ISBN 978-83-60810-14-9
ISSN 1896-7094

Real Time Behavior of Data in Distributed Embedded Systems

Tanguy Le Berre, Philippe Mauran, Gérard Padiou, Philippe Quéinnec
Université de Toulouse – IRIT
2, rue Charles CAMICHEL
31071 TOULOUSE Cedex 7
{tleberre,mauran,padiou,queinnec}@enseeiht.fr

***Abstract*—Nowadays, most embedded systems become distributed systems structured as a set of communicating components. Therefore, they display a less deterministic global behavior than centralized systems and their design and analysis must address both computation and communication scheduling in more complex configurations.**

We propose a modeling framework centered on data. More precisely, the interactions between the data located in components are expressed in terms of a so-called observation relation. This abstraction is a relation between the values taken by two variables, the source and the image, where the image gets past values of the source. We extend this abstraction with time constraints in order to specify and analyze the availability of timely sound values.

The formal description of the observation-based computation model is stated using the formalisms of transition systems. Real time is introduced as a dedicated variable.

As a first result, this approach allows to focus on specifying time constraints attached to data and to postpone task and communication scheduling matters. At this level of abstraction, the designer has to specify time properties about the timeline of data such as their freshness, stability, latency…

As a second result, a verification of the global consistency of the specified system can be automatically performed. A forward or backward approach can be chosen. The verification process can start from either the timed properties (e.g. the period) of data inputs or the timed requirements of data outputs (e.g. the latency).

As a third result, communication protocols and task scheduling strategies can be derived as a refinement towards an actual implementation.

I. Introduction

DISTRIBUTED Real Time Embedded (DRE) systems become more and more widespread and complex. In this context, we propose a modeling framework centered on data to specify and analyze the real time behavior of these DRE systems. More precisely, such systems are structured as time-triggered communicating components. Instead of focusing on the specification and verification of time constraints upon computations structured as a set of tasks, we choose to consider data interactions between components. These interactions are expressed in terms of an abstraction called observation, which aims at expressing the impossibility for a site to maintain an instant knowledge of other sites. In this paper, we extend this observation with time constraints limiting the time shift induced by distribution. Starting from this modeling framework, the specification and verification of real time data behaviors can be carried out.

In a first step, we outline some related works which have adopted similar approaches but in different contexts and/or different formalisms.

Then, we describe the underlying formal system used to develop our distributed real time computation model, namely state transition systems. In this formal framework, we define a dedicated relation called observation to describe data interactions. An observation relation describes an invariant property between so-called "source" and "image" variables. Informally, at any execution point, the history of the image variable is a sub-history of the source variable. In other words, an observation abstracts the relation between arguments/results of a computation or between inputs/outputs of a communication protocol.

To express timed properties on the variables and their relation, we extend the framework so as to be able to describe the timeline of state variables. Therefore, for each state variable x, an abstraction of its timeline is introduced in terms of an auxiliary variable $\hat{x}$ which records its update instants. Then, real time constraints on data, for instance periodicity or steadiness, are expressed as differences between these dedicated variables and/or the current time. These auxiliary variables are also used to restrict the time shift between the source and the image of an observation. The semantics of the observation relation is extended to express different properties: for instance time lag or latency boundaries between the current value of the image and its corresponding source value.

The real time constraints about data behavior can be specified by means of these timed observations as illustrated in an automotive speed control example.

Lastly, we discuss the possibility to check the consistency of a specification. A specification is consistent if and only if the verification process can construct correct executions. However, the target systems are potentially infinite and an equivalent finite state transition system must be derived from the initial one before verification. The feasibility of this transformation is based upon assumptions about finite boundaries of the time constraints.

II. State of the Art

We are interested in systems such as sensors networks. Our goal is to guarantee that the input data dispatched to processing units are timely sound despite the time shift introduced by distribution.

Most approaches taken to check timed properties of distributed systems are based on studying the timed behavior of tasks. For example, works like [1] propose to include the timed properties of communication in classical scheduling analysis.

Our approach is state-based and not event-based. We express the timed requirements as safety properties that must be satisfied in all states. The definition of these properties do not refer to the events of the system and is only based on the variable values. We depart from scheduling analysis by focusing on variable behavior and not considering the tasks and related system events.

Others approaches based on variables are mainly done in the field of databases. For example, the variables semantics and their timed validity domain are used in [2] to optimize transaction scheduling in databases. Our work is at a higher level as we propose to give an abstract description of the system in terms of a specification of data relations. Our framework can be used to check the correctness of an algorithm with regards to the freshness of the variables. It can also be used to specify a system without knowing its implementation.

Similar works are done using temporal logic to specify the system. For example, in [3], OCL constraints are used to define the validity domain of variables. A variation of TCTL is used to check the system synchronization and prevent a value from being used out of its validity domain. This work also defines timed constraints on the behavior and the relations between application variables, but these relations are defined using events such as message sending whereas our definitions are based on the variable values.

In [4], an Allen linear temporal logic is proposed to define constraints between intervals during which state variables remain stable. In other words, this approach also uses an abstraction of the data timelines in terms of stability intervals. However, the constraints remain logical and do not relate to real time. Nevertheless, this approach is intended to be applied in the context of autonomous embedded systems.

Using a semantics based on state transition system, we give a framework which aims at describing the relations between the data in a system, and specifying its timed properties and requirements.

III. Theoretical Basis

A. State Transition System

Models used in this paper are based on state transition systems. More precisely, this paper relies on the TLA+ formalism [5]. A *state* is an assignment of values to variables. A *transition relation* is a predicate on pair of states. A *transition system* is a couple (set of states, transition relation). A *step* is a pair of states which satisfies the transition relation. An *execution* σ is any infinite sequence of states $\sigma_0\sigma_1\ldots\sigma_i\ldots$ such that two consecutive states form a step. We note $\sigma_i \rightarrow \sigma_{i+1}$ the step between the two consecutive states σ_i and σ_{i+1}.

A *temporal predicate* is a predicate on executions; we note $\sigma \models P$ when an execution σ satisfies the predicate P. Such a predicate is generally written in linear temporal logic. A *state expression* e (in short, an expression) is a formula on variables; the value of e in a state σ_i is noted $e.\sigma_i$. The sequence of values taken by e during an execution σ is noted $e.\sigma$. A *state predicate* is a boolean-valued expression on states.

B. Introducing Time

We consider real time properties of the system data. To distinguish them from (logical) temporal properties, such properties are called *timed* properties. Time is integrated in our transition system in a simple way, as described in [6]. Time is represented by a variable T taking values in an infinite totally ordered set, such as $\mathbb{N}$ or $\mathbb{R}^+$. T is an increasing and unbound variable. There is no condition on the density of time and moreover, it makes no difference whether time is continuous or discrete (see discussion in [7]). However, as an execution is a sequence of states, the actual sequence of values taken by T during a given execution is necessarily discrete. This is the digital clock view of the real world. Note that we refer to the variable T to study time and that we do not use the usual timed traces notation.

An execution can be seen as a sequence of snapshots of the system, each taken at some instant of time. We require that there are "enough" snapshots, that is that no variable can have different values at the same time and so in the same snapshot. Any change in the system implies time passing.

Definition 1: Separation. An execution σ is separated if and only if for any variable x:

$$\forall i, j : T.\sigma_i = T.\sigma_j \Rightarrow x.\sigma_i = x.\sigma_j$$

In the following, we consider only separated executions. This allows to timestamp changes of variables and ensures a consistent computation model.

C. Clocks

Let us consider a totally ordered set of values $\mathcal{D}$, such as $\mathbb{N}$ or $\mathbb{R}^+$. A clock is a (sub-)approximation of a sequence of $\mathcal{D}$ values. We note $[X \rightarrow Y]$ the set of functions with domain X and range contained by Y.

Definition 2: A clock c is a function in $[\mathcal{D} \rightarrow \mathcal{D}]$ such that:

- it never outgrows its argument value: $\forall t \in \mathcal{D} : c(t) \leq t$
- it is monotonously increasing: $\forall t, t' \in \mathcal{D} : t < t' \Rightarrow c(t) \leq c(t')$

In the following, clocks are used to characterize the timed behavior of variables. They are defined on the values taken by the time variable T, to express a time delayed behavior, as well as on the indices of the sequence of states, to express a logical precedence. A clock subset is used:

Definition 3: A clock c from $[\mathcal{D} \rightarrow \mathcal{D}]$ is a *liveclock* if and only if:

$$\forall t \in \mathcal{D} : \exists t' \in \mathcal{D} : c(t') > c(t)$$

IV. Specification of Data Timed Behavior

We introduce here the relation and properties used in our framework to describe the properties that must be satisfied by a system. Our approach is state-based and gives the relation that must be satisfied in all states. We define the observation relation to describe the relation between variables. A way to describe the timed behavior of the variables, that is properties of the history of data, is then introduced. We then extend the observation relation to take into account the timed constraints on the relation between variables. For that purpose we define predicates which bind relevant instants of the timeline of the source and the image of an observation. The predicates are expressed as bounds on the difference between two relevant instants.

A. The Observation Relation

We define an observation relation on state transition systems as in [8]. The observation relation is used to abstract the value correlation between variables. The values taken by one variable are values previously taken by another variable.

Thus the observation relation binds two variables, the source x and the image $`x$, and denotes that the history of the variable $`x$ is a sub-history of the variable x. The relation is defined by a couple $< source, image >$ and the existence of at least a clock that defines for each state which one of the previous values of the source is taken by the image. The formal definition is:

Definition 4: The variable $`x$ is an observation of the variable $x$ in execution $\sigma$: $\sigma \vDash {`x} \prec x$ iff:

$$\exists\ c \in [\mathbb{N} \rightarrow \mathbb{N}] : liveclock(c) \wedge\ \forall i : {`x}.\sigma_i = x.\sigma_{c(i)}$$

This relation states that any value of $`x$ is a previous value of $x$. Due to the properties of the observation clock $c$, $`x$ is assigned x values respecting the chronological order. Moreover, c always eventually increases, so $`x$ is always eventually updated with a new value of x. Figure 1 shows an example of an observation relation.

The observation can be used as an abstraction of communication in a distributed system, but it can as well be used as an abstraction of a computation:

- Communication consists in transferring the value of a local variable to a remote one. Communication time and lack of synchronization create a lag between the source and the image, which is modeled by $distant \prec local$.
- In state transition systems, a computation $f(x)$ is instantaneously computed. By writing $y \prec f(x)$, we model the fact that the computation takes time and the value of y is based on the value of x at the beginning of the computation.

In order to extend the observation relation with real time properties, we define instants that are used to characterize the timeline of variables.

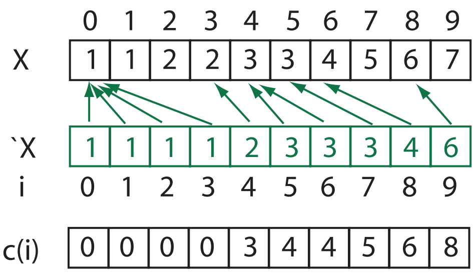

Fig. 1. The Observation Relation

B. The Update Timeline of Variables

In order to state properties on the timed behavior of a variable x, we want to be able to refer to the last time this variable was updated. These are called the update instants $\hat{x}$. This referential can be either explicit or implicit. In the explicit case, the developer is responsible for giving its own variable $\hat{x}$. This can be the case if there is a periodic behavior of x without having to describe actual values of x.

In the implicit case, a formal definition of $\hat{x}$ is given based on the history of the values taken by x. The goal is to capture the instant when the current value of x appeared, e.g. the beginning of the current occurrence.

Definition 5: For a separated execution σ and a variable x, the variable $\hat{x}$ is defined by:

$$\forall i : \hat{x}.\sigma_i = T.\sigma_{min\{j | \forall k \in [j..i]:\ x.\sigma_i = x.\sigma_k\}}$$

$\hat{x}$ is built from the history of x values. For a variable x, the update instant of x is defined as the value taken by the time T at the earliest state when the current value appeared and continuously remained unchanged until the current state.

When x is updated and its value changes then the value of $\hat{x}$ is also updated. Conversely if $\hat{x}$ changes then x is updated. This property allows us to rely on the values of $\hat{x}$ to study the timed properties of x.

