Scalable Computing: Practice and Experience
Volume 10, Number 3, pp. 229–239. http://www.scpe.org
ISSN 1895-1767

REAL TIME BEHAVIOR OF DATA IN DISTRIBUTED EMBEDDED SYSTEMS*

TANGUY LE BERRE, PHILIPPE MAURAN, GÉRARD PADIOU, PHILIPPE QUÉINNEC†

Abstract. Nowadays, embedded systems appear more and more as distributed systems structured as a set of communicating components. Therefore, they show 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, a source and an 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 formalism of transition systems, where real time is handled 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. The verification process can start either from the timed properties (e.g. the period) of data inputs or from the timed requirements of data outputs (e.g. the latency). Lastly, communication protocols and task scheduling strategies can be derived as a refinement towards an actual implementation.

Key words: real time data, distributed systems, verification

1. Introduction. Distributed Real Time Embedded (DRE) systems are increasingly 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 formal frameworks.

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. Actually, the source is an arbitrary state expression. An observation abstracts the relation between the inputs and the outputs of a communication protocol or between the arguments and the results of a computation.

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, its timeline, an abstraction of its time behavior, 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 by relating these dedicated variables and 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 allow to relate the time behavior of a source and of an image by expressing different properties, such as the time lag 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 stated in terms of timed observations. A specification is consistent if and only if the verification process can construct correct executions.

*An earlier version of this paper was presented at the 3rd Real-Time Software Workshop, RTS2008, in Wisla, Poland, October 20, 2008.

†Université de Toulouse—IRIT, 2, rue Charles Camichel, 31071 Toulouse, FRANCE {tleberre, mauran, padiou, queinnec}@enseeiht.fr

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 bounds of the time constraints.

2. 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 the transit of data. Most approaches taken to check timed properties of distributed systems are based on studying the timed *behavior* of tasks. For example, works such as [10] 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 values of the system variables. We depart from scheduling analysis by focusing on the variables behavior and not considering the tasks and related system events. Our intent is to allow the developer to give a more declarative statement of the system properties, easier to write and less error-prone. Indeed, reasoning about state predicates is usually simpler than reasoning about a set of valid sequences of events.

Others approaches based on variables are mainly related to the field of databases. For example, the variables semantics and their timed validity domain are used in [12] to optimize transaction scheduling in databases. Our work stands at a higher level since we propose to give an abstract description of the system in terms of a specification of relations between data. For instance, our framework can be used to check the correctness of an algorithm with regards to the aging of the variables values. It can also be used to specify a system without knowing its implementation.

Similar works use temporal logic to specify the system. For example, in [2], OCL constraints are used to define the temporal 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 [9], constraints between intervals during which state variables remain stable are defined by means of Allen's linear temporal logic. 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, the authors expect to apply this approach 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 the required timed properties of the system.

3. Theoretical settings.

3.1. State Transition System. Models used in this paper are based on state transition systems. Our work uses the TLA+ formalism [7], but this paper does not require any prior knowledge of TLA+. A *state* is an assignment of values to variables. A *transition relation* is a predicate on pairs 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 the 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.

3.2. 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 [1]: 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 [8]). 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 3.1 (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.

3.3. 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 all functions whose domain is X and whose range is any subset of Y.

Definition 3.2 (Clock) *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:*
  $\forall t, t' \in \mathcal{D} : t < t' \Rightarrow c(t) \leq c(t')$
- *It is lively:*
  $\forall t \in \mathcal{D} : \exists t' \in \mathcal{D} : c(t') > c(t)$

The predicate $clock(c)$ is true if the function c is a clock.

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.

4. 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 variables, that is properties of the history of data, is introduced. We then extend the observation relation to enable the expression of timed constraints on the behavior of system variables linked by observations. For that purpose we define predicates which bind and constraint relevant instants of the timeline of the source and the image of an observation. These predicates are expressed as bounds on the difference between two relevant instants.

4.1. The Observation Relation. We define an observation relation on state transition systems as in [5]. The observation relation is used to abstract a value correlation between variables. Namely, the observation relation states that the values taken by one variable are values previously taken by another variable or state expression.

In the basic case, 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. This definition is actually given to allow any state expression (a formula on variables) as the source[1]. The formal definition is:

Definition 4.1 (Observation) *The variable $`x$ is an observation of the state expression $e$ in execution $\sigma$: $\sigma \vDash `x \prec e$ iff:*

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

[1] As we could introduce a new variable aliased to this expression, we often talk, in the following, of the source *variable*. This is to simplify the wording and the description.

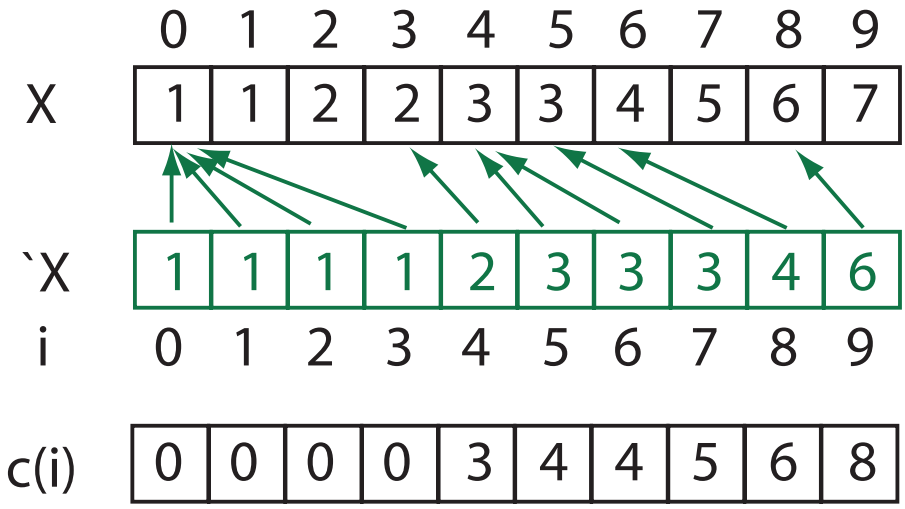

FIG. 4.1. *The Observation Relation*

This relation states that any value of $`x$ is a previous value of $e$. Due to the properties of the observation clock $c$, $`x$ is assigned e values in accordance with the chronological order. Moreover, c always eventually increases, so $`x$ is always eventually updated with a new value of $e$. Figure 4.1 shows an example of an observation relation binding two variables $x$ and $`x$.

The observation can be used to abstract communication in a distributed system, as well as to abstract computations:

- 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 $remote \prec local$.
- In state transition systems, an expression $f(X)$ models an instantaneous computation. By writing $y \prec f(X)$, we model the fact that a computation takes time and that the value of y is based on the value of X at the beginning of the computation. Here X can be a tuple of variables, according to the arity of f: given $X = \langle x_1, \ldots, x_n \rangle$, the observation $\sigma \models `x \prec f(X)$ means that $\exists\ c \in [\mathbb{N} \to \mathbb{N}] : clock(c) \wedge\ \forall i : `x.\sigma_i = f(x_1.\sigma_{c(i)}, \ldots, x_n.\sigma_{c(i)})$. As the same clock is used, all values of the inputs (X) are read at the same time, implying a synchronous behavior.
  Additional observation relations can be introduced to model an asynchronous reading of the inputs. For instance, $`a \prec a, `b \prec b, c \prec f(`a, `b)$ models a system where a and b are independently read (the first two observations), and then c is computed through a function f.

Note that the observation definition does not refer to real time and only models an arbitrary delay in terms of state sequences. Real time properties will now be introduced.

4.2. The Timeline of Variables. In order to state properties about the timed behavior of a variable x, we want to be able to refer to the last time x was updated. These are called the update instants and form its timeline $\hat{x}$. The definition of $\hat{x}$ is based on the history of the values taken by x and captures the instants when each value of x appeared, e.g. the beginning of each occurrence.

Definition 4.2 (timeline) *For a separated execution σ and a variable x, the variable $\hat{x}$ is the timeline of x and is defined by:*

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

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

Note that the developer may provide an explicit definition of $\hat{x}$, without having to describe the actual values of x, e.g. by stating that x is periodically updated.

