tasks. In fact, approaches such as [4], [5], [6] as well as [7] allow for (automatically) realizing the given CCSL description, e. g., in SystemC and proving whether this realization indeed satisfies the requirements as given in CCSL, respectively.

Besides that, CCSL enables designers and engineers to check the original timing specification in early stages of the design flow. To this end, approaches such as [8], [9], [10] have been proposed, which allow for verifying whether a given CCSL description is consistent, i. e., whether it does not include contradictory statements and indeed allows for the execution of an (arbitrary) timing behavior. While this prevents designers and engineers from implementing self-contradictory timing requirements, it however does not guarantee that the requirements indeed specify the actually desired behavior. This is unfortunate particularly considering that CCSL already provides all description means necessary for this purpose: In fact, CCSL specifications are usually composed of clock constraints (describing the ticking of the clocks to be implemented) and instant relations (which can be interpreted as a description of the actually intended timing behavior). But thus far, instant relations have only been utilized in order to monitor the correct execution of the clock constraints after implementation [7].

In this work, we are proposing a solution which exploits this potential of the CCSL prior to the implementation. Instead of using instant relations for monitoring purposes only, we consider them as properties to be satisfied by the clock constraints. Based on this interpretation, a methodology is introduced which is capable of verifying clock constraints against the given instant relations.

To this end, the timing behavior is represented in terms of an automaton followed by its verification through satisfiability solvers. By this, a verification scheme becomes available which is capable of checking whether the timing behavior of a system has been specified in CCSL as intended. A case study illustrates the application of the proposed methodology.

```
1  ClockConstraintSystem sensors {
2   Clock minClock;
3   Clock sensor1;
4   Clock sensor2;
5   Clock echo is sensor1 delayedBy 1;
6
7   sensor2 # echo;
8   sensor1 isPeriodicOn minClock period 1.0;
9   sensor2 isPeriodicOn minClock period 1.0;
10 }
```

Fig. 1. A CCSL clock specification

In the following, the proposed solution is described as follows: Section II reviews the basics of CCSL, before the main idea of our approach is motivated by means of an example in Section III. Afterwards, Section IV introduces the proposed methodology in detail before its applicability is illustrated and discussed in Section V. Finally, the paper is concluded in Section VI.

II. The Clock Constraint Specification Language

Within the MARTE profile, a language dedicated to the description of timing issues is provided: the *Clock Constraint Specification Language* (CCSL, [3]). Central part of the underlying time definition are *instants*, i. e., moments in the raw, unordered time, defined by clock ticks. The *clock* is an instrument to access a set of instants [11]:

Definition 1. *A clock $\langle \mathcal{I}, \prec, \mathcal{D}, \lambda, u \rangle$ consists of a set of instants $\mathcal{I}$, which owns a quasi-order relation $\prec$, a set of labels for the instants $\mathcal{D}$, a labeling function λ, and a unit u for the clock ticks. A finite clock has a finite number of ticks. If no ticks are left, the clock is empty.*

Example 1. *Consider a system with two sensors triggered by the clocks `sensor1` and `sensor2`. Both sensors have to reply in every second step with respect to a third (minimal) clock `minClock`. From its second reply on, `sensor1` causes an echo interfering with the signal of `sensor2`. Hence, this echo and the reply of `sensor2` are not allowed to occur coincidentally. This is described in the CCSL specification in Fig. 1.*

First, the clocks are defined (lines 2–5). Clock `echo` in line 5 is defined as a subclock of `sensor1`. Both tick together, but `echo` starts one tick after `sensor1`. Afterwards, the relations between the clocks are restricted. Line 7 prohibits that `echo` and `sensor2` tick at the same time. Line 8 and Line 9 define the periodicity of `sensor1` and `sensor2`, respectively, to `minClock`.

Clocks are used to define sets of uniformed instants. Moreover, special instants satisfying certain conditions can additionally be defined using CCSL statements. These specific instants can be used, e. g., to trigger certain events in the system. More precisely, CCSL allows for defining instants of a clock c in categories such as:

```
sensor1(i) precedes sensor2(i);       1
sensor2(i) precedes sensor1(i+1);     2
                                      3
sensor2(k) precedes sensor1(k);       4
sensor1(k) precedes sensor2(k+1);     5
```

Fig. 2. CCSL specification of instants

- *Unspecific.* An arbitrary instant of the clock (e. g., `Instant i1 is myClock`),
- *Fixed.* A fixed instant of the clock (e. g., `myClock(4) precedes myOtherClock(8)`),
- *Relative.* A relative instant of the clock (e. g., `myClock(i) precedes myOtherClock(i-7)`), or
- *Conditional.* A conditional instant of the clock (e. g., `Instant i1 is myClock suchThat k>14`).

