Loosely Time-Triggered Architectures based on Communication-by-Sampling*

Albert Benveniste
IRISA/INRIA
Campus de Beaulieu
35042 Rennes cedex, France
benveniste@irisa.fr

Paul Caspi
VERIMAG/CNRS
2 avenue de Vignate
38610 Gières, France
paul.caspi@imag.fr

Marco Di Natale
ReTiS Lab.
Scuola Superiore S. Anna,
Pisa, Italy
marco@sssup.it

Claudio Pinello
Cadence Berkeley Labs
1995 University Ave, Suite 460
Berkeley, CA 94704, USA
pinello@cadence.com

Alberto Sangiovanni Vincentelli
EECS Dept., U.C. Berkeley
Berkeley, CA, USA
alberto@eecs.berkeley.edu

Stavros Tripakis
Cadence Berkeley Labs
1995 University Ave, Suite 460
Berkeley, CA 94704, USA
tripakis@cadence.com

ABSTRACT

We address the problem of mapping a set of processes which communicate synchronously on a distributed platform. The Time Triggered Architecture (TTA) proposed by Kopetz for the communication mechanism of a distributed platform offers a direct mapping that would preserve the semantics of the specification. However, its exact implementation may, at times, be problematic as it requires the distributed platform to have the clocks of its components perfectly synchronized. We propose as implementation architecture a relaxation of TTA called Loosely Time-Triggered Architecture (LTTA), in which computing units perform writes into and reads from the communication medium independently, triggered by local, quasi-periodic but non synchronized, clocks. LTTA offers some of the advantages of TTA with lower hardware cost and greater flexibility. So far LTTA was studied for single directional two-users communications over an LTT bus. General topology was not studied. In this paper we propose a design flow that ensures semantics preservation for an LTT communication network with arbitrary topology. Key elements are two new protocols for clock regeneration and predictive traffic shaping. Our approach relies on a mathematical Model of Communication (MoC) that we describe in detail.

*This research was supported in part by the European Commission under the projects IST-2001-34820 ARTIST and IST-004527 ARTIST2 Networks of Excellence on Embedded Systems Design, by the NSF under the project ITR (CCR-0225610), and by the GSRC.

EMSOFT'07, September 30–October 3, 2007, Salzburg, Austria.

Categories and Subject Descriptors

D.4.7.e [**Real-time systems**]: Embedded systems; F.1.1 [**Theory of computation**]: Computation by abstract devices—*Models of Computation*; C.0.e [**Computer systems organization**]: General—*System architectures, integration, and modeling*

General Terms

Design, Theory

Keywords

Real-time systems, distributed control, time-triggered, loosely time-triggered, MoCC

1. INTRODUCTION

In automotive and avionics applications, the propagation of information from one end to the other of a functional chain is typically implemented by a set of periodically activated tasks and messages. The execution platform is a distributed architecture consisting of several ECUs (Electronic Control Units) connected by buses or an interconnection network. Some of the communications between tasks must guarantee no loss of data. We refer to this constraint as *data (or stream) semantics preservation*. Furthermore, real-time constraints may be defined on the computation and communication latencies.

In this paper we address the problem of mapping the functional requirements onto an implementation platform so that stream semantics preservation constraints are satisfied, timing constraints are met, and appropriate cost functions can be optimized. To this end,we start from

- a set of (possibly interacting) functions. Each function obeys the following model: it proceeds by a (possibly nonterminating) sequence of successive logical reactions composed of a finite set of actions; in addition, timing constraints such as periodicity and/or deadlines can be attached to each action.
- an implementation platform consisting of interconnected ECUs, protocols for accessing the interconnection

network and middleware for each ECU. The platform is characterized by its performance in terms of timing, capacity, power and cost.

Then, we perform mapping of the functions onto the implementation architecture so that the constraints are verified and cost functions defined on the implementation architecture can be minimized.

The mapping problem is complex since the limitation of the implementation platform may make stream semantics preservation difficult to achieve. Often, when the implementation platform is event-based, guaranteeing that the logical and timing constraints are satisfied is an unsolved problem or requires a large overhead. A possible solution to this problem is to select the implementation architecture so that some of the constraints are satisfied by construction and the analysis can be carried out using formal methods.

The Time Triggered Architecture (TTA) proposed by Kopetz in [9] consists in implementing, on a distributed hardware, the real-time periodic synchronous model. However, this approach carries cost and timing penalties that may not be acceptable for some applications. Hence, there has been growing interest in less constrained architectures such as the Loosely Time-Triggered Architecture (LTTA) used in the aerospace industry and studied in [3, 15, 10, 11, 2, 1]. LTTA is characterized by the following features:

- access to the communication medium occurs quasi-periodically, using the different local clocks; while not synchronized, these clocks are bound to deviate from each other with limited drift and jitter. We call such clocks *quasi-periodic*.
- writings and readings are performed independently at all nodes connected to the medium in synchrony with the above mentioned local clocks;
- the communication medium behaves like a shared memory, i.e., values are sustained and are periodically refreshed, based on a local clock owned by the medium; how multi-user access to the communication medium is performed, is left unspecified.

The LTTA mechanisms can be either implemented in customized hardware, or built on top of existing distributed execution infrastructures — for example, CAN based networks as shown in [6].

The LTTA mechanism has not been fully characterized nor a complete design flow with potential design space exploration has been offered. In [18, 20, 19, 2] protocols for time-sensitive and LTT architectures were proposed to compensate for a change in latencies, from specification to implementation. In [3], sufficient conditions were given to ensure preservation of stream semantics. However, the results of [3] are specific to single-user case and provide no basis for a systematic extension to multi-user, multi-bus, communication. Recent work [6] extends the work of [3] by showing how cascade LTT communication can be implemented on top of a CAN based architecture.

In this paper, we propose a comprehensive design flow that maps functional requirements onto an LTTA with arbitrary topology that can be implemented in a variety of ways including a set of LTT buses. The design flow is guaranteed to produce a semantic preserving implementation if appropriate assumptions are satisfied.

The paper is organized as follows: we first present a mathematical toolkit in Section 2 that details the model of computation used throughout the paper. Communication by Sampling is studied in detail in Section 3. Following this study, we propose a Platform Based Design approach [17] to map a set of functional requirements on LTTA potentially implemented with an LTT network (LTTN) offering CbS communication links. In particular, in Section 4 we present the LTTN and two protocols that guarantee semantics preservation on the mapping of the requirements onto the network. Additional results regarding the deployment of LTTN over LTT busses, multiple access, and fault tolerance, are found in [16].

2. MATHEMATICAL TOOLKIT

In this section we develop the toolkit we shall use throughout this study.

2.1 An algebra of flows, daters, and counters

A mathematical model for CbS must handle flows, defined as successive dated occurrences of valued events. We first present the corresponding material, by building upon the pioneering work [4]. Symbol $\mathbb{N}$ denotes the positive integers: $\mathbb{N} = 1, 2, 3, \ldots$. Formally, *flows* are infinite sequences $e = (e_n)_{n\in\mathbb{N}}$ of *valued and dated events*.

The *value* of event e_n is denoted by ν^e_n. Unless ambiguity can result from such an overloading, we shall simply write e_n instead of ν^e_n. We call *stream of* e the sequence of values $(\nu^e_n)_{n\geq 0}$, where, by default, $\nu^e_0 = \star$, where symbol "$\star$" means *undefined*. A *clock* is a flow with values in the singleton set $\{\text{tick}\}$.

The *dater* of e is a sequence t^e such that $t^e_n \in \mathbb{R}_+$ is the date of e_n; denote by T^e the set of all dates of events of flow e. When the considered flow is a clock κ, by abuse of notation we also denote by κ its set of dates, instead of T^κ. Accordingly,

$$\text{we shall write } \kappa \subseteq \kappa' \text{ to mean } T^\kappa \subseteq T^{\kappa'}. \tag{1}$$

The *counter* of e is a non decreasing function $\mathbb{R}_+ \mapsto \mathbb{N}$ defined by

$$c^e_t =_{\text{def}} \text{Card}\{\, n \mid t^e_n \leq t \,\} \tag{2}$$

and we define the *strict counter* of e by

$$c^{e^-}_t =_{\text{def}} \lim_{s \nearrow t} c^e_s = \text{Card}\{\, n \mid t^e_n < t \,\} \tag{3}$$

The dater and the counter of a flow carry the same information. The counter can be obtained from the dater as shown above. Alternatively, the dater can be obtained from the counter as follows.

$$t^e_n = \min\{t \mid c^e_t = n\} \tag{4}$$

To be able to refer to "the date of the nth event", or "the index of the last event before t", or "the last value before t", etc, we will need in the sequel to compose the above operators. The following generic notation will be used for this purpose: the composition $t^e \circ c^e$ is defined by

$$(t^e \circ c^e)_t =_{\text{def}} t^e_{c^e_t}$$

Formally, $t^e \circ c^e$ is the composition of the two functions $t \mapsto c^e_t$ and $n \mapsto t^e_n$. To simplify the notations,

$$\text{we shall write } t^e \circ c^e_t \text{ instead of } (t^e \circ c^e)_t.$$

Thus, $t^e \circ c^e_t$ delivers, for flow e, the date of the event whose index is the last before or including t.

It will be at times needed to reason about delayed flows. Flows can be delayed, both logically (by passing them through a k-step shift register) and physically (by delaying the date of event occurrences). Delaying flow e logically by an amount of k is modeled by considering that date t^e_n of the nth occurrence of e is in fact the date of the $(n-k)$th event of the delayed flow e'; that is, e' is characterized by the delayed dater $t^{e'} = t^e \circ (Id + k)$, where Id is the identity function and the context allows to determine whether it operates on times or on indices. Thus, $t^{e'}_n = t^e_{n+k}$.

Similarly, delaying flow e physically by t means that the new flow e' is characterized by the dater $t^{e'}_n = t^e_n + t$. This can be modeled by considering that the number c^e_s of occurrences of e before time s is in fact the number of events of e' at time $s + t$; that is, e' can be characterized by the delayed counter $c^{e'} = c^e \circ (Id - t)$.

So far we discussed only daters and counters. The following macros are useful when considering values.

Interpolating flows. It will be useful to interpolate values between the occurrences of successive events. This is simply achieved by overloading the "value" operator ν^e_n:

$$\forall t \in \mathbb{R}_+ : \nu^e_t \quad =_{\text{def}} \quad \nu^e \circ c^e_t$$

The following *current* operator[1] is a variant of the former one. It sustains the last value seen in the strict past:

$$\forall t \in \mathbb{R}_+ : \nu^{e^-}_t \quad =_{\text{def}} \quad \nu^e \circ c^{e^-}_t$$

For $t = t^e_n$, we get $\nu^{e^-}_t = \nu^e_{n-1}$ and $\nu^e_t = \nu^e_n$, whereas, for $t \notin T^e$, $\nu^{e^-}_t = \nu^e_t$ holds. When no confusion can result, we shall again use the

simplified notations e_t and e^-_t, instead of ν^e_t and $\nu^{e^-}_t$.

Regarding dates, the operator *last* provides at any time the last occurrence time of the flow:

$$l^e_t \quad =_{\text{def}} \quad t^e \circ c^e_t \tag{5}$$

Combining flows. The operator *when* filters the occurrence of flow e according to some predicate b provided synchronously with e (i.e., $T^b = T^e$):

$$\begin{array}{rcl} e \text{ when } b & = & \widehat{e}, \text{ where} \\ T^{\widehat{e}} & = & \left\{ t^e_n \mid n \in \mathbb{N} \text{ and } \nu^b_n = true \right\} \\ \forall t \in T^{\widehat{e}} : \nu^{\widehat{e}}_t & = & \nu^e_t \end{array} \tag{6}$$

The operator *at* delivers, at each occurrence of some flow κ, the current value of flow e:

$$\begin{array}{rcl} e \text{ at } \kappa & = & \widehat{e}, \text{ where} \\ T^{\widehat{e}} & = & T^\kappa \\ \forall n \in \mathbb{N} : \nu^{\widehat{e}}_n & = & \nu^e \circ c^e \circ t^\kappa_n \end{array} \tag{7}$$

2.2 Semantics preservation

Consider two flows e_1 and e_2. We say that *flow* e_2 *stream preserves flow* e_1 if the following holds:

$$\forall n \in \mathbb{N} : \nu^{e_1}_n \quad = \quad \nu^{e_2}_n. \tag{8}$$

[1] The names "current" and "when" (used later in this text) are taken from similar operators found in the Lustre language [5].

Stream preservation between computing units guarantees that the considered distributed architecture is GALS (Globally Asynchronous, Locally Synchronous), so that techniques from [13, 14, 7] can be used to ensure correct-by-construction deployment of synchronous (or polychronous) specifications.

The so defined stream preservation does not account for timing issues. Strict preservation of timing is too strong and irrelevant for LTTA. Instead, we shall complement stream preservation with bounds on the relative periods and jitters, for the considered flows, in each case.

3. COMMUNICATION BY SAMPLING

Communication by Sampling (CbS) is the only basic building block of our LTT Architecture. CbS involves pairs of the form {writer, reader}. It is formalized using a composition operator $\triangleright$ between flows. First we discuss this operator informally and the define it formally and provide some of its properties. Denote by w and r the flows of writings and readings. Then,

$$w \triangleright r$$

is the flow collecting the successive deliveries, by the reader, of the successive writes. That is:

- if w performs a writing that is overwritten by a subsequent writing before being read by r, then no corresponding event for flow $w \triangleright r$ is produced;
- if r performs a reading that follows a previous reading without having a writing occurring between the two, then no corresponding event for flow $w \triangleright r$ is produced;
- if w performs a writing that is followed by a corresponding reading of r prior to a next writing, then an event for flow $w \triangleright r$ is issued at the time of that reading; equivalently, if r performs a reading that follows a corresponding writing of w, then an event for flow $w \triangleright r$ is issued at the time of this reading.

Operator $w \triangleright r$ is illustrated on Figure 1. Note that, despite the notation, $w \triangleright r$ depends on the flow w of writings, but it depends on r only through its clock κ_r. So, we have

$$w \triangleright r \quad = \quad w \triangleright \kappa_r, \tag{9}$$

which enlightens the causal dependency, from the pair (w, κ_r), to the output stream ν^r.

We shall now formalize this description. To define $w \triangleright r$, we need to define two things: its "timing aspect", that is, its dater; and its "value aspect", that is, its sequence of values.

Timing aspect. The dater of flow $w \triangleright r$ is characterized by its counter:

$$\begin{array}{rcl} & & c^{w \triangleright r}_t \\ & = & \text{Card} \left\{ t^r_k \;\middle|\; \left[t^r_k \le t \right] \wedge \left[c^{w^-}_{t^r_k} - c^{w^-}_{t^r_{k-1}} \ge 1 \right] \right\} \qquad (10) \\ & = & \text{Card} \left\{ t^w_k \;\middle|\; \left[t^w_k \le t \right] \wedge \left[c^r_{\min(t, t^w_{k+1})} - c^r_{t^w_k} \ge 1 \right] \right\} \qquad (11) \end{array}$$

Formula (10) consists in counting the number of reads that are preceded by at least one write. Dual formula (11) consists in counting the number of writes that are followed by at least one read before being overwritten. These two formulas are illustrated in Figure 1.

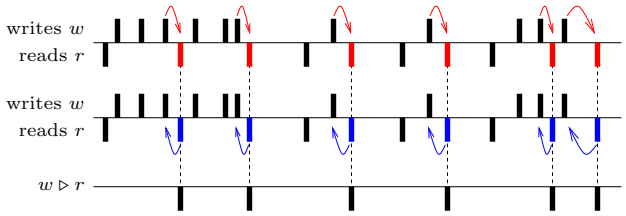

Figure 1: Illustrating: $w \triangleright r$–bottom; formula (10)–middle; and formula (11)–top. The origin of each arrow points: for formula (10), to a read event (in blue) satisfying the associated predicate, and, for formula (11), to a write event satisfying the associated predicate. The end of each arrow points to the event that realizes satisfaction of the associated predicate.

Notice that the dater $t_n^{w \triangleright r}$ can be derived from the counter of $w \triangleright r$ as shown in Equation (4).

Value aspect. Regarding values, the nth value read by the reader is the currently written value at the time of the nth read:

$$\nu_n^{w \triangleright r} \quad = \quad w^- \circ t_n^{w \triangleright r} \tag{12}$$

This completes the definition of flow $w \triangleright r$.

3.1 Effective procedure for computing $c^{w \triangleright r}$, and properties

Using formulas (10) and (11), the following precise description of $c^{w \triangleright r}$ can be given, by switching between these two formulas at appropriate times. To get such a formula, key remarks are:

- Suppose that, for some $t > 0$, condition

$$c_{t_k^r}^{w^-} - c_{t_{k-1}^r}^{w^-} \geq 1 \tag{13}$$

  is satisfied for every $t_k^r \leq t$. Then, applying formula (10) simply yields $c_t^{w \triangleright r} = c_t^r$. We say that such an interval is *of type read*. The set of t's satisfying (13) is an interval of the form $[0, t_{(10)})$, where $t_{(10)} \geq 0$.

- Alternatively, suppose that, for some $t > 0$, condition

$$c^r_{\min(t, t^w_{k+1})} - c^r_{t^w_k} \geq 1 \tag{14}$$

  is satisfied for every $t_k^w \leq t$. Then, applying formula (11) simply yields $c_t^{w \triangleright r} = c_t^w$. We say that such an interval is *of type write*. The set of t's satisfying this property is an interval of the form $[0, t_{(11)})$, where $t_{(11)} \geq 0$.

When a read occurs, we know that no current write is pending, and thus we can regard this read event as if it was the origin of times: $t = 0$.

Thus, we can repeat the above reasoning starting from any read event. Read events of interest for doing this are obviously those where one switches between the above two cases. Intervals of type read and of type write alternate, and the evaluation of $c_t^{w \triangleright r}$ is performed accordingly. This mechanism is illustrated on Figure 2. The switch from type write to type read is discovered at events of w. However it must be implemented at an event of r, whence the backtracking

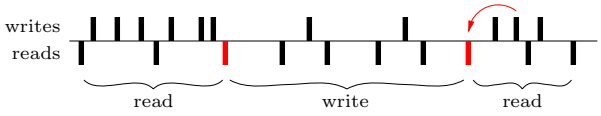

Figure 2: Illustrating the evaluation of $c_t^{w \triangleright r}$. The switching times between the two modes are shown in red.

shown by the backward pointing arrow in the figure. From this analysis, a number of consequences can be drawn:

PROPERTY 1 (COMMUNICATION BY SAMPLING).

1. *If $c_t^{w \triangleright r} = c_t^w$ holds, then $w \triangleright r$ stream preserves w, meaning that $\nu^{w \triangleright r} = \nu^w$.*
2. *If writes are more frequent than reads (formally, type read always holds), then, for every $t \in T^r$, $c_t^{w \triangleright r} = c_t^r$ holds, i.e., no read is superfluous.*
3. *If reads are more frequent than writes (formally, type write always holds), then, for every $t \in T^r$, $c_t^{w \triangleright r} = c_t^w$ holds. Thus, in this case, $w \triangleright r$ stream preserves w.*

Statement 1 expresses that the preservation of counters guarantees stream preservation. Statements 2 and 3 describe the behaviour of communication by sampling in case of slow/fast and fast/slow modes for the pair {writer, reader}. These properties are immediate. Additional properties are found in [16].

3.2 Taking latencies into account

So far we have ignored delay in the operator $\triangleright$: we have assumed that what is written is immediately available for reading. Clearly, this is rarely the case in practice, where various types of latencies are introduced between a writer and a reader, including program and operating system execution, communication, etc. To account for latencies, we consider the physically-delayed flow w', related to w by $\forall n : \nu_n^{w'} = \nu_n^w$ and

$$t_k^{w'} \quad = \quad t_k^w + \delta_k, \tag{15}$$

where δ_k is a (possibly variable) positive latency. Since latencies δ_k vary, it is possible that the order of events is reverted in w': for instance, if $\delta_k - \delta_{k+1} > t_{k+1}^w - t_k^w$, then we have $t_k^{w'} = t_k^w + \delta_k > t_{k+1}^w + \delta_{k+1} = t_{k+1}^{w'}$. We wish to forbid this, therefore we make the following assumption.

ASSUMPTION 1. *Latencies δ_k do not revert data:*

$$\forall k \in \mathbb{N} : t_{k+1}^{w'} \quad \geq \quad t_k^{w'} \tag{16}$$

Using (16), (15) can be rewritten as $\forall t \in \mathbb{R}_+ : c^{w'}_{t+\delta_{c_t^w}} = c_t^w$, also written as

$$c^{w'} \circ (Id + \delta_{c^w}) \quad = \quad c^w \tag{17}$$

We denote by δ the flow of latencies $\delta_k, k \in \mathbb{N}$, and by

$$\delta\,[w] \tag{18}$$

the flow w' related to w via (15) or (17). Whenever needed, all the results we provide in the sequel can be adapted to handle latencies, by replacing a considered flow e by its delayed version $\delta\,[e]$.

3.3 Buffered Communication by Sampling

Up to now, we have considered basic CbS, where the communication medium behaves like a shared memory. It is of interest to extend this mechanism with bounded buffers, as follows. We assume that the reader is equipped with a buffer of size M. The buffering mechanism, illustrated in Figure 3, is as follows:

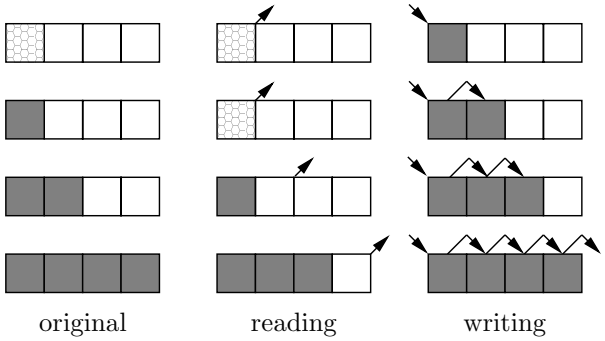

Figure 3: Buffered Communication by Sampling. Places in the buffer are indexed, from left to right, by $1, 2, 3, 4$. Four scenarios are shown in the figure (from top to bottom). Each scenario illustrates how the original buffer is transformed by a read or a write operation (the write operation is applied to the original buffer and not to the buffer resulting after the read). Grey boxes indicate places filled with data; white boxes are empty; and paved white boxes indicate that the data is removed except when this would make the buffer empty. Arrows indicate the move of data. When writing, a paved box is regarded as empty, as depicted in the first row.

- When the buffer is full, an additional writing puts the fresh data in place 1 of the buffer, and shifts by 1 place all data previously sitting in the buffer. This causes the loss of the oldest data, sitting in place M.
- Readings from the buffer get the oldest data from it, i.e., the data sitting at place $N_t^{w,r}$, where $N_t^{w,r}$ is the buffer level at instant t.
- When the buffer level is > 1, then readings consume the data. Alternatively, when the buffer level equals 1, then readings do not consume the data.
- The buffer is initially non-empty.

This mechanism is non blocking, for both writings and readings, and reduces to basic CbS when $M = 1$. Note the first row in Figure 3, which illustrates that the data is not consumed when buffer level is 1. This mechanism is denoted by

$$w \triangleright_M r \tag{19}$$

It is formalized next. For x and y two flows, set

$$\begin{array}{rcl} c^x_{s,t} & = & c^x_t - c^x_s \\ N^{x,y}_{s,t} & = & (c^x_{s,t} - c^y_{s,t}) - \min\limits_{u\in[s,t]}(c^x_{s,u} - c^y_{s,u}) \end{array} \tag{20}$$

$N_t^{w,r} =_{\text{def}} N_{0,t}^{w,r}$ yields the buffer level between writer w and reader r at instant t; (20) generalizes the two-counter mechanism (30) to the case where the initial instant is s instead of 0. This mechanism is analysed in Section 5.1.

Let w and r be the input and output flows of $w \triangleright_M r$, respectively. The clocks of w and r are the writer's and reader's clocks, respectively. In the following equations, vector flow $w[0, \ldots, M-1]$ is a buffer of size M, whose components are denoted by $w[k]$ for $k = 0, \ldots, M-1$. For every $m = 0, \ldots, M-1$ and $s \in \mathbb{R}_+$, set

$$\begin{array}{rcl} t^+[m,s] & = & \min\left\{t \in T^w \mid t > s \text{ and } m + N^{w,r}_{s,t} \geq M-1\right\} \\ t^-[m,s] & = & \min\left\{t \in T^r \mid t > s \text{ and } m + N^{w,r}_{s,t} \leq 0\right\} \\ t[m,s] & = & \min(t^+[m,s]\,,\, t^-[m,s]) \\ p[m,s] & = & \text{if } t[m,s] = t^-[m,s] \text{ then } 0 \text{ else } M-1 \end{array} \tag{21}$$

where, by convention, the minimum of the empty set is $+\infty$. Instant $t^+[m,s]$ is the first instant where the cumulated excess of writings over readings starting from level m at time s, reaches $M-1$. Symmetrically, $t^-[m,s]$ is the first instant where the cumulated excess of writings over readings starting from level m at time s, reaches 0. Finally, if the buffer level is m at time s, $t[m,s]$ is the first instant where the buffer reaches one of the two boundaries 0 or $M-1$, and $p[m,s]$ gives the buffer level reached at that time. Define inductively the following sequence of instants and buffer levels:

$$\begin{array}{rcl@{\quad,\quad}rcl} t_0 & = & 0 & m_0 & = & 0 \\ t_{n+1} & = & t[m_n, t_n] & m_{n+1} & = & p[m_n, t_n] \end{array} \tag{22}$$

If $t_n = +\infty$, then the subsequent terms of the sequence are also infinite. Then, for every $t \in \mathbb{R}_+$, the buffer level at time t is equal to:

$$t_n \leq t \leq t_{n+1} \qu\Rightarrow \quad N^{x,y}_t = m_n + N^{w,r}_{t_n,t} \tag{23}$$

And, finally, the buffer contents and output of the mechanism are given by:

$$\begin{array}{rcl} & & w[0]_t = w_t \\ \forall k = 1, \ldots, N^{w,r}_t & \Rightarrow & w[k]_t = w[k-1]^-_t \\ & & r_t = w[N^{w,r}_t]_t \end{array} \tag{24}$$

Equations (21)–(24) formalize buffered communication $w \triangleright_M r$. Observe that these equations are easily implemented in an on-line form. When

$$t_1 = +\infty \tag{25}$$

holds, i.e., the buffer never gets full, Equations (21)–(24) simplify as follows:

$$\begin{array}{rcl} & & N^{w,r}_t = N^{w,r}_{0,t} \\ & & w[0]_t = w_t \\ \forall k = 1, \ldots, N^{w,r}_t & \Rightarrow & w[k]_t = w[k-1]^-_t \\ & & r_t = w[N^{w,r}_t]_t \end{array} \tag{26}$$

Buffered CbS can be used to analyse the preservation of stream semantics by a LTT bus as follows. Consider a triple {writer w, bus b, reader r} communicating by sampling, in this order. The overall model of this communication medium is described by

$$(w \triangleright b) \triangleright r \tag{27}$$

Not surprisingly, due to (9), $(w \triangleright b) \triangleright r \neq w \triangleright (b \triangleright r)$ in general. Suppose that, in communication model (27), bus b implements M-buffered CbS communication, whence the overall

communication, from writer to bus to reader, is modelled as follows, see (19): $(w \triangleright_M b) \triangleright r$. Then, sufficient conditions for stream preservation are the following [16]:

PROPERTY 2 (LTT BUS WITH BUFFER). *Assume that the writing, bus and reading times satisfy the following equations, where δ^b, Δ^b, etc. are positive finite constants:*

$$\begin{array}{lrcl} \forall n: & \delta^w & \leq & t^w_{n+M} - t^w_n \\ \forall p: & \delta^b & \leq & t^b_{p+M} - t^b_p \leq \Delta^b \\ \forall m: & & & t^r_{m+M} - t^r_m \leq \Delta^r \end{array} \quad (28)$$

If

$$\delta^w \geq \Delta^b \text{ and } \left\lfloor \frac{\delta^w}{\Delta^b} \right\rfloor \geq \frac{\Delta^r}{\delta^b} \quad (29)$$

where $\lfloor x \rfloor$ denotes the largest integer $\leq x$, then $(w \triangleright_M b) \triangleright r$ stream preserves w.

4. THE LTT COMMUNICATION NETWORK (LTTN)

In this section, we consider mapping over a LTT Network, that is an ideal communication network in which every communication occurs according to the exact (unbuffered) CbS scheme modeled in Section 3. The case of mapping over a network of LTT busses is discussed in [16].

Formally, we consider a network involving computing units $C_i, i \in I$, where set I of sites is finite. Each site C_i is equipped with a quasi-periodic clock κ_i. Clocks κ_i are loosely, not strictly, synchronized (this is formalized later). The network is modeled as a directed graph $\mathcal{G}$ having $C_i, i \in I$ as set of vertices. Having a branch $C_i \rightarrow C_j$ in $\mathcal{G}$ means that C_i acts as a writer w_i and C_j acts as a reader r_j in a point to point CbS communication. Bi-directional and ring communications are allowed, thus $\mathcal{G}$ can have loops.

Communication $C_i \rightarrow C_j$ is by sampling, with the following characteristics (a) – (d), where $\kappa \downarrow 2$ denotes clock κ, down sampled by a factor of 2, i.e.,

$$\forall n \geq 0 : t_n^{\kappa\downarrow 2} = t_{2n}^{\kappa}$$

(a) Loose synchronization: for evey pair (i, j) of sites such that $C_i \rightarrow C_j$ is a branch of $\mathcal{G}$, pair $(\kappa_i \downarrow 2, \kappa_j)$ satisfies the assumption of Property 1.3, i.e., clock κ_j is more frequent than downsampled clock $\kappa_i \downarrow 2$.

(b) Writes w_i are tentatively triggered by clock $\kappa_i \downarrow 2$. Effective writes occur at a clock κ_i^w, downsampled from $\kappa_i \downarrow 2$ by traffic shaping [12], see below for a detailed description. A key feature of our traffic shaping policy is that it only depends on the effective writer's clocks for the different sites, not on the messages transferred.

(c) Reads r_j are triggered by clock κ_j. To cope with upsampling, from writer's clock κ_i^w to reader's clock κ_j, each message written by w_i comes equipped with an additional one-bit stamp that alternates between values 0 and 1. Alternations in the messages read by r_j, from 0 to 1 and from 1 to 0, indicate a fresh value. Other occurrences of w_i are repetitions. Together with the loose synchronization property (a), this one-bit stamp is in charge of maintaining data semantics, in the Kahn process network sense (see later).

(d) Only fresh data are used in performing writes w_j at rate $\kappa_j^w \subseteq \kappa_j \downarrow 2$. This operation is called "clock regeneration" and is studied next.

Communication by sampling, from writer w_i to reader r_j, is illustrated on Figure 4.

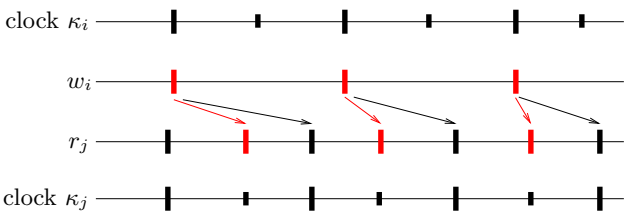

Figure 4: Illustrating communication in LTT network. Ticks of clock $\kappa \downarrow 2$ are indicated by large ticks of clock κ. Red bars indicate fresh data. Directed arrows indicate copying values.

We now have to link the LTTN to the underlying LTTA and to optimize the use of LTTA resources. In LTTA, being loosely synchronized, the physical clocks may suffer from relative drift and jitter. If not compensated for, clock drifts cause buffer overflows, thus resulting in loss of data and lack of semantical preservation. Thus, a set of protocols will be developed to ensure that the set of distributed logical clocks built on top of the loosely synchronized physical clocks exhibit no relative drift, but only bounded jitter.

5. PROTOCOLS TO ADAPT LTTA TO LTTN SEMANTICS

The two protocols introduced here (Clock Regeneration and Predictive Traffic Shaping) use a simple two-counter mechanism that we introduce first.

5.1 A generic two-counter monitoring protocol

All the protocols needed for LTTA will rely on a unique mechanism that we describe now. Consider two flows x and y, together with one of the following two hypotheses:

$\mathbf{H}_1$: there exists a real number $D > 0$ such that, for every $k > 0$, $t_k^y - t_{k-1}^y < D < t_k^x - t_{k-1}^x$ holds.

$\mathbf{H}_2$: there exists an integer $M > 0$ such that, for every $k > M$, $c^y \circ t_k^x - c^y \circ t_{k-M}^x \geq M$ holds.

Hypothesis $\mathbf{H}_1$ expresses that the (possibly time varying) period of flow y possesses an upper bound that is also a lower bound for the period of flow x. Hypothesis $\mathbf{H}_2$ expresses that, if flow x puts tokens in a buffer for (possibly immediate) consumption by flow y, then this buffer will never overflow provided that it has size at least M. Define the following quantity, for $t \in \mathbb{R}_+$:

$$\begin{array}{rcl} N_t^{x,y} & =_{\text{def}} & (c_t^x - c_t^y) - \min_{s \in [0,t]} (c_s^x - c_s^y) \\ & = & c^x_{s_t,t} - c^y_{s_t,t} \end{array} \quad (30)$$

where

$$c^x_{s,t} =_{\text{def}} c_t^x - c_s^x$$

and s_t is the last instant where the minimum over $s \in [0,t]$ is reached in (30) — s_t exists since counters are right continuous. Then, under $\mathbf{H}_1$ we have $\forall t \in \mathbb{R}_+,\ N_t^{x,y} \leq 1$ and, under $\mathbf{H}_2$ we have $\forall t \in \mathbb{R}_+,\ N_t^{x,y} \leq M$. So, monitoring the violation of either hypothesis is performed by maintaining counter $N^{x,y}$ and comparing it with the appropriate threshold. However, formula (30) defining $N^{x,y}$ is not suitable for on-line evaluation. We shall thus reformulate (30) in an on-line form. First, note that, although it is integer valued, $N^{x,y}$ is not a counter associated to some flow in the sense of Section 2.1. We must regard it as a flow itself. Seen as a flow, $N^{x,y}$ has set $T^{N^{x,y}}$ of dates of occurences, and sequence of values $\nu_n^{N^{x,y}}$, for $n = 1, 2, \ldots$, which we shall denote by $N_n^{x,y}$, with the abuse of notation proposed in the beginning of Section 2.1. This being said, note that

$$T^{N^{x,y}} \quad = \quad T^x \cup T^y. \tag{31}$$

Then, let z be the flow with values in the set $\{-1, 0, +1\}$, such that $T^z = T^x \cup T^y$, and whose value at a given occurence is $+1$ if x occurred alone, -1 if y occurred alone, and 0 if both x and y occurred simultaneously. Formally:

$$\begin{array}{rcl} \nu^z & \in & \{-1, 0, +1\} \\ T^z & = & T^x \cup T^y \\ \forall t \in T^z : \nu_t^z & = & \left\{ \begin{array}{rcl} +1 & \text{if} & x \text{ occurred alone at } t \\ -1 & \text{if} & y \text{ occurred alone at } t \\ 0 & \text{if} & \text{otherwise} \end{array} \right. \end{array} \tag{32}$$

Then, the following recursive formula for evaluating the successive values $N_1^{x,y}, N_2^{x,y}, \ldots$ of flow $N^{x,y}$ holds: $N_0^{x,y} = 0$, and, for every $n > 0$:

$$N_n^{x,y} \quad = \quad \max\big(N_{n-1}^{x,y} + z_n\,,\, 0\big) \tag{33}$$

This mechanism (31–33) for monitoring the violation of an hypothesis of the form $\mathbf{H}_1$ or $\mathbf{H}_2$ will be used in several contexts to develop our LTT architecture.

5.2 Clock Regeneration

The preservation of stream semantics is formalized by condition (8). To guarantee (8), statement 3 of Property 1 essentially requires that the writer shall be slower than the reader—what "slower" means is quantified precisely in the referred statement. Compensating for the ever decreasing sampling period in cascade communications is performed by a clock regeneration protocol, implemented on each computing unit C, see Figure 5. This protocol performs consistent downsampling of the input data read at rate κ, for re-emission at a rate (possibly downsampled from) $\kappa \downarrow 2$. The function performed by this protocol is to discard replicates, while properly emitting fresh data with no loss. This protocol is formalized next.

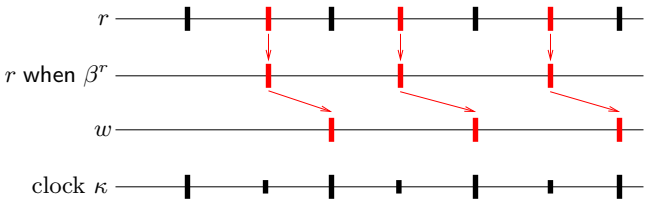

Figure 5: Illustrating clock regeneration Protocol 1. See Figure 4 for graphical conventions.

Protocol 1 (Clock Regeneration protocol). *C possesses a quasi-periodic clock κ and hosts a reader r and a writer w. Reader r and writer w are driven by κ and κ^w, respectively. Clock κ^w is some clock downsampled from $\kappa \downarrow 2$, i.e., $\kappa^w \subseteq \kappa \downarrow 2$, see (1), resulting from the traffic shaping policy described later. Read flow r has value of the following type*

$$\forall n \geq 0 : r_n \in D \times \{0, 1\}$$

meaning that, for every n, the nth value of flow r is marked by a bit, denoted by β_n^r.

Then, site C prepares flow w for its output by selecting the fresh reads and discarding repetitions, i.e., w is the output of 2-buffered CbS communication $\widehat{r} \triangleright_2 \kappa^w$, where

$$\widehat{r} \quad = \quad r \text{ when } [\beta^r \neq \beta^r \circ (Id - 1)]$$

Buffered CbS communication is defined in Section 3.3 and operator " when " is defined in Section 2.1. The relevant information transmitted by our LTT network is captured by the set of reads "r when β^r" and writes "w", attached to each site C. Protocol 1 is illustrated on Figure 6, obtained by "merging" Figures 4 and 5.

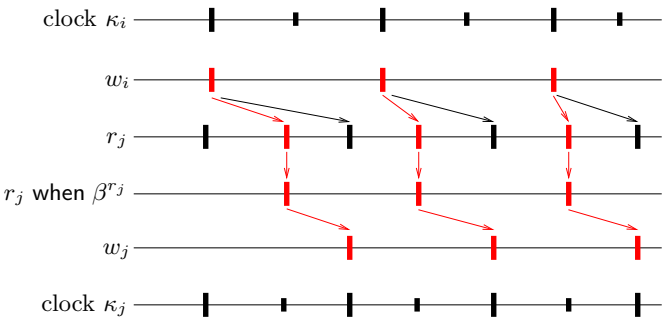

Figure 6: Combining $C_i \rightarrow C_j$ communication with Clock Regeneration Protocol at C_j.

So far Figure 6 shows the favorable case, where stream semantics is preserved without the need for a 2-buffer. However, Figure 7 shows that problems can occur if a 1-buffer

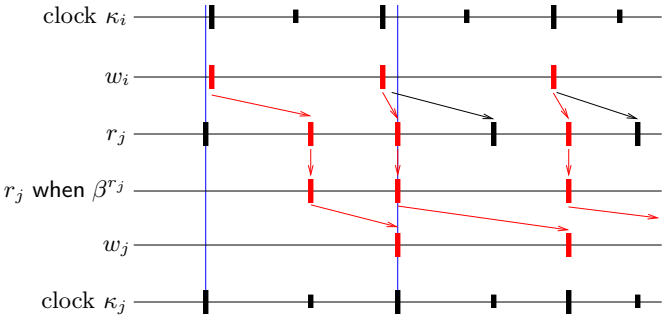

Figure 7: Combining $C_i \rightarrow C_j$ communication with Clock Regeneration Protocol at C_j. A situation where buffering is needed, compare with Figure 6. The problem is that two ticks of $\kappa_i \downarrow 2$ occur between the first two ticks of $\kappa_j \downarrow 2$.

only is used, despite our technique of downsampling, when clock κ_j is slower than clock κ_i. In this figure, the buffer gets filled at the second blue thin bar. It may never get empty, and may eventually overflow later, thus violating the preservation of stream semantics.

To overcome this problem, we use *predictive traffic shaping* at the writer. The idea is that the writer would implement some LTT based monitor detecting the risk of overwhelming its corresponding reader—see below. When approaching a risky situation, the writer would then anticipate and decide to skip a clock cycle and delay its transmission to the next cycle. This is formalized next.

5.3 Predictive traffic shaping

Predictive traffic shaping is implemented at site i in the following way. Assume for a while the following regarding network $\mathcal{G}$:

ASSUMPTION 2. *For each direct link $C_i \to C_j$ of $\mathcal{G}$, the reverse link $C_j \to C_i$, also exists in $\mathcal{G}$.*

Using this reverse communication link, site i can observe counters $c^{\kappa_i^w}$ and $c^{\kappa_j^w}$, so it can compare them by maintaining the counter

$$N_t^{ij} \quad =_{\text{def}} \quad N_t^{\kappa_i^w, \kappa_j^w} \tag{34}$$

If transmission exhibits zero latency, then, at any instant t, N_t^{ij} yields the level of the input buffer attached, at site j, to the communication link $C_i \to C_j$. Note that there is no circular definition in doing this, since traffic shaping will be performed in a predictive way, as we shall see. Maintaining counter N^{ij} on-line is performed by using the mechanism of Section 5.1.

For example, the situation depicted in Figure 7 occurs at the first instant t_* where $N_t^{ij} \geq 2$ holds. More precisely, the second blue thin bar would occur at that instant. Assume a 2-buffer is available at site j, as indicated in the figure. Then, the buffer gets full at instant t_* for the first time. If bounds exist for the relative drift and jitter between the two clocks at sites i and j, then the buffer will not overflow immediately, but some safe period exists, such that $N_t^{ij} \leq 2$ is guaranteed during this period. Now, two cases can occur:

- The buffer is emptied before the end of the safe period, thus bringing the communication link back to its safe mode, i.e., $N_t^{ij} \leq 1$.
- The end of the safe period is reached while the buffer is still full. Then, writer w_i decides to postpone sending its data to site j.

Since traffic shaping delays emissions, effective writes at site i are not according to nominal clock $\kappa_i \downarrow 2$, but are indeed possibly downsampled, with clock $\kappa_i^w \subseteq \kappa_i \downarrow 2$, see (1). This predictive traffic shaping policy is formalized next. Recall the notation $c_{s,t}^{\kappa} = c_t^{\kappa} - c_s^{\kappa}$, for $s < t$.

ASSUMPTION 3.

1. *There exists $K > 0$ such that, $\forall\ s, t$ with $s < t$*

$$\max\left(1\,,\,\min_i c_{s,t}^{\kappa_i}\right) \quad \geq \quad K(t-s) \tag{35}$$

Constant K is called a pessimistic rate *for network $\mathcal{G}$.*

2. *There exists a* safe period $\tau_{\text{safe}} \geq 0$ *such that, for any two s and t such that $0 \leq t - s \leq \tau_{\text{safe}}$,*

$$\max_{i,j}\left(c_{s,t}^{\kappa_i} - c_{s,t}^{\kappa_j}\right) \ \leq \ 2$$

Condition 1 expresses that the rates of all clocks are uniformly bounded from below—i.e., no clock can be "possibly infinitely slow".

PROTOCOL 2 (TRAFFIC SHAPING POLICY).

- *As part of running link $C_i \to C_j$, site i maintains counter N^{ij} defined in (34). Define the* safe mode of link $C_i \to C_j$ *by the condition $N_t^{ij} \leq 1$. Say that* site i is in safe mode *if all its outgoing links $C_i \to C_j$ for each $j \neq i$ are safe.*
- *When safe mode is left at site i (i.e., $N_t^{ij} = 2$ occurs for some $j \neq i$) a timer is started by site i, with timeout value equal to τ_{safe}. This timer is killed as soon as the site returns to its safe mode.*
- *Alternatively, if timeout occurs, then site i delays its next emission to all sites it communicates with, until return to the safe mode occurs.*

Note that chosing $\tau_{\text{safe}} = 0$ amounts to using no timer.

5.4 Semantics preserving properties

THEOREM 1. *Assumptions 2 and 3 are in force.*

1. *A sufficient condition ensuring stream preserving for each communication over network $\mathcal{G}$ is that each link $C_i \to C_j$ comes equipped with Protocols 1 and 2.*
2. *In this case, we have, for each site i, and every pair (s,t) such that $s < t$:*

$$\max\left(1\,,\,c_{s,t}^{\kappa_i^w}\right) \quad \geq \quad \tfrac{K}{2}(t-s)$$

Comments: The first statement says that our architecture is a Kahn process network [8]. The second statement says that, although some slow down results from applying traffic shaping, the overall network rate is at least $K/2$.

Proof of Theorem 1. See [16]. Applying repeatedly Theorem 1 shows that:

COROLLARY 1. *LTT networks, when combined with nodes implementing Protocols 1 and 2, preserve stream semantics.*

5.5 Relaxing Assumption 2 on network topology

So far we assumed that, for each direct link $C_i \to C_j$ of $\mathcal{G}$, the reverse link $C_j \to C_i$, also exists in $\mathcal{G}$. Assume now the following:

ASSUMPTION 4. *Network $\mathcal{G}$ is strongly connected.*

Site i in general has no direct knowledge of clock κ_j^w so it cannot compute counter N^{ij} defined in (34). Let $C_i \to C_j$ be a link of $\mathcal{G}$, and $C_j = C_{j_1} \to \ldots \to C_{j_M} = C_i$ be a path of shortest length, from site j back to site i. The idea is that site i will use $\kappa_{j_{M-1}}^w$ as an estimate of κ_j^w. We claim that

THEOREM 2. *The combination of Protocols 1 and 2 preserves stream semantics, provided that a M-buffer is used instead of a 2-buffer in Protocol 1.*

PROOF. See [16]. ◇

When $\mathcal{G}$ is not strongly connected, we apply Theorem 2 to each connected component. The different connected components are partially ordered and form a cascade communication, for which it is enough to have clocks of decreasing nominal periods, adjusted to ensure type write for communications between successive connected components.

6. CONCLUSION

We presented a comprehensive design flow from functional requirements to implementation on a distributed system consisting of multiple ECUs and interconnections that leverages the concept of Loosely Time Triggered Architecture. To do so, we introduced a formal model of computation to capture the communication mechanisms typical of LTTA and an intermediate layer of abstraction called Loosely Time Triggered Network. This layer of abstraction can then be mapped onto a variety of implementation architectures. The design process is guaranteed to preserve stream semantics if appropriate assumptions hold. The implicit assumption we have made in this paper is that the functional requirements are related to discrete control functions that include protection (fault tolerance, voting), mode changes, and some event triggered side functions.

Open problems are the efficient application of the LTTA architecture to implement continuous and hybrid functions such as low level safety critical feedback control loops, whose underlying design is performed based on continuous time models. Typical examples include flight control of aircrafts or combustion control for engines.

7. REFERENCES

[1] M. Baleani, A. Ferrari, L. Mangeruca, and A. L. Sangiovanni-Vincentelli. Efficient embedded software design with synchronous models. In *EMSOFT*, pages 187–190, 2005.

[2] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, A. L. Sangiovanni-Vincentelli, and S. Tripakis. Communication by sampling in time-sensitive distributed systems. In *EMSOFT*, pages 152–160, 2006.

[3] A. Benveniste, P. Caspi, P. L. Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis. A protocol for loosely time-triggered architectures. In *EMSOFT*, pages 252–265, 2002.

[4] P. Caspi and N. Halbwachs. A functional model for describing and reasoning about time behaviour of computing systems. *Acta Inf.*, 22(6):595–627, 1986.

[5] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. Lustre: a declarative language for programming synchronous systems. In *14th ACM Symp. POPL*, 1987.

[6] M. Di Natale, A. Benveniste, P. Caspi, C. Pinello, A. Sangiovanni-Vincentelli, and S. Tripakis. An implementation of the Loosely Time-Triggered Architecture on the Controller Area Network: feasibility conditions and system design. Technical report, 2007. submitted.

[7] A. Girault. A survey of automatic distribution method for synchronous programs. In F. Maraninchi, M. Pouzet, and V. Roy, editors, *International Workshop on Synchronous Languages, Applications and Programs, SLAP'05*, ENTCS, Edinburgh, UK, Apr. 2005. Elsevier Science.

[8] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74, Proceedings of IFIP Congress 74*, Stockholm, Sweden, 1974. North-Holland.

[9] H. Kopetz. *Real-Time Systems*. Kluwer Academic Publishers, 1997.

[10] C. Kossentini. *Implantation robuste de contrôles-commandes hybrides répartis: application aux commandes de vol Airbus*. PhD thesis, Institut National Polytechnique de Grenoble (INPG), 2006.

[11] C. Kossentini and P. Caspi. Approximation, