Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Periodic scheduling for MARTE/CCSL: Theory and practice

Min Zhang [a,b], Feng Dai [a,b], Frédéric Mallet [b,c,d,*]

[a] *Shanghai Key Laboratory of Trustworthy Computing, ECNU, Shanghai, China*
[b] *MoE International Joint Lab of Trustworthy Software, ECNU, Shanghai, China*
[c] *Université Cote d'Azur, CNRS, I3S, France*
[d] *INRIA Sophia Antipolis Méditerranée, France*

A R T I C L E I N F O

Article history:
Received 1 May 2016
Received in revised form 7 July 2017
Accepted 25 August 2017
Available online xxxx

Keywords:
Periodic scheduling
MARTE/CCSL
Maude
Rewriting logic
Model checking

A B S T R A C T

The UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) is used to design and analyze real-time and embedded systems. The Clock Constraint Specification Language (CCSL) is a companion language for MARTE. It introduces logical clocks as first class citizens as a way to formally specify the expected behavior of models, thus allowing formal verification. CCSL describes the expected infinite behaviors of reactive embedded systems. In this paper we introduce and focus on the notion of periodic schedule to allow for a nice finite abstraction of these infinite behaviors. After studying the theoretical properties of those schedules we give a practical way to deal with them based on the executable operational semantics of CCSL in rewriting logic with Maude. We also propose an algorithm to find automatically periodic schedulers with the proposed sufficient condition, and to perform formal analysis of CCSL constraints by means of customized simulation and bounded LTL model checking.

1. Introduction

Lamport's logical clocks [10] originated from the need to synchronize distributed systems without assuming a reliable single common timing mechanism to compare events. Indeed, maintaining a global clock in a widely distributed system may be very costly. Rather than maintaining a total order on events, a partial view was proposed as sufficient to maintain causal relationships.

In a very different community, synchronous languages [3,2,21] use the word clock to emphasize a multiform notion of time where the notion of physical time is initially relaxed with the notion of ordering. The main important difference is that synchronous languages accept instantaneous causal relations, which are an important abstraction in synchronous circuits but are less relevant in widely distributed environments.

The Clock Constraint Specification Language (CCSL) [1,13] proposes a concrete syntax to handle logical clocks as first-class citizens. While synchronous languages mainly focus on signals and values and use logical clocks as a controlling mechanism, CCSL discards the values and only focuses on clock-related issues. In this paper, we focus on the periodic scheduling of CCSL from both theoretical and practical perspectives, considering that reactive embedded systems have recurrent behaviors for which the design of correct periodic schedulers is very important in the development of such systems [12,6]. While deciding on the existence of a schedule for a given set of CCSL constraints is still an open problem, we discuss here a

[*] Corresponding author at: MoE International Joint Lab of Trustworthy Software, ECNU, Shanghai, China.
E-mail addresses: zhangmin@sei.ecnu.edu.cn (M. Zhang), fdai_itlogic@163.com (F. Dai), Frederic.Mallet@unice.fr (F. Mallet).

http://dx.doi.org/10.1016/j.scico.2017.08.015
0167-6423/© 2017 Elsevier B.V. All rights reserved.

sufficient condition for having a class of bounded schedules which can be extended to infinite but periodic ones. In our earlier work [26], we considered a pragmatic point of view and a very restrictive condition for the existence of periodic schedules. We propose here a much less restrictive condition, that (1) allows to find more periodic schedules, (2) is less restrictive on the condition to find a periodic schedule, and (3) considers a bigger subset of CCSL constraints. About (3), we consider specifically the *periodic filter* of CCSL, which offers a generic notion of periodicity on logical clocks that generalizes the classical definition of periodic activation. In this paper, we prove the correctness of our condition and present an operational method for building periodic schedules from a bounded schedule that satisfies the condition.