In general, the clocks these instants form can either refer to a constant physical (and possibly dense) time or to logical events, which not necessarily follow constant timing rules, but can occur with varying periods in physical time. In our scenario, the target system (i. e., SystemC) is discrete and, because of that, can only simulate logical time. The processor clock itself is a logical clock counting cycles. Hence, all times derived and simulated in this context are logical.

If there is more than one clock, all clocks and their respective instants can be structured through *instant relations* [11]:

Definition 2. *Given a set of clocks $\mathcal{C}$, a binary* instant relation *$\preccurlyeq$ (called precedence) can be defined stating that one instant occurs before or coincidently with the other. From $\preccurlyeq$, three further instant relations can be derived, namely*

- *precedence (denoted by $\preccurlyeq$), i. e., one of the clock ticks takes place before or coincidently with the other tick (e. g., i_1 `precedes` i_2),*
- *coincidence (denoted by $\equiv \triangleq \preccurlyeq \cap \succcurlyeq$), i. e., two clock ticks take place coincidently (e. g., i_1 `coincidentWith` i_2), and*
- *strict precedence (denoted by $\prec \triangleq \preccurlyeq \setminus \equiv$), i. e., one of the clock ticks takes place strictly before and not coincidently with the other tick (e. g., i_1 `strictly precedes` i_2).*

Example 2. *CCSL can be used to describe complex temporal relations. As an example, consider the clocking as already described in Example 1. Based on this, more precise statements about the instants of the clocks are given in Fig. 2. Lines 1–2 denote, that the i^{th} tick of `sensor1` is followed by or coincident with the i^{th} tick of `sensor2` and is again followed by or coincident with the $(i+1)^{th}$ tick of `sensor1`. In the CCSL statements, the same is stated in lines 4–5 for*

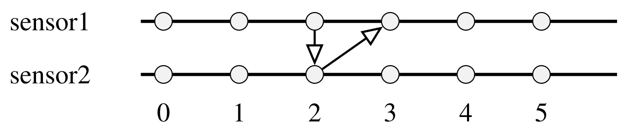


Fig. 3. Instant relations with two precedes statements

the k^{th} and $(k+1)^{th}$ tick of `sensor2` *as well as the k^{th} tick of* `sensor1`*. Fig. 3 illustrates the relations. Obviously, all relations could only be satisfied, if* `sensor1` *and* `sensor2` *tick coincidently.*

Instant relations only affect the instants to which they are referring to, not the clocking behavior itself. Because of that, no behavior can be derived from the instant relations and enforcing instants to appear is not possible. Consequently, the clock constraints represent the actual clocking behavior while instant relations provide a more detailed description of the intended timing behavior. Thus, instead of generating behavior, instants are thus far applied by some approaches to monitor the timing behavior and to report unwanted behavior (e. g., a meltdown instant) [7]. Respectively, from the clock constraints indeed the actual behavior can be derived as described, e. g., in [7], [10].

III. Motivation and Proposed Idea

In this section, we briefly discuss and illustrate the current exploitation of CCSL for verification purposes. Afterwards, we discuss how this state-of-the-art can significantly be improved. This eventually motivates the verification methodology proposed in the work. To this end, the following example is considered:

Example 3. *A simple satellite application for space systems shall be designed. The satellite shall take photos of the earth and the sun. In between, it has to ensure that its orbit is still correct, i. e., it has to check (and, if necessary, correct) its height. As it has only one camera, it shall alternately take photos of the sun and the earth. The height control is a complex task leaving not enough processor performance for image processing, thus, the satellite can not take photos and control its height at the same time.*

As reviewed in the previous section, CCSL provides description means to describe the ticking of the clocks to be implemented (in terms of clock constraints). Using these notations, the clock behavior of this system can be specified:

Example 4. *Fig. 4 shows a possible specification of the clocks to be used in order to realize the timing of the satellite application. The clocks* `photo_earth` *and* `photo_sun` *specified in line 2 and line 3 trigger the camera to take a photo of the earth and the sun, respectively. The constraints in line 6 and in line 7 respectively ensure that these clocks do not tick at the same time and indeed alternate between each other. Checking height is triggered by clock* `check_height` *(line 4) which must not tick concurrently with the other two clocks as restricted by the last two constraints in lines 8/9.*

```
ClockConstraintSystem satellite {
  Clock photo_earth;
  Clock photo_sun;
  Clock check_height;

  photo_earth # photo_sun;
  photo_earth alternatesWith photo_sun;
  photo_earth # check_height;
  photo_sun # check_height;
}
```

Fig. 4. CCSL description of a satellite

Besides that, CCSL allows to describe certain timing behavior which is intended to be executed on the realized system. To this end, detailed constraints over single instants can be specified in terms of instant relations:

Example 5. *In order to further refine the specification of the satellite application, instant relations as depicted in Fig. 5 are added. They state that between a tick of* `photo_sun` *and the consecutive tick of* `photo_earth` *the height shall be checked, i e. clock* `check_height` *shall tick. This is expressed by two strict precedence instant relations. The identifier for the instants are the indices i and j. The two consecutive instants of* `photo_sun` *and* `photo_earth` *are the i^{th} ticks of their clocks, while the* `control_height` *tick in between is the j^{th} tick of its clock.*

Eventually, a CCSL description results which, using, e. g., the approaches from [8], [9], [10], can be checked for consistency. However, whether the intended behavior as specified by the instant relations from Fig. 5 can indeed be realized considering the clock constraints from Fig. 4 is not covered by these verification methods. Instead, the state-of-the-art methodology continues with an implementation of the timing constraints (e. g., using code-generation approaches such as [7]). As this might result in the creation of monitors (as, e. g., suggested in [7]), corresponding checks can be conducted after the implementation has been finalized.

Obviously, this causes a significant drawback: Checking the intended timing behavior is considered rather late in the design process, namely not until a fully-fledged implementation is available. If it turns out that the intended behavior is not possible, re-spins of the implementation or even the CCSL specification have to be conducted. This leads to long debugging loops and poses a serious threat to time-to-market constraints.

Motivated by this, we are proposing an alternative methodology which aims for verifying at the CCSL level, i. e., prior to implementation, whether the given descriptions allow for the intended timing behavior. To this end, we apply a slightly

```
photo_sun(i) strictly precedes
  check_height(j);
check_height(j) strictly precedes
  photo_earth(i);
```

Fig. 5. CCSL instant relations

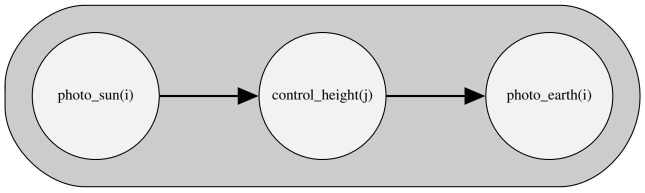


Fig. 6. An instant relation group

different interpretation of the CCSL description means. Instead of considering instant relations as behavior to be monitored, we apply them as *properties* to be satisfied by the clock constraints. Based on this, we propose a verification methodology whose general idea rests on three main steps:

1) Formulating the properties to be checked according to the instant relations
2) Deriving a symbolic representation of all possible ticking behavior of the clocks according to the clock constraints
3) Checking whether the resulting symbolic representation indeed satisfies the properties derived from the instant relations

In the remainder of this work, the implementation and application of this general idea is described and discussed, respectively.

IV. Implementation

In the following, we describe the three steps to be conducted in order to verify instant relations against clock constraints in detail. All steps are illustrated by means of the satellite example from above.

A. Formulating the Properties

Single instant relations already provide a formal description of timing behavior to be verified. However, as they may be connected to each other (cf. Example 5), a sole consideration may not provide a sufficient verification objective or property to be verified. In fact, an instant relation is only satisfied, if all other relations connected to it are also satisfied. Hence, a property to be checked has to be composed of a group of instant relations. To this end, we use the notation of an instant group defined as follows:

Definition 3. *Let i be an instant and $\oplus \in \{\equiv, \prec, \preccurlyeq\}$ an instant relation symbol. Let further $(i_j \oplus_k i_l)$ be an instant relation and $\mathfrak{I}$ the set of all instant relations. Then, an* instant group *$\mathcal{I}$ is a set of instant relations such that*

$$\forall\,(i_1 \oplus_1 i_2) \in \mathcal{I} : \exists\,(i_3 \oplus_2 i_4) \in \mathcal{I} : \{i_1, i_2\} \cap \{i_3, i_4\} \neq \emptyset.$$

An instant group is maximal*, iff*

$$\forall\,(i_1 \oplus_1 i_2) \in \mathcal{I} : \nexists\,(i_3 \oplus_2 i_4) \in \mathfrak{I} : \\ (\{i_1, i_2\} \cap \{i_3, i_4\} \neq \emptyset) \wedge (i_3 \oplus_2 i_4) \notin \mathcal{I}$$

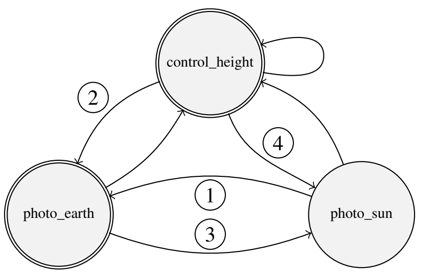
Example 6. *Consider again the instant relations from Fig. 5. Both instant relations refer the same instant, `control_height`, one on the left and one on the right side of the relation. Thus, both relations are connected by the instant `control_height`, and finally form one group. Eventually, this results in an instant group as shown in Fig. 6 and, hence, a property to be verified.*

In this definition, the maximality of the instant group is crucial. Each instant in a group of instants is defined with respect to all other instants. This means, for the occurence of one instant the occurence of all other instants connected to it is a mandatory requirement. Thus, all connected instants have to be considered altogether, since missing one means missing the whole group. As a result, the maximality of a group guarantees to consider all necessary instants.

B. Deriving a Symbolic Representation

Next, a symbolic representation of all possible clock tickings is required. To this end, an automaton representation as proposed in [10] is utilized. Here, all possible clock ticks which may occur in a certain time step are represented in terms of states. The starting states are hereby all states which do not contain clocks that are dependent on other clocks ticking earlier. Transitions allow for moving from one state to another and, hence, describe the valid sequences of ticks according to the clock constraints. To this end, transitions may have guard conditions over global variables (such as counters for periodic behaviors) which state whether a transition may be taken or not. The values of these global variables may be changed during a transition by means of update functions. In general, the states describe how clocks can be grouped together for ticking while the transitions describe how they are related to each other regarding chronological or external constraints. The concepts of the resulting automaton are illustrated by means of the following example.

Example 7. *Consider again the clock constraints from Fig. 4. The corresponding automaton representation is provided in Fig. 7. As no clock can tick together with another clock, only three states have to be considered (one in which each clock is allowed to tick). The states `control_height` and `photo_earth` are initial states—the third one cannot be an initial state because of the `alternatesWith`-statement requiring `photo_earth` ticking prior to `photo_sun`. The guard and update conditions consider additional Boolean variables b_{alt} (denoting who ticks next) and f (denoting if this is the initial alternation) which direct the alternation between `photo_earth` as well as `photo_sun` and are initially set to `true`. This ensures that the clock `photo_earth` ticks first*



$V_0 = (b_{alt} = true, f = true)$
① guard: b_{alt} =`true`
update: $b_{alt} \leftarrow$`false`
② guard: b_{alt} =`true`
update: $b_{alt} \leftarrow$`false`
③ guard: (b_{alt} =`false`)$\vee$(f =`true`)
update: ($b_{alt} \leftarrow$`true`)$\wedge$($f \leftarrow$`false`)
④ guard: (b_{alt} =`false`)$\vee$(f =`true`)
update: ($b_{alt} \leftarrow$`true`)$\wedge$($f \leftarrow$`false`)

Fig. 7. Automaton representing the CCSL

and, afterwards, the states `photo_earth` and `photo_sun` are only reached if the respectively other clock has ticked before.

C. Checking the Properties

Having the automaton, all clock ticking behavior which is possible according to the clock constraints is available in a symbolic fashion. Now, it has to be verified whether this ticking behavior indeed allows for the execution of a sequence of clock ticks which satisfies the property given by the instant group. In a naive fashion, this can be conducted, e. g., by enumeration. However, we propose a solution which exploits the computational power of satisfiability solvers such as [12]. To this end, we represent the considered problem as an instance of the satisfiability problem: "Is it possible to execute a sequence defined by the instant group on the automaton derived from the clock constraints?"

