Ensuring Timed Validity of Distributed Real Time Data

Tanguy Le Berre, Philippe Quéinnec, Gérard Padiou

Ensuring Timed Validity of Distributed Real Time Data

Tanguy Le Berre, Philippe Quéinnec, Gérard Padiou
Université de Toulouse
INP-IRIT
CNRS UMR 5505

***Abstract*—The goal of this paper is to study timed requirements on distributed real time data. More precisely, we want to ensure that values used for computation are valid considering timed requirements. For that purpose, we model relations and interactions between data and define timed properties and requirements on those relations.**

The possibility is given to express different requirements on data by considering the update times of variables. For example, we expect a value to be fresh when used or we filter transient values or sporadic values. Moreover we generally characterize the behaviour of a variable depending on the properties of any of its values.

Communication in distributed systems introduce a latency and a timed indeterminism incompatible with real time systems requirements. We define an abstract communication mechanism as a relation between a couple of variables: a source and an image. To introduce timed properties, we characterize the communication delay by binding the time shift between the source and the image.

The properties of the model are then used to check that, during any execution, timely values are always available (safety) or available in enough states (liveness).

1. Introduction

Embedded systems are more and more current in everyday's life and in the same time become more complex. One factor of the complexity increase is the use of distributed architecture. In such systems, an important aspect is the verification of timed properties and most researches are done on the task schedulability and on the synchronization of these tasks.

In this paper, we use another approach for the problem of timed properties verification by using a system model focused on the system variables and not on the tasks. The variables are the products of the tasks execution, the interdependency of the tasks is bound to the semantics which links the different variables; for instance the value of a variable A is needed to compute the value of a variable B. We here check the timed validity of used values. For that purpose, we must first specify the properties of variables and of relations between variables. For example, systems modeled here are distributed system so the latency introduced by communication should be taken into account. The model of the system is used to check the properties of all variables used and produced by a distributed system: for example timed requirements specified like the freshness and the stability of values must be satisfied.

2. Related Works

Most approaches done to check timed properties of distributed systems are based on studying the timed behaviour of tasks. For example, works like [1] try to include the timed properties of communication in classical scheduling analysis. We diverge from this work by focusing on variables and their values instead of tasks.

Others approaches based on variables instead than being based on the process computing these variables values are mainly done in the field of database. For example, the variables semantics and their timed validity domain are used in [2] to optimize the transaction scheduling in databases.

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. Instead of formally defining synchronization mechanisms, we use an abstraction of communication to check the variables properties.

A closed work was done in [4] where a grammar able to describe the timed behaviour of a variable is defined. Rules are given to combine the properties of the different variables of the system depending on their relations.

3. Context

3.1 State Transition System

Models used in this paper are based on state transition systems. More precisely, this paper use the

semantics of TLA+ [5]. A *state* is an assignment of values to variables. A *transition relation* is predicate on pair of states. A *transition system* is a couple (set of states, transition relation). A *step* is a pair of states which satisfy the transition relation. An *execution* σ is a infinite sequence of states $\sigma_0\sigma_1 \dots \sigma_i \dots$ such that two consecutive states form a step. We note $\sigma_i \rightarrow \sigma_{i+1}$ a step between the two consecutive states σ_i and σ_{i+1}.

A *temporal predicate* is a predicate on execution; 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 set of values taken by e during an execution σ is noted $e.\sigma$. A *state predicate* is a formula whose meaning is a boolean-valued expression on states.

3.2 Time

We want to study real time properties of the system. To distinguish them from the (logical) temporal properties, such properties are called *timed* properties. Time is integrated in our transition system in a simple way, as described in [5]. 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. It makes no difference whether time is dense (real) or not (natural). Moreover, it makes no difference whether time is continuous or discrete. However, as an execution is a sequence of states, the actual sequence of values taken by T during a given execution is necessarily discrete.

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. Any change in the system implies time passing.

Definition 1: *Separation. An execution σ is separated iff for any variable x:*

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

In the following, we only consider separated executions. This allows to timestamp changes of variables.

3.3 Clocks

Let's consider a totally ordered set of values $\mathcal{T}$, such as $\mathbb{N}$ or $\mathbb{R}^+$. A clock is a (sub-)approximation of a sequence of $\mathcal{T}$ values.