Furthermore, we also give a formal executable operational semantics of the extended CCSL language using Maude [4], an algebraic language based on rewriting logic. The formal operational semantics of CCSL was initially defined in a research report [1] in a bid to provide a reference semantics for building simulation tools, like TimeSquare [7]. Maude provides various formal analysis approaches, such as simulation, state space exploration (exhaustive or bounded, symbolic or explicit-state), by which we can analyze CCSL specifications to, for instance, check the existence of desired schedules with specific properties (e.g., reduce memory usage), produce a schedule or simulation by applying customized scheduling policies, and verify the satisfiability of expected properties.

The benefits of the new semantics defined in rewriting logic are multifold. The first benefit is that rewriting logic gives a direct implementation of the operational semantics while TimeSquare provides a Java-based implementation which is prone to introduce unexpected complexity. The second and most important benefit is that we can directly use the tools that are provided for rewriting logic to analyze a CCSL specification by means of simulation, state-space exploration, and even linear temporal logic model checking. Previous work on studying CCSL properties [14] relies on several intermediate transformations to automata and other specific formats so that model-checking becomes possible when a CCSL specification is finite. A CCSL specification is called finite if it can be transformed into a finite-state automaton [15]. However, some CCSL operators, which are called unsafe operators, cannot be transformed into finite-state automata. It either meant reducing to a safe subset of CCSL [9] or detecting that the specification was describing a finite reachable state-space even though relying on unsafe operators. In this contribution, we rely on the Maude environment [4] to provide a direct analysis support to CCSL specifications by formally defining its operational semantics in Maude, and we can explore unsafe specifications using bounded model checking and do not restrict to the safe subset.

For periodic scheduling, we provide a prototype implementation in Maude based on the new formal semantics of CCSL to detect the proposed conditions and build the satisfying schedule. As another contribution, we propose five arbitration policies to reduce the set of possible solutions when a CCSL specification is under-specified. Such arbitration policies can be seen as optimization criteria akin to those classically used in real-time scheduling to optimize memory or bandwidth. Those arbitration policies can also be naturally implemented in Maude based on the new formal semantics.

This paper is an extended version of our previous work [26]. Apart from adding much more detail on the formal semantics of CCSL, this extended version adds the following contributions:

1. We consider an operator for the CCSL language that was ignored in previous work, the so-called *periodic filter*, which serves for the specification of logical repetitive patterns between logical clocks.
2. We propose a less constraining sufficient condition for the existence of periodic schedules and give a formal proof of its correctness.
3. We present a formal executable operational semantics of the extended CCSL language in Maude and illustrate its applications to various formal analysis tasks, such as checking the existence of bounded and periodic schedules, deriving a customized simulation and performing LTL model-checking.
4. More user-defined arbitration policies are supported for customized simulations of schedules.

The rest of this paper is organized as follows. Section 2 introduces CCSL and the notion of schedule with bounded and periodic restrictions. It also gives conditions for being able to build a periodic schedule, and we prove that those conditions are sufficient. Section 3 gives a brief introduction to Maude. Section 4 discusses the encoding of the semantics of CCSL in Maude and details the way its environment can be used to compute bounded and periodic schedules. Section 5 considers several examples to illustrate the interest of this encoding and the usefulness of our tool. Finally, Section 6 compares this work to previous work, including our own, and Section 7 gives some concluding remarks.

2. The clock constraint specification language with periodic filter

2.1. Syntax and semantics of CCSL

CCSL relies on the notion of logical clocks, which are commonly used to express partial orders in distributed systems [10] or synchronization conditions in synchronous languages [2]. We use the wording clock or logical clock indistinctly in the following. While traditional synchronous languages give a syntax to combine signals, infinite sequences of values, and use clocks to express when the signals are present (have a value), CCSL discards the values on purpose to focus on relationships among clocks.

Definition 1 *(Logical clock).* A logical clock c is an infinite sequence (a stream) of ticks, $(c_n)_{n\in\mathbb{N}^+}$.