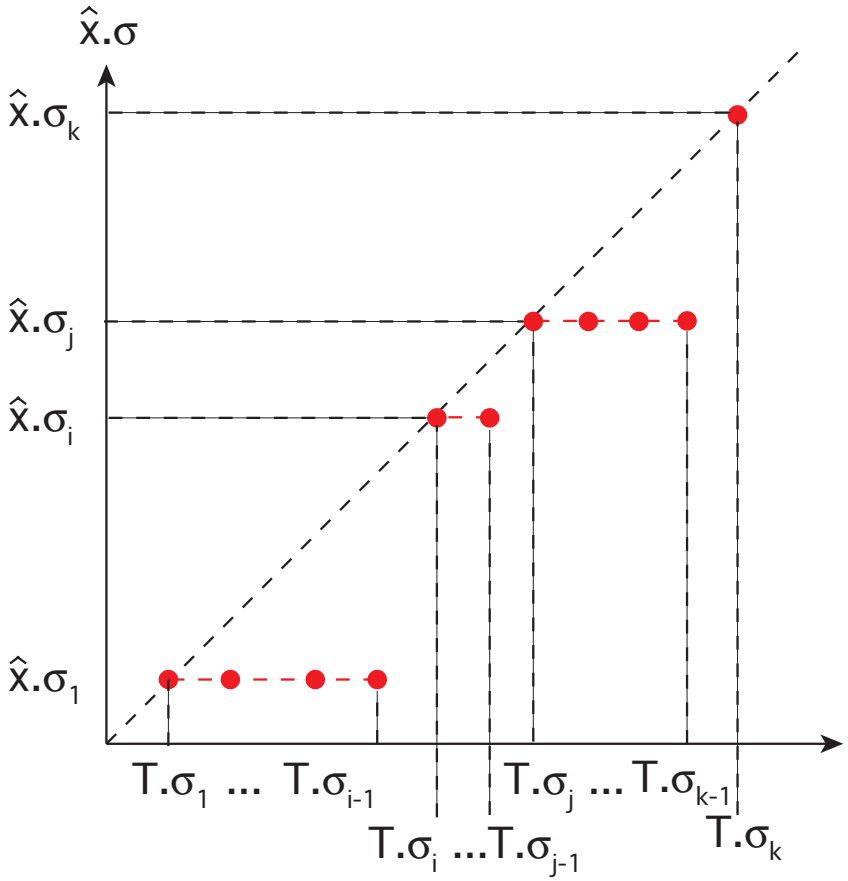

FIG. 4.2. *Graph of* $\hat{x}$

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 exclusively on the values of $\hat{x}$ to study the timed properties of x.

We also define the instant $Next(\hat{x})$ that returns, at each state, 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 at a state σ_i (no new update), then $Next(\hat{x}).\sigma_i = +\infty$.

As in the case of source variable versus source expression, the definition of a timeline $\hat{x}$, which is given for a variable x, is actually valid for a state expression. For the sake of clarity, we will once again talk of "variables" where "state expressions" could equally be used in the remainder of this section.

4.3. Behavior of Variables. The timeline $\hat{x}$ is used to describe the timed behavior of a variable x. In this paper, we focus on specific kinds of variables. We expect each value of each variable to remain unchanged for a bounded number of time units. We want to be able to express the minimum and the maximum duration between two consecutive updates. This allows to describe two basic behaviors: a sporadic variable keeps each value for a minimum duration, and on the contrary, a lively variable has to be updated often, no value can be kept longer than a given duration. These properties are formulated by bounds on the difference between $\hat{x}$ and $Next(\hat{x})$, using a property called *Steadiness* applied to a variable. These bounds denote how long each value of x can be kept.

Definition 4.3 (Steadiness) *The steadiness of a variable* x *in the range* $[\delta, \Delta]$ *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. More elaborate properties can be derived from the steadiness property. For example, we can introduce a stronger property, periodicity, where no time drift is allowed.

Definition 4.4 (Periodicity) *A variable x is periodic of period P with jitter J and phase ϕ iff:*

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

Such a variable is updated around all instants $\phi + nP$. Note that J must verify $J < P/4$ to ensure that the variable is updated once and only once per period.

4.4. Timed Observation. We use the concept of timeline 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 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 4.5 (Timed Observation) *A timed observation is defined as an observation satisfying a set of predicates.*

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

The predicates that can be used to describe the timed properties of the relation between two variables are the following ones:

Definition 4.6 *Given a variable $`x$ and a state expression $e$ such that $\sigma \vDash {}`x \prec e$ with a clock $c \in [\mathbb{N} \to \mathbb{N}]$, the predicates are:*

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

When no lower (resp. upper) bound is significant, 0 (resp. $+\infty$) should be used.

These predicates have to be true at every state and every instant. The definition of an observation is done by stating which predicates must be satisfied. So far, this set has been sufficient to express the different behaviors that we had to analyze, but it can be extended.

- Predicate *Lag* is used to bound the duration between an update of the source and an update of the image. An upper bound states that, when the image is updated, it must be updated with an expression of source that was updated in a recent time. A lower bound states that when there is an update of the source, the new value cannot be used to update the image before the lower bound has elapsed.
- Predicate *Latency* bound in each state the time elapsed since the assignment of the image's current value on the source.
- Predicate *Stability* is used to filter sources values depending on their duration. For example we can eliminate transient values and keep sporadic ones, or the contrary.
- The observation clock and the difference $i - c(i)$ give the logical delay introduced by the observation. Predicate *Medium* bounds the temporal delay related to this logical delay. So the bounds state that there must exist a logical delay inducing a temporal delay satisfying the bounds, i. e. in each state, there must be one previous state so that the time elapsed since that state is below this upper bound and above the lower bound and so that the image's current value was assigned on the source. A lower bound can be used to state that a value of the source cannot appear on the source before this lower bound has elapsed and so this bounds denotes a communication or computation time.

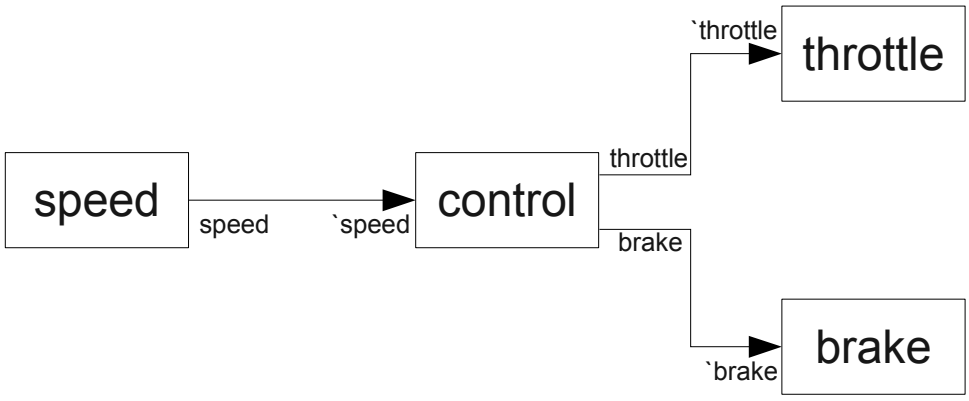

FIG. 5.1. *Cruise Control System*

- Predicates *Freshness* and *Fitness* are used to define intervals of time, relative to the update instants, that the observation clock is prevented to refer. So the logical delay that satisfies the predicate Medium must refer to an instant that satisfies the Freshness and Fitness predicates. An upper bound on Freshness prevents states where the value of the source is not fresh anymore to be referred. For example, a conjunction of Medium and Freshness predicates states that the current value of the image must have been available on the source recently and that it was still fresh at these instants. On the contrary, a lower bound on Freshness denotes an impossibility to access a value just after its assignment. Fitness allows or forbids the states depending on the time remaining until the source value is updated. A lower bound prevents to refer to a state where the value is about to be updated.

Note that, at the beginning of an execution, some predicates such as *Medium* cannot be satisfied. In order to address this problem, the 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 [6].

5. Specifying a System in Terms of Timed Observations.

5.1. A Brief Description. As an example, we consider a simplified car cruise control system. The goal of such a system is to control the throttle and the brakes in order to reach and keep a given target speed. The system is composed of several interacting components (see Figure 5.1):

- 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 down 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 target speed is chosen, the control system can choose either to accelerate by increasing the voltage of the throttle actuator or to decelerate by decreasing this voltage and by using brakes. In order to ensure a reactive behavior, each command issued by the cruise control system must be carried out within a given time limit.

Each component uses and/or produces data. We use observations to specify the system and characterize correct executions.

5.2. Data and Observations. Firstly, we define the state variables of the cruise control system, and we bind these variables using observation relations.

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 choices 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 by the throttle actuator. Using the speed values, we compute the values of two variables: $throttle \prec control1(`speed)$ and $brake \prec control2(`speed)$.

Lastly, the values of *throttle* and *brake* are delivered to dedicated devices into variables $`throttle$ and $`brake$, such that $`throttle \prec throttle$ and $`brake \prec brake$.

- variables behaviors:

$$speed\ \{Steadiness(\delta_1, +\infty)\}$$
$$throttle\ \{Steadiness(\delta_2, +\infty)\}$$
$$brake\ \{Steadiness(\delta_3, +\infty)\}$$

- 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:

$$`throttle \prec control1(speed)\ \{Latency(0, \Delta)\}$$
$$`brake \prec control2(speed)\ \{Latency(0, \Delta)\}$$

FIG. 5.2. *System Specification*

5.3. Requirements and Properties. We express the requirements and known timed properties of the system, and we state them as characteristics of the system variables and observations. These characteristics are given in Figure 5.2.

The speed is computed using the ratio of the number of wheel turns to the elapsed time. A minimum time is required to produce a significant result. Thus, there must be a minimum time δ_1 between each update of *speed*. Also, due to scheduling constraints, there must be a minimum time δ_2 (respectively δ_3) between each computation and update, of *throttle* (respectively *brake*).