Definition 2: *Clock. A function c from $\mathcal{T}$ to $\mathcal{T}$ is a* clock *iff:*

- *a clock never outgrows the value:* $\forall t \in \mathcal{T} : c(t) \leq t$
- *a clock is monotonously increasing:* $\forall t, t' \in \mathcal{T} : t < t' \Rightarrow c(t) \leq c(t')$

In the following, clocks are used to characterized the timed behavior of variables. They are defined on the time variable T, to express a time delayed behavior (section 4), as well as on the indices of the sequence of states, to express a logical delay (section 5.1). Two clock subsets are used:

Definition 3: *A clock c from $\mathcal{T}$ to $\mathcal{T}$ is* idempotent *iff:*

$$\forall c : clock(c) : \forall t \in \mathcal{T} : c(c(t)) = c(t)$$

Definition 4: *A clock c from $\mathcal{T}$ to $\mathcal{T}$ is a* liveclock *iff:*

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

4. Timed Properties of Variables

We give here the definition of some timed properties of the system variables. Transition systems do not explicitly mention the events of the system such as read or update events. We focus on the history of values taken by variables in an execution and deduce the interesting timestamps from this history.

4.1 Updates

In order to express properties on the timed behaviour of a variable x, we want to be able to refer to the last time this variable was updated. They are called the update instants $\hat{x}$. Two possibilities exists, this referential can be implicit or explicit. In the explicit case, the developer has the responsibility to give its own variable $\hat{x}$. This can be the case if there is a periodic behaviour 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 values taken by x. The goal is to capture the moment the current value of x appeared, i.e. the beginning of the current occurrence. A clock $\hat{x}_t$ is used to give this definition.

Definition 5: *For an execution σ and a variable x, function $\hat{x}_t$ is defined so that:*

$$clock(\hat{x}_t) \ \wedge \ idempotent(\hat{x}_t) \ \wedge \ \forall \ I = [s, e] : \left(\begin{array}{c} \forall \ t, t' \in (I \cap T.\sigma)^2 : \hat{x}_t(t) = \hat{x}_t(t') \\ \Leftrightarrow \\ \forall \ i, j : T.\sigma_i, T.\sigma_j \in I^2 : x.\sigma_i = x.\sigma_j \end{array} \right)$$

$\hat{x}_t$ is built from the history of x values. When x is updated and its value changes then the value of $\hat{x}_t$ is also updated. Conversely, x changes when $\hat{x}_t$

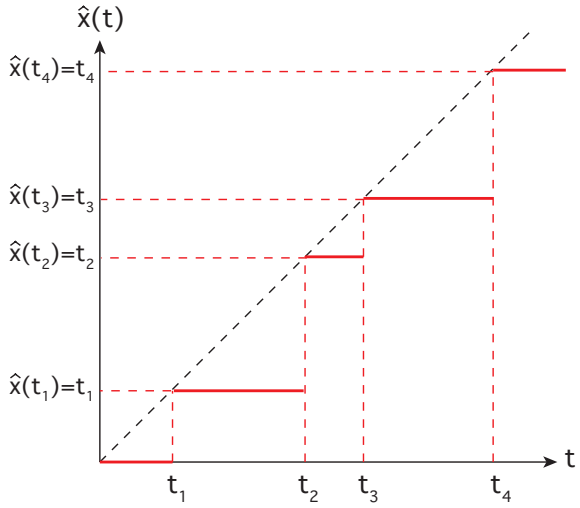

Figure 1: General Aspect of $\hat{x}$

changes. Idempotency ensures the value of $\hat{x}_t$ to be the beginning of the interval when the current value is stable.

Proposition 1: *Given an execution σ and a variable x, $\hat{x}_t$ exists and is unique.*

Existence is due to the use of separated execution and x and unicity is due to idempotency.

Definition 6: *For an execution σ and a variable x, the variable $\hat{x}$ is defined by:*

$$\forall i : \hat{x}.\sigma_i = \hat{x}_t(T.\sigma_i)$$

$\hat{x}$ is used rather than $\hat{x}_t$ in this paper to directly refer to state of transition systems.

4.2 Next

For a given execution, in any state we want to be able to denote the next distinct value a variable takes. So we define a function that can be applied to any expression.

Definition 7: *For an expression e, and a value $none$, the function $Next(e, none)$ is:*

$$Next(e, none).\sigma_i \triangleq \begin{array}{l} \textit{LET } S \triangleq \{j > i : e.\sigma_j \neq e.\sigma_i\} \textit{ IN} \\ \quad \textit{IF } S \neq \emptyset \quad \textit{THEN } e.\sigma_{min(S)} \\ \quad \textit{ELSE } none \end{array}$$

For example, for a variable x, $Next(\hat{x}, +\infty)$ is the instant when x current value changes or is $+\infty$ in case x remains stable.

4.3 Freshness

An interesting property is that a value taken by a system variable is recent enough to be meaningful when used. this property is called freshness and we define a predicate denoting if it is worth using a value with respect to freshness requirements. Given a variable x, the difference $T.\sigma_i - \hat{x}.\sigma_i$ denotes the time elapsed since x last update. So this difference must be upper bounded. On the other hand, giving a lower bound to this difference shows that the value has not changed for some time.

Definition 8: *For a variable x, the state predicate $Freshness$ is:*

$$Freshness_x(\delta, \Delta) \triangleq \delta \leq T - \hat{x} \leq \Delta$$

This predicate is true when the current value of variable x meets the freshness requirement. Here 0 is used when there is no lower bound and $+\infty$ when there is no upper bound.

4.4 Stability

Depending on the modeled system, we may also require a value to remain stable long enough or on the contrary, to be transient. In each state, we define the predicate $Stability$ which, given a variable, states if the current value remains stable or if it is transient according to two parameters.

Definition 9: *For a variable x, the state predicate $Stability$ is:*

$$Stability_x(\delta, \Delta) \triangleq \delta \leq Next(\hat{x}, +\infty) - \hat{x} \leq \Delta$$

Here the difference $Next(\hat{x}, +\infty) - \hat{x}$ is used to compute the time elapsed before the disappearance of the current value. Compared to freshness, this predicate not only consider current instant but the whole interval when the value remain stable.

4.5 Timed Behaviour of a Variable

$\hat{x}$ is used to describe two possible behaviours of a variable: sporadicity and liveness. A variable is sporadic if in any execution each value remains stable for at least some duration.

Definition 10:

$$Sporadicity(x, m) \triangleq \forall \sigma : \forall i : Stability_x(m, +\infty).\sigma_i$$

This ensures that new updates can never occur before m time units has elapsed.

Conversely, we say that a variable meets the liveness property if it is updated in a periodic way. For such a variable, time does not pass too long before the variable is updated.

Definition 11: *Given a system variable x, x satisfies M-liveness:*

$$Liveness(x, M) \triangleq \forall \sigma : \forall i : Freshness_x(0, M).\sigma_i$$

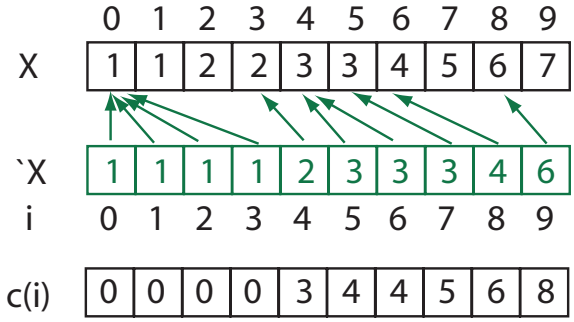

Figure 2: The Observation Relation

So here, there cannot be more than M time units elapsed since the update of the current value.

The difference $M - m$ is the jitter of x. These two properties can be combined to define a strictly periodic variable. In this case no jitter is allowed on the updates. If $m = M$ then necessarily x is updated every m units.

5. Abstracting Communication

5.1 The Observation Relation

The observation relation is an abstraction of communication in a distributed system defined in [6]. This relation binds two variables: the source x and the image $`x$ and denotes that the history of the variable $`x$ is a subhistory of the (remote) variable x. The relation is defined by a triplet $< source, image, liveclock >$ where the clock defines for each instant which one of the previous values of the source is taken by the image. The formal definition is:

Definition 12: *The variable $`x$ is an observation of the variable $x$ ($`x \prec x$) iff:*

$$\forall \sigma : \exists\ c : liveclock(c) \wedge\ \forall i : `x.\sigma_i = x.\sigma_{c(i)}$$

This relation states that any value of $`x$ is a previous value of $x$. Due to properties of $c$, $`x$ is assigned x values in a chronological order. Moreover, c always eventually increases, so $`x$ is always eventually updated with a new value of x. An example of an observation relation is shown figure 5.1.

5.2 Timed Observation

In order to specify timed constraints upon communications, we bind the delays introduced by distribution. For each state i, the difference $T.\sigma_i - T.\sigma_{c.\sigma_i}$ denotes the timed latency between the source and the image. A lower bound on this difference is the shortest time taken by communications. An upper bound prevents the image from drifting with respect to the source. We define

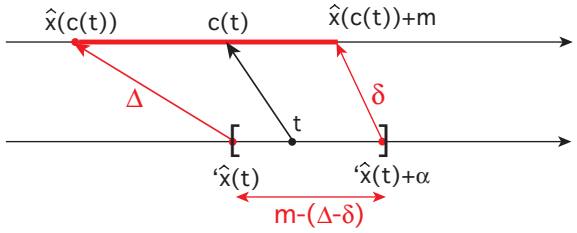

Figure 3: Observation of a Sporadic Variable

the latency predicate and use it to define timed observation:

Definition 13:

$$Latency(c, \delta, \Delta) \triangleq \forall\ i : \delta \leq T.\sigma_i - T.\sigma_{c(i)} \leq \Delta$$

Definition 14: *The variable $`x$ is a timed observation of the variable $x$ with parameter $(\delta, \Delta)$ ($`x \prec_{[\delta,\Delta]} x$) iff*

$$\forall \sigma : \exists\ c : liveclock(c) \wedge \forall i : \begin{pmatrix} `x.\sigma_i = x.\sigma_{c(i)} \\ \wedge\ Latency(c, \delta, \Delta) \end{pmatrix}$$

The difference $\Delta - \delta$ is the jitter introduced by communication.

5.3 Influence of Distribution on Variables Timed Behaviour

Knowing the timed behaviour of a variable, the effect of reading this variable through a network can be defined. We give here properties showing the weakest timed properties for an image given the properties of the source.

Proposition 2: *If there are two variables $`x$ and $x$ so that $`x \prec_{[\delta,\Delta]} x$ then:*

$$Sporadicity(x, m) \Rightarrow Sporadicity(`x, m + (\delta - \Delta))$$
$$Liveness(x, M) \Rightarrow Liveness(`x, M + (\Delta - \delta))$$

To prove this proposition we use the observation relation and the definition of $\hat{x}$ and $`\hat{x}$ to build the shortest and longest stability intervals of $`x$ (see figure 3 for the sporadicity).

The observation weakens the sporadicity and liveness of the source. The local copy of the source is not be as stable as the source and conversely takes a longer time to change. Here, the jitter of the image is the jitter of the source increased by twice the jitter of the observation $(M - m + 2(\Delta - \delta))$.

If $(m + (\delta - \Delta) > 0)$ is positive, then any value taken by x is present in $`x$ history at least $m+(\delta-\Delta)$. Then communication does not enable loss of values.

We can also give necessary conditions on the source properties for the image to be sporadic or liveness :

Proposition 3: *If there are two variables* $`x$ *and* $x$ *so that* $`x \prec_{[\delta,\Delta]} x$ *then:*

$$Sporadicity(`x, m) \Rightarrow Sporadicity(x, m + (\delta - \Delta))$$
$$Liveness(`x, M) \Rightarrow Liveness(x, M + (\Delta - \delta))$$

These conditions do not imply sufficient conditions but on the contrary just show the impossibility of having an image fulfilling the sporadicity or liveness requirements without further conditions on the source and the communication.

5.4 Filtering the Values

When the source do not satisfy the sporadicity or liveness requirements, we extend the observation relation by filtering the source values. Values that do not satisfy the requirements cannot be seen. For example, in a state i the image is assigned the value taken by the source in state $\sigma_{c(i)}$. If this value is not fresh enough, $x.\sigma_i$ is replaced by $\perp$. Filtering can also be used to keep values according to a criterion of stability.

There are four cases:

- the value is not fresh enough
- the value can not be seen just after an update (for example in order to synchronize different images)
- the current value is transient and only stable versions are interesting
- only transient values are interesting and the current one is stable

6. Case Study

In order to illustrate the properties defined in this paper, we rely on a simple example of a distributed system. Our example is a meteorological station. Two sensors measure two phenomena: the wind speed and the temperature. Due to the specificities of each measures, the sensors are distributed. A station collects the measurement and processes them, the packed results are then sent to a central station, for example, to collect all measurement in an area. The system topology is given on figure 4.

The temperature may not be stable in front of short modification of the environment but the station is only interested in stable values. Due to the distribution of the system, there is a delay between the sensors computation of data and their availability at the station. For each variable, there are freshness requirements depending on the variable and their use.

We call $sensW$ the variable assigned the wind speed measures, $sensT$ the one assigned the temperature measures. The couple $(statW, statT)$ is an

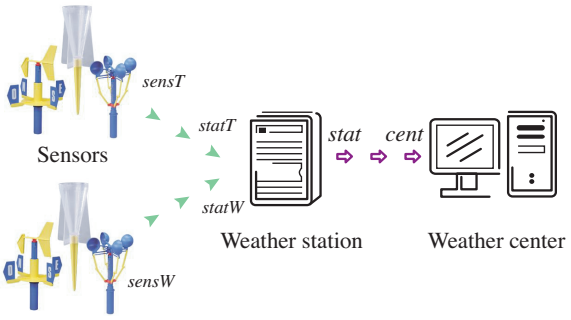

Figure 4: System Topology

observation by the station of the sensors measures. This couple is packed to create the variable $stat$. This variable is bound to the variable $cent$ which is an observation of $stat$ in the central station. Timed observations are used to describe more precisely relations between the variables of this distributed system.

6.1 Relation $statW \prec sensW$

This relation is characterized by three parameters:

- δ_1 and Δ_1 the bounds on the observation latency;
- Δ_2 the upper bound of the freshness limiting the oldness of available values.

6.2 Relation $statT \prec sensT$

This relation is similar to $statW \prec sensW$ except that there is also a criterion on the value stability so there are four parameters:

- δ_3 and Δ_3 the bounds on the observation latency;
- Δ_4 the upper bound of the freshness limiting the oldness of available values;
- δ_4 the lower bound of the stability to keep stable values only.

6.3 Relation $cent \prec stat$

This relation, such as $statW \prec sensW$ is characterized by three parameters:

- δ_5 and Δ_5 the bounds on the observation latency;
- Δ_6 the upper bound of the freshness limiting the oldness of available values?

6.4 Additional Information

In order to study the characteristics of $cent$ and $stat$, the timed behaviours of $sensW$ and $sensT$ should be known. So either an explicit description of $\widehat{sensW}$ and $\widehat{sensT}$ is given, either there is a representative sample of value taken by these variables which allows us to implicitly compute the update instants.

7. Analysis

Analysing a system such as the meteorological station can be done with two different objectives:

- any values of a variable always meets the requirements. Considering filtering, the $\perp$ value is never taken. This is a safety property.
- define the set of states where the value is not $\perp$. This is a liveness property where it must be proved that infinitely often there are values other than $\perp$, so there are an infinite number of states meeting the requirements.

Both of these analysis can be done for a restricted part of the system or considering all variables at once. Analysis is based on two methods: model checking and use of proofs. Both strategies are explained with their pros and cons.

7.1 Proof Strategy

The proof strategy is not based on building a proof for a given system but on reusing proposition to deduce system properties. For example, here, the characteristics of $sensW$ and $sensT$ are used (more precisely of $\widehat{sensW}$ and $\widehat{sensT}$) to derive automatically the properties of the other variables. Let consider the couple $(sensW, sensT)$. Suppose both variables have properties of sporadicity. Then properties of $stat$ and $cent$ are derived and their sporadicity characteristics are deduced. Moreover if the value of $(\delta_1+\delta_5-\Delta_1-\Delta_5)$ or $(\delta_3+\delta_5-\Delta_3-\Delta_5)$ are smaller than the sporadicity characteristics of $sensW$ and $sensT$, all observation are loss free. Then each value taken by the inputs is taken by $cent$.

These results can only be derived in certain cases as the properties deal with the worst cases. In case this method is not sufficient model checking is used instead of building a proof dedicated to a particular model.

7.2 Model Checking

In most cases, such proof cannot be automated and become cumbersome. This does not imply the system does not meet the requirements. In this case, using model checking is proposed as an automated way to analyse the system.

Most of the system properties are here given as bound on differences. Therefore, difference bound matrices are used as in [7] and in Uppaal [8]. Difference bound matrices are used to check that a set of inequalities does not imply a contradiction. For that purpose, matrice multiplication in a $\{min, +\}$ algebra is used. Each multiplication is equivalent to a search for shortest paths or here for tighter bounds. Incompatible inequalities are detected when a positive inequality becomes upper bounded by a negative value. If no error is detected, each inequality is reduced to its minimal form and a fix point (the canonical form of the problem) is found.

The size of a matrice is related to the number of inequalities and so to the number occurrences requested for a sample of value to be representative of a full execution. A set is considered to be representative if it is similar to a prefix where a loop can be built. Given n the size of a set, this analysis has a n^3 complexity. Before using DBM, different sets of inequalities must be built. Some inequalities such as inequalities derived from freshness requirements are bound to the update instants. So a set of inequalities exists for each possible value of these update instants. This number of inequalities sets depends on the jitter on these update instants and is roughly n^2 if the size of each set is n. Finally, the complexity of checking a full model is n^5.

Back to the case study, model checking can be used to check that values assigned to $cent$ meets the requirements. For example, the proof strategy can be used to prove that there is no loss except those due to stability requirements and transient values. In order to be able to filter transient values the availability of $sensT$ values is delayed. It is to say, the values are filtered using a lower bound on the freshness predicate. In those conditions,the following checks can be performed:

- there are not only stable value available and transient ones are filtered;
- there exists states where stable values are available and still fresh.

So it checks that there are enough states where the requirements are not incompatible.

Another example concerns the couple $(sensW, sensT)$. The relations $statT \prec sensT$ and $statW \prec sensW$ have different parameters. The variable $cent$ is assigned a couple of values obtained through this two relations. The model is checked to determine if a value assigned to $cent$ satisfies the requirements in the same state, i.e. there are states where none of the values of the couple $cent$ has been filtered and is equal to $\perp$.

7.3 Perspectives

Due to the complexity of model checking, analysis of a model using this method is long. A perspective is to reduce this complexity by decreasing the number of sets to check. Propositions are used to analyse

the model and deduce properties of the variables. In this case the properties and the proved propositions are used to avoid considering a case (i.e. a set of inequalities) leading to a contradiction. For example the necessary conditions given in 5.3 are used to eliminate sample inputs that can not lead to a system meeting the requirements.

8. Conclusion

We proposed an approach focused on variables instead of task and process to model and analyse distributed real time systems. Based on the state transition system semantics extended by a timed referential, we express timed properties of variables, and of communications. These properties are used to check the freshness of values, their stability and the compatibility of requirements. The analysis is done using propositions to derive simple proof or in more complex case using model checking. The complexity of model checking is a problem when analysing large systems so we work on combining proofs, to reduce the problem size, and model checking, to easily analyse simple models. For that purpose we will extend the number of properties and of proved propositions binding these properties.

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. [Online]. Available: citeseer.ist.psu.edu/tindell94holistic.html

[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 (RTSS '96)*, 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] D. Delfieu, "Expression et validation de contraintes temporelles pour la spécification des systèmes réactifs," Master's thesis, Université Paul Sabatier - Toulouse III, 1995.

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

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

[7] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools." [Online]. Available: http://citeseer.ist.psu.edu/703625.html

[8] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems," in *Proc. of Workshop on Verification and Control of Hybrid Systems III*, ser. Lecture Notes in Computer Science, no. 1066. Springer–Verlag, Oct. 1995, pp. 232–243.