We also define the instant $Next(\hat{x})$ that returns the next value of $\hat{x}$ and thus the next instant when the value of x is updated, i.e. the instant when the current value disappears. If x is stable in a state σ_i (no new update), then $Next(\hat{x}).\sigma_i = +\infty$.

C. Variable Behavior

$\hat{x}$ is used to describe the timed behavior of a variable x. In this paper, we focus only on a certain type of variable. We expect each value of each variable to remain unchanged a bounded number of time units. We want to be able to give two characteristics for each variable: the minimum and the maximum duration between two updates. This basically describes two behaviors: a sporadic variable keeps each value for a minimum duration and, on the contrary, an alive variable has to be updated often, no value can be kept more than a given duration. These properties are expressed by a limit on the difference between $\hat{x}$ and $Next(\hat{x})$ using a characteristic called $Steadiness$. This bound denotes how long each value of x can be kept.

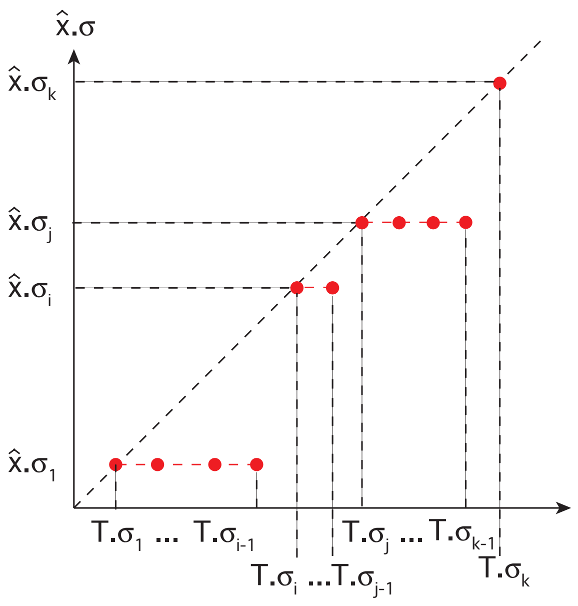

Fig. 2. Graph of $\hat{x}$

Definition 6: The steadiness of a variable x is defined by:

$$\sigma \vDash x\ \{Steadiness(\delta, \Delta)\} \triangleq \\ \forall i : \delta \leq Next(\hat{x}).\sigma_i - \hat{x}.\sigma_i < \Delta$$

$\Delta - \delta$ is the jitter on x updates. As long as there exists δ and Δ such that a variable timeline can be described using the $Steadiness$ feature then other properties can be given. For example, we introduce a stronger property, periodicity, where no time drift is allowed.

Definition 7: A variable $\hat{x}$ is periodic of period P with jitter J and phase ϕ iff:

$$\sigma \vDash x\ \{Periodic(P, J, \Phi)\} \triangleq \\ x\ \{Steadiness(P - 2J, P + 2J)\} \wedge \\ \forall i : \exists n \in \mathbb{N} : \hat{x}.\sigma_i \in [\phi + nP - J, \phi + nP + J]$$

(J must verify $J < P/4$)
Such a variable is updated around all instants $\phi + nP$.

D. Timed Observation

We use the update instants to extend the observation relation with timed characteristics. The timed constraints that extend the observation must capture the latency introduced by the observation and the modification of the timeline of the source to produce the timeline of the image. We define a set of predicates on the instants characterizing the source and the image timelines and the observation clock. Formally, a timed observation is defined as follows:

Definition 8: A timed observation is defined as an observation satisfying a set of predicates.

$$\sigma \vDash `x \prec x \left\{ \begin{array}{l} Predicate_1(\delta_1, \Delta_1), \\ Predicate_2(\delta_2, \Delta_2), \\ \quad\ldots \end{array} \right\} \triangleq \\ \exists c \in [\mathbb{N} \rightarrow \mathbb{N}] : liveclock(c) \wedge \\ \forall i : `x.\sigma_i = x.\sigma_{c(i)} \wedge \\ Predicate_1(c, \delta_1, \Delta_1) \wedge \\ Predicate_2(c, \delta_2, \Delta_2) \ldots$$

The predicates that can be used to describe the timed properties of the relation between two variables are the following:
Definition 9: Given two variables $`x$ and $x$ such that $\sigma \vDash `x \prec x$ with a $liveclock$ $c \in [\mathbb{N} \rightarrow \mathbb{N}]$

$$\begin{array}{rcl} Lag(c, \delta, \Delta) & \triangleq & \delta \leq `\hat{x}.\sigma_i - \hat{x}.\sigma_{c(i)} < \Delta \\ Stability(c, \delta, \Delta) & \triangleq & \delta \leq Next(\hat{x}).\sigma_{c(i)} - \hat{x}.\sigma_{c(i)} < \Delta \\ Latency(c, \delta, \Delta) & \triangleq & \delta \leq T.\sigma_i - \hat{x}.\sigma_{c(i)} < \Delta \\ Medium(c, \delta, \Delta) & \triangleq & \delta \leq T.\sigma_i - T.\sigma_{c(i)} < \Delta \\ Freshness(c, \delta, \Delta) & \triangleq & \delta \leq T.\sigma_{c(i)} - \hat{x}.\sigma_{c(i)} < \Delta \\ Fitness(c, \delta, \Delta) & \triangleq & \delta \leq Next(\hat{x}).\sigma_{c(i)} - T.\sigma_{c(i)} < \Delta \end{array}$$

When no lower (resp. upper bounds) is given, 0 (resp. $+\infty$) is used.

These predicates have to be true at every state and every instant. The definition of an observation is done by giving which predicates must be satisfied. Other predicates can be proposed but we believe this set is sufficient to express the different behaviors that must be analyzed.

The first three predicates describe timed requirements on the system. Lag is used to bound the lag introduced by the observation. The current value of the image was available on the source in one of the past instants. The bounds constrain how far in the past the image's current value is found on the source. They are the strongest needed to characterize all possible values of the image and so are defined using particular instants: the update instants of the image and the source. $Stability$ is used to get source values based on their duration. For example, we can eliminate transient values and keep sporadic ones, or the contrary. $Latency$ is a measure of the total duration between the current instant and the assignment of the image's current value on the source.

The last three predicates are used to define or restrict the behavior of the system due to its architecture. Predicate $Medium$ characterizes the medium implementing the observation, for example a communication bus or a processing unit. A lower bound is the shortest time needed to read the value of the source and assign it to the image. For example, it corresponds to the minimum communication time. An upper bound is the longest duration needed to create and send a message, for example, the maximum communication time plus the maximum time during which the source is unavailable.

When an observation denotes a computation, the bounds stand for the minimum execution time and the maximum execution time plus the blocking time (due to the scheduling policy for example). $Freshness$ and $Fitness$ are used to define intervals, relative to the update instants, where the value of the source is not available. The observation clock

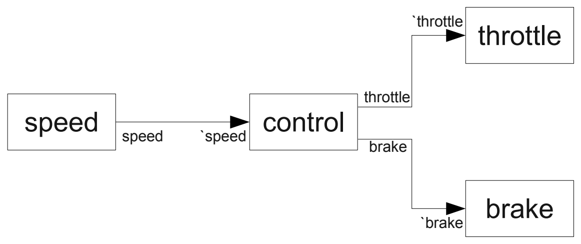

Fig. 3. Cruise Control System

is prevented from referring to one of these instants. An upper bound on $Freshness$ prevents values that are not fresh anymore to be read. On the contrary, a lower bound denotes an impossibility to access a value just after its assignment. $Fitness$ allows, or forbids, to read a value according to the time remaining until its removal; a lower bound prevents the value to be used just before a new update, for example when the computation of this new update has already started.

Note that, at the beginning of an execution, some predicates such as $Medium$ cannot be satisfied. In order to solve this problem, we define an observation where timed predicates do not have to be satisfied in initial states. The image values are replaced by a given default value. This extension is similar to the ”followed by” operator $\rightarrow$ in Lustre [9].

V. A System Definition

A. A Brief Description

We give here a system specification using previous properties. This system is a simplified car cruise control system used as an example.

The goal of such a system, when activated, is to control the throttle and the brakes in order to reach and keep a chosen speed. The system is based on different components (see Figure 3):

- a speed monitor which computes the current speed, based on a sensor counting wheel turns;
- the throttle actuator which controls the engine;
- the brakes which slow the car;
- the control system which handles the speed depending on the current and the chosen speed;
- a communication bus which links the devices and the control system.

The environment, the driver and the engine influence the speed of the car. Once the cruise control is activated and a speed is chosen, the control system chooses whether to accelerate by increasing the voltage of the throttle actuator or to decelerate by decreasing this voltage and by using brakes. The system has a maximum delay between each change to ensure a reactive behavior.

Each component uses and/or produces data. We use observations to specify the system and characterize correct executions. The notation is the one introduced when the timed observation and the variable behavior were defined.

B. Relations Between Data

Firstly, we present some relations between the variables of the example. These relations, either computations or communications, are expressed as observations. The speed monitor computes the values of a variable $speed$ and these values are sent to the control system as a variable $`speed$. We express this as an observation $`speed \prec speed$.

The decisions of the control system are based on the current speed and more precisely on the value of $`speed$. Two functions are used to compute the values used as inputs by the brakes and the throttle actuator. Using the speed values, we compute the values of two variables: $throttle \prec control1(`speed)$ and $brake \prec control2(`speed)$.

Finally, the decisions $throttle$ and $brake$ are sent to dedicated devices into variables $`throttle$ and $`brake$, such that $`throttle \prec throttle$ and $`brake \prec brake$.

C. Requirements and Properties

We state the requirements and known timed properties of the system and explain how to express them as characteristics of the system variables and observations. These characteristics are all given in Figure 4.

The speed is computed using the ratio between the number of wheel turns and the time. A minimum time is required to give a significant result. So the variable $speed$ must have a minimum time δ_1 between each update. Due to scheduling constraints, computation of variables $throttle$ and $brake$ also have minimum times δ_2 and δ_3 between each update.

Each communication on the bus has a minimum communication time, regardless of the communicating protocol that is chosen. We introduce a lower bounded $Medium$ predicate on the observations denoting communication. Similarly there is also a lower bounded delay to represent the minimum computation time of the functions $control1$ and $control2$.

We expect each data to be used not too long after each update. More precisely we want each decision applied to the brake or to the throttle to be based on fresh values of the speed. Thus, we require the complete processing chain to be achieved in a short enough time.

A composition of observations is an observation, for example if $y \prec x$ and $z \prec f(y)$ then $z \prec f(x)$ [8]. We use this property to express the requirement on the complete processing chains between $`throttle$, $`brake$ and the variable $speed$. The definitions of these observations are compatible with the dependencies between the variables. The processing chains must satisfy upper bounded $Latency$ predicates on the observations. These are the only upper bounded characteristics given in the abstract system specification. Upper bounds on the $Medium$ and $Steadiness$ characteristics of the other observations and variables are implicitly imposed by the $Latency$ upper bound (Δ_7).

D. Case Study Analysis

The goal of the analysis is to prove that the specification is consistent and that there is at least one execution satisfying the requirements. In our example, the set of valid executions

```
- variables behaviours:
```
$$speed\ \{Steadiness(\delta_1, +\infty)\}$$
$$throttle\ \{Steadiness(\delta_2, +\infty)\}$$
$$brake\ \{Steadiness(\delta_3, +\infty)\}$$
```
- communications:
```
$$`speed \prec speed\ \{Medium(\delta_4, +\infty)\}$$
$$`throttle \prec throttle\ \{Medium(\delta_4, +\infty)\}$$
$$`brake \prec brake\ \{Medium(\delta_4, +\infty)\}$$
```
- computations:
```
$$throttle \prec control1(`speed)\ \{Medium(\delta_5, +\infty)\}$$
$$brake \prec control2(`speed)\ \{Medium(\delta_6, +\infty)\}$$
```
- complete processing chains:
```
$$`throttle \prec control1(speed)\ \{Latency(0, \Delta_7)\}$$
$$`brake \prec control2(speed)\ \{Latency(0, \Delta_7)\}$$

Fig. 4. System Specification

ensures the availability of timely sound values. From this set, we deduce the required update frequency of the *speed* variable. For example, we check the existence of a maximum time acceptable between each update. We analyze the admissible values of the $Medium$ to deduce the communication and computation times that are permitted. Finally, we determine the possible values of the observation clocks in the states corresponding to update instants of the image. These values give the instants at which the values of the source are caught and so, for example the instants when a message must be sent or a computation must start.

For all these properties, a choice must be done. For example, choosing a set of executions may alleviate the bounds on communication time but then reduce the instants where the message must be sent.

VI. System Analysis

We give here properties of our framework based on observations in order to carry out an analysis. A system specified with observation relations must be analyzed to check the consistency of the specification, i.e. if there exists an execution satisfying the specification.

We discuss the analysis method in a discrete context. The semantics of the specification is restricted by discretizing time: i.e. the values taken by time T are in $\mathbb{N}$. For discussion about the loss of information using discrete time instead of dense time and defending our choice, see [7] for example.

A. Equivalence with a Finite System

Given a specification based on our framework, the value of T is unbounded and we have no restriction on the values that can be taken by variables. Therefore the system defined by the specification is infinite. Nevertheless, we can build a finite system equivalent to the specification for the timed properties studied with this framework. This allows us to check the consistency of the specification in a finite time. Here are the main principles of this proof. The definition of a finite system bisimilar to the specified one is based on two equivalence relations.

Since the scope of this framework is to check the satisfaction of timed requirements, we focus on the auxiliary variables used to describe the timeline of each application variable. We define a system where only variables denoting instants are kept, i.e.. the variable describing the timelines and the observation clocks. The states and transitions of the system are defined by the values of these variables and the satisfaction of observations and variables properties. Allowed states and transitions do not depend on the values that can be taken by each variable but on the instants describing their timeline and on the observation clocks. Thus, when we build a system where only these instants are considered, we do not lose or add any characteristics about the timed behavior of the system. We define an equivalence where two states are equivalent if and only if the observation clocks and the variables denoting the update instants are equal. This equivalence is used to build a bisimilarity relation between the specified system and the one built upon only the instants.

The second reason preventing to consider a bounded number of states is the lack of bound on time. The values of the update instants and observation clocks are also unbounded. In order to reduce the possible values that can be taken by the system variables denoting instants, we define a system where all values of the instants are stored modulo the length of an analysis interval. We denote this number as L. L must be carefully chosen, greater than the upper bounds on the variables $Steadiness$ and the observations $Latency$ characteristics and it has to be a multiple of the variable periods.

Such a number L only exists if all variables and observations have upper bounded characteristics. When the source of an observation is bounded and so is the observation, such a bound is deduced for the image. Restricting the behavior by expecting variables to be frequently updated and the shift introduced by distribution to be bounded seems consistent for such real time systems.

In the system defined by the specification, transitions are based on differences between the instants characterizing the variable timelines. These differences cannot exceed the chosen length L. Thus, for each state, if the value of the time T is known and if the values of the other variables are known modulo L, then for each variable there is only one possible real value that can be computed using the value of T. Consequently, considering the clock values modulo this length does not add or remove any behavior of the original system. We define an equivalence where two states are equivalent if the update instants and the observation clocks are equal modulo L. A system built by considering all values modulo L is bisimilar with the original system using this equivalence.

Based on these two equivalences, we build a system by removing variables which do not denote update instants or observation clocks and by considering the values modulo L. This system is bisimilar to the specification and preserves the timed properties. Since all values are bounded by the length of the analysis interval and there is a bounded number of values, it defines a system with a bounded number of states. This result proves the decidability of the framework for the verification of safety properties that can be done using the finite system.

B. Complexity

We have proved the existence of a finite system equivalent to our system. We give here the complexity of a process to effectively build this equivalent finite system. In order to build a transition from a state to a new state we build a set of inequalities deduced from the properties of the previous state and from the observation and variable properties. To solve this set of inequalities and so deduce the possible values of instant variables in the new state, we use difference bound matrices [10]. Considering a system where n variables are studied, the size of each matrix is in $O(n^2)$ and thus the complexity for reducing it to its canonical form and so building the new state is in $O(n^3)$ [10]. The maximum number of states to build depends on all possible combinations of values taken by variables. Each timed variable can take values between 0 and L and the number of instant variables is a multiple of n so we have $O(L^n)$ states. Lastly the complexity to build the system is in $O(n^3 * L^n)$ and considering the memory, we have to store $O(L^n)$ states and $O(L^{2n})$ transitions. Therefore this direct approach is technically feasible only with small enough systems.

C. Other Approaches

Since our approach relies on the TLA+ formalism, we could have used the dedicated tool TLC, the TLA+ model checker. A logical definition of the observation requires the temporal existential quantifier $\boldsymbol{\exists}$, which is not implemented in TLC. Therefore a concrete definition of the observation based on an explicit observation clock has been used. It is only after we have reduced the system to a finite one that a model checker such as TLC could be used.

To be able to more precisely characterize executions satisfying the specification, we currently explore methods to build these executions more easily. A first proposal is to reduce the complexity of such a process by relying on proofs on system properties. The proof approach can easily be used only under certain conditions and in order to proceed to some system simplifications. For example, a periodic source induces properties for its image through an observation. Using these properties reduces the number of states we have to build by forecasting some impossible cases. Proving the full correctness of the system is possible but it is complex and it has not been automatized yet.

Another way is to use controller synthesis methods [11]. Properties of the observation can be expressed as safety properties using LTL and be derived as Bchi automata [12]. Two automata describe the behavior of the source and the image of an observation, exchanging values through a queue. Restrictions can be added to introduce the used implementation and its compatibility with executions defined by the specification. The complexity of controller synthesis methods has still to be explored.

VII. Conclusion

We propose an approach focused on variables instead of tasks and processes, to model and analyze distributed real time systems. We specify an abstract model postponing task and communication scheduling. Based on the state transition system semantics extended by a timed referential, we express relations between variables and the timed properties of variables and communications. These properties are used to check the freshness of values, their stability, and the consistency of requirements. A possible analysis is to build a finite system bisimilar to the specified one. The results are used to help implementation choices.

Perspectives are to search other methods that decrease the complexity of the analysis of a specification and to use this approach with different examples to expand the number of available properties and increase expressivity. We also work on using analysis results to help generating an implementation satisfying the specification.

References

[1] K. Tindell and J. Clark, “Holistic schedulability analysis for distributed hard real-time systems,” *Microprocessing and Microprogramming—Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems)*, vol. 40, pp. 117–134, 1994.

[2] M. Xiong, R. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley, “Scheduling transactions with temporal constraints: exploiting data semantics,” in *RTSS ’96: Proc. of the 17th IEEE Real-Time Systems Symposium*, 1996, pp. 240–253.

[3] S. Anderson and J. K. Filipe, “Guaranteeing temporal validity with a real-time logic of knowledge,” in *ICDCSW ’03: Proc. of the 23rd Int’l Conf. on Distributed Computing Systems*. IEEE Computer Society, 2003, pp. 178–183.

[4] G. Roşu and S. Bensalem, “Allen Linear (Interval) Temporal Logic—Translation to LTL and Monitor Synthesis,” in *International Conference on Computer-Aided Verification (CAV’06)*, ser. Lecture Notes in Computer Science, no. 4144. Springer Verlag, 2006, pp. 263–277.

[5] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[6] M. Abadi and L. Lamport, “An old-fashioned recipe for real time,” *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 5, pp. 1543–1571, September 1994.

[7] L. E. A. and S.-V. A., “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, Dec. 1998.

[8] M. Charpentier, M. Filali, P. Mauran, G. Padiou, and P. Quinnec, “The observation : an abstract communication mechanism,” *Parallel Processing Letters*, vol. 9, no. 3, pp. 437–450, 1999.

[9] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, “The synchronous data-flow programming language LUSTRE,” *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, September 1991.

[10] J. Bengtsson and W. Yi, “Timed automata: Semantics, algorithms and tools,” in *Lecture Notes on Concurrency and Petri Nets*, ser. Lecture Notes in Computer Science vol 3098, W. Reisig and G. Rozenberg, Eds. Springer-Verlag, 2004.

[11] E. Asarin, O. Maler, and A. Pnueli, “Symbolic controller synthesis for discrete and timed systems,” in *Hybrid Systems II*. London, UK: Springer-Verlag, 1995, pp. 1–20.

[12] M. Y. Vardi, “An automata-theoretic approach to linear temporal logic,” in *Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science*. Springer-Verlag, 1996, pp. 238–266.