1) Creating the Instance: In order to formulate the corresponding instance, the execution of a sequence of clock ticks is symbolically considered. Each step in this sequence is denoted as a (symbolic) *simulation step* in the remainder of the section. All possible clocking ticks are encoded in terms of variables to be assigned by the satisfiability solver. More precisely, for each clock $c \in \mathcal{C}$, the following variables are introduced and added to the sequence for every simulation step:

- A Boolean variable t_c representing whether c does tick ($t_c = 1$) or does not tick ($t_c = 0$) in the current step.
- An integer variable ind_c representing the index of the current tick of the clock c. The variable is initially undefined and set to 0 when c ticks first. Afterwards, its value is increased by 1 with any further tick.

Additionally, another integer variable *step* is introduced which serves as counter for the respective (symbolic) simulation steps. For the formulation, we assume that a maximum number $step_{max}$ of simulation steps to be considered is available.

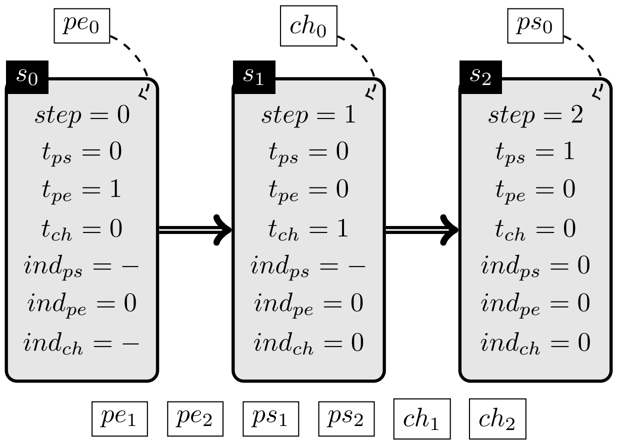


Fig. 8. Additional variables for instant checking

Finally, constraints are added to enforce the satisfiability solver to choose the variable assignments in the consecutive simulation steps according to the behavior of the automaton. This yields an instance representing all possible ticking sequences of length $step_{max}$ which is allowed by the clock constraints.

Example 8. *Consider again the clock constraints from Fig. 4 and the instant group from Fig. 5. In Fig. 8, a possible simulation sequence including the introduced variables is depicted. The assignment represents the simulation of the following clock ticks:*

$$\{\texttt{photo\_earth}\} \rightarrow \{\texttt{control\_height}\} \rightarrow \{\texttt{photo\_sun}\}.$$

The variable `step` counts the simulation steps and, thus, increases with each step. For each step, the assignments to t_c, $c \in \{$`ps`, `ch`, `pe`$\}$ (representing the clocks `photo_sun`, `control_height`, `photo_earth`), state the ticking clocks. For example, in the first step, `photo_earth` ticks, hence t_{pe} is assigned 1, while t_{ps} and t_{ch} are assigned 0. Correspondingly, the index variable for clock `photo_earth`, i. e., ind_{pe} is initialized with 0.

Note that, for illustration purposes, the example above as shown in Fig. 8 considers particular assignments. However, in order to symbolically represent all possible sequences of simulation steps, the created satisfiability instance is composed of unassigned variables only. The precise assignment has to be determined by the satisfiability solver.

2) Adding the Properties: Simply passing the satisfiability instance created thus far to a satisfiability solver would simply yield an assignment representing an arbitrary sequence of simulation steps possible by the clock constraints. However, we are interested in a dedicated sequence which shows the executability of a considered instant group. Hence, additional constraints are added to the satisfiability instance enforcing the solver to choose assignments in this regard.

The instant relation groups refer instants with certain indices. This means, we need to extract the corresponding state number of the referred instants and to check whether the ordering of the obtained state numbers follows the constraints from the instant relations.

To extract the state number for a certain index, for all clocks and all $step_{max}$ simulation steps, an Integer variable c_i with $0 \leq i \leq step_{max}$ is added for each $c \in \mathcal{C}$ denoting the simulation step number of the i^{th} tick of c. To model the corresponding i^{th} ticks, special constraints are added to the SAT solver. The considered property extracts the variables c_i from the encoded instance and connects them by precedence or coincidence statements. Hence, the connection between the corresponding c_i is restricted using simple comparison-statements to model precedence ($\leq$), strict precedence ($<$), or coincidence ($=$).

The final verification task is then to check the following statement:

$$\exists\, k \in 0 \leq k \leq step_{max} : [\![\mathcal{I}]\!]_k,$$

where $[\![\mathcal{I}]\!]_k$ is the evaluation of the instant relation group concerning the given index k as the index variable from the CCSL specification[1], e. g., $(A_2 < B_3)$ for $k = 2$ and `A(k) strictly precedes B(k+1)`. This means that the respective index variables c_i connected by the relations are chosen for every k according to the definition in the CCSL statements.

Example 9. *Consider again the instance from Example 8. Now, additional variables are created for each clock, e. g., $pe_0, pe_1, pe_2, \ldots$ for `photo_earth`, which denote in which step this clock has been ticking for the 0th, 1st, 2st, etc. time. For example, the variable pe_0 pointing to `step` $= 0$ as illustrated in Fig. 8 states that the 0th tick occurred in step 0. The variables without assignment (as shown in the bottom of Fig. 8) simply state that their respective tick has not been conducted yet.*

Having this formulation, the order of clock ticks is symbolically described. Now, this can be constrained in a fashion that a particular order (namely as given by the instant group) is enforced. Considering the instant group from Fig. 5, this requires constraining the variables ps_0, ch_0, and pe_0. In fact, as the corresponding clock tickings are restricted by strict precedes statements, the constraint

$$(ps_0 < ch_0) \wedge (ch_0 < pe_0)$$

has to be employed to ensure a valid order of the respective ticks.

3) Solving the Instance: The instance is now given to a satisfiability solver (in our case an SMT solver) which tries to determine an assignment satisfying all constraints. If such an assignment can be determined, the corresponding values of each variable can be translated back to a particular sequence of simulation steps. This works as witness demonstrating that the behavior defined by the instant group indeed is possible with the additionally considered clock constraints.[2]

In contrast, if the satisfiability solver proves that no assignment exists which satisfies all constraints, it has been shown that no sequence of $step_{max}$ simulation steps exists which satisfies both, the clock constraints as well as the constraints enforced by the instant group. In this case, $step_{max}$ can be increased, e. g., until the diameter of the automaton is reached or until the expected maximum number of time steps in which the behavior is supposed to become evident is reached. If this still yields no satisfiable assignment, it has been proven that the given instant group cannot be executed under the defined clock constraints. This unveils an error in the CCSL specification which can be addressed by the designer prior to the implementation.

[1]Note that the evaluation will be `false`, if one of the index variables remained unassigned.

TABLE I
BEHAVIORAL ANALYSIS FOR INSTANT RELATION GROUPS

Instance	Clk.	Inst.	Rel.	Grp.	States	Trans.	Res.	Time (s)
alternates_corr	3	3	2	1	3	7	✓	2.7
alternates_incorr	3	3	2	1	3	7	✗	5.1
sensors1	4	3	2	1	5	13	✓	4.1
sensors2	4	3	2	1	5	13	✓	4.0
sensors3	4	6	4	2	5	13	✗	4.0
sensors4	8	3	2	1	80	1053	✓	733.6
sensors5	8	4	4	1	80	1053	✓	773.3

V. Application and Evaluation

Using the method described above, clock constraints can automatically be checked against given instant relations and, hence, the timing behavior can be verified prior to its implementation. In order to confirm the applicability of the proposed methodology, the three steps described in Section IV have been implemented in Java/Xtend. Afterwards, we applied the resulting tools to the CCSL descriptions from Figs. 1/2 (denoted by *sensors*) as well as from Figs. 4/5 (denoted by *alternates*). In order to evaluate correct as well as incorrect specifications, different variations of the first example have been considered (distinguished by the suffix *corr* and *incorr*, respectively). Furthermore, the later example (*sensors*) has been considered with a different amount of sensors (and, hence, clocks to be considered).

The results of the application are summarized in Table I. The first columns provide the name of the respective example including its number of clocks (*Clk.*), number of instants (*Inst.*), number of instant relations (*Rel.*), and the eventually resulting number of instant groups (*Grp.*) derived from the respectively given CCSL representation (derived in step 1 of the proposed methodology; see Section IV-A).

[2]Note that this does not guarantee the occurrence of a group in an actual system, as this additionally depends on the precise implementation of the system. Nevertheless, this proof shows that the desired timing behavior is, in principle, possible.

The results from the second step, i e. from the derivation of the automaton (see Section IV-B), are summarized in the next columns. Here, the number of the states (denoted by *States*) and the number of transitions (denoted by *Trans.*) are reported. Note that these numbers only report the size of the final automaton. Intermediate results may be significantly larger due to the need to consider all possible subsets of clock tickings in the automaton—this has already been identified as bottleneck in [10].

Finally, the result of the third step (see Section IV-C) and, hence, the overall result is presented in the final two columns. More precisely, it is denoted whether the instant relations indeed can be realized by the clock constraints or not (denoted by ✓ and ✗ in Column *Res.*). Furthermore, the total run-time (in CPU seconds) is given in the column *Time*.

Overall, the applicability of the proposed approach can be confirmed. In fact, the considered instant relations can be verified against the given clock constraints in acceptable run-time. More importantly, the proposed solution even helped unveiling design flaws in the examples considered above. Actually, the *alternates*-examples behaved as expected, i. e., the correct instance has been proven correct, while the error in the incorrect version has been detected.

Moreover, the *sensor*-example from Figs. 1/2 (where no error has been introduced on purpose) has also been found erroneous. If the two instant groups are considered separately (as done in the experiments summarized in row *sensors1* and *sensors2*), the clock constraints indeed realize the intended behavior. But enforcing both together (as done in the experiments summarized in row *sensors3*) would require both clocks (i. e., `sensor1` and `sensor2`) to tick together. This is implicitly prohibited due to the exclusion between `sensor2` and `echo` (cf. Fig. 1). As a consequence, these instant groups cannot be realized using the clock constraints—the specified behavior is not possible. While not obvious at a first glance, this flaw can easily be detected using the proposed methodology.

VI. Conclusions

In this work, we considered the verification of CCSL clock constraints against their corresponding instant relations. By this, we provided a methodology which, prior to an implementation, allows for checking whether an intended timing behavior is indeed possible with the specified clocking behavior. To this end, we applied an interpretation of instant relations which does not solely rely on monitoring purposes only any more. Furthermore, a symbolic formulation of the clocking behavior is created and, afterwards, checked against the desired instant functionality. Satisfiability solvers are employed to conduct the actual checks. A case study confirmed the general applicability of the proposed methods. Moreover, the proposed solution allowed for identifying errors in CCSL description which were not obvious at the first glance.

Acknowledgments

This work has been supported by the Graduate School SyDe, funded by the German Excellence Initiative within the University of Bremen's institutional strategy, by the German Research Foundation (DFG) within the Reinhart Koselleck project DR 287/23-1, by the German Federal Ministry of Education and Research (BMBF) within the project SPECifIC under grant no. 01IW13001 and the project SELFIE under grant no. 01IW16001 as well as the Siemens AG.

References

[1] R. Drechsler, M. Soeken, and R. Wille, "Formal Specification Level: Towards Verification-driven Design Based on Natural Language Processing," in *Forum on Specification and Design Languages (FDL)*, 2012, pp. 53–58.

[2] Object Management Group, *OMG Systems Modeling Language (OMG SysML^{TM})*. Object Management Group, 2012.

[3] ——, *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*. Object Management Group, 2011.

[4] C. André, F. Mallet, and J. DeAntoni, "VHDL Observers for Clock Constraint Checking," in *International Symposium on Industrial Embedded Systems (SIES)*, 2010, pp. 98–107.

[5] F. Mallet, "Automatic Generation of Observers from MARTE / CCSL," in *Symposium on Rapid System Prototyping (RSP)*, 2012, pp. 86–92.

[6] H. Yu, J.-P. Talpin, and L. Besnard, "Polychronous controller synthesis from MARTE CCSL timing specifications," in *International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, 2011, pp. 21–30.

[7] J. Peters, R. Wille, and R. Drechsler, "Generating SystemC Implementations for Clock Constraints Specified in UML/MARTE CCSL," in *International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2014, pp. 116–125.

[8] F. Mallet and L. Yin, *Correct Transformation from CCSL to Promela for verification*. Institut National de Recherche en Informatique et en Automatique, 2012.

[9] L. Yin, F. Mallet, and J. Liu, "Verification of MARTE/CCSL Time Requirements in Promela/SPIN," in *International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2011, pp. 65–74.

[10] J. Peters, R. Wille, N. Przigoda, U. Kühne, and R. Drechsler, "A Generic Representation of CCSL Time Constraints for UML/MARTE Models," in *Design Automation Conference (DAC)*, 2015.

[11] C. André and F. Mallet, *Clock Constraints in UML/MARTE CCSL*. Institut National de Recherche en Informatique et en Automatique, 2008.

[12] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for Construction and Analysis of Systems*, 2008, pp. 337–340.