Each communication on the bus takes a minimum transit time, regardless of the communicating protocol that is chosen. Predicate *Medium* (see Definition 4.6) is used to define a lower bound on the observations expressing communication. Similarly, we represent the minimum computation time of functions *control*1 and *control*2, by means of predicate *Medium*.

We expect each data to be used soon enough after each update. More precisely, we want each command issued 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 completed in a short enough time.

A composition of observations is an observation, for example if $y \prec x$ and $z \prec f(y)$ then $z \prec f(x)$ [5]. We use this property to define the processing chains relating *`throttle* and *`brake* to *speed*, via *'speed* as observations, which enables us to express the requirements on the duration of the processing chains as upper bounds of *Latency* predicates (see Definition 4.6) on these observations. Note that, although the *Latency* upper bound (Δ) is the only upper bound given in the system specification, it implicitly sets upper bounds on the *Medium* and *Steadiness* characteristics of the other observations and variables of valid executions.

5.4. 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, a nonempty set of valid executions ensures the availability of timely sound values. From this set, we can 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. Then, we determine the possible values of the observation clocks in the states corresponding to the timeline 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 when 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 when the message must be sent.

6. 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 [8] for example.

6.1. Feasibility of a Specification. 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 model-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 original 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 timeline variables 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 timelines 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 timelines 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 timelines 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.

6.2. 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 observations and variables properties. To solve this set of inequalities and deduce the possible values of instant variables in the new state, we use difference bound matrices [4]. Considering a system where n variables are studied, the size of each matrix is $O(n^2)$, and the complexity for reducing it to its canonical form and building the new state is $O(n^3)$ [4]. 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 $O(n^3 * L^n)$. 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. The complexity is more heavily impacted by the number of variables (n) than by the analysis interval (L).

6.3. Verification of an Implementation. A second goal is to check that an implementation is correct with regard to a specification based on observations. This approach is fully described in [13] and is only hinted here.

As all timed properties are safety properties, an implementation is correct if no execution deadlocks (so as to ensure liveness) and all its executions are included in the executions defined by the specification.

In order to check the satisfaction of the specification by an implementation, we give a model of the specification in the same semantics we use to model an implementation. Such a model is described by defining elementary transitions. An elementary transition relation models the evolution of the values states of availability in the observation relations of the system. These elementary transition relations are used to build the variable transition relation of the image of an observation. The variable transition relations are then used to build the global transition relation.

Once both the specification and the implementations have been translated into such transition relations, we must verify that the model of the specification simulates the implementation. In order to check this property, we build a state transition system similar to the synchronized product of labelled transition systems. The actions are used as labels on the transitions of the systems.

6.4. 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 $\exists$, 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 [3]. Properties of the observation can be expressed as safety properties using LTL and be derived as Büchi automata [11]. 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.

7. 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 expressiveness. We also work on using analysis results to help generating an implementation satisfying the specification.

REFERENCES

[1] M. Abadi and L. Lamport, *An old-fashioned recipe for real time*, ACM Transactions on Programming Languages and Systems, 16 (1994), pp. 1543–1571.

[2] 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.

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

[4] J. Bengtsson and W. Yi, *Timed automata: Semantics, algorithms and tools*, in Lecture Notes on Concurrency and Petri Nets, W. Reisig and G. Rozenberg, eds., Lecture Notes in Computer Science vol 3098, Springer–Verlag, 2004.

[5] M. Charpentier, M. Filali, P. Mauran, G. Padiou, and P. QuÃľinnec, *The observation : an abstract communication mechanism*, Parallel Processing Letters, 9 (1999), pp. 437–450.

[6] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, *The synchronous data-flow programming language LUSTRE*, Proceedings of the IEEE, 79 (1991), pp. 1305–1320.

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

[8] E. Lee and A. Sangiovanni-Vincentelli, *A framework for comparing models of computation*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17 (1998), pp. 1217–1229.

[9] 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), no. 4144 in Lecture Notes in Computer Science, Springer Verlag, 2006, pp. 263–277.

[10] 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), 40 (1994), pp. 117–134.

[11] 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.

[12] 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.

[13] Tanguy Le Berre, Philippe Mauran, Gérard Padiou, and Philippe Quéinnec, *A data oriented approach for real-time systems*, in IEEE Int'l Conf. on Real-Time and Network Systems RTNS09, 2009.

Edited by: Janusz Zalewski
Received: September 30, 2009
Accepted: October 19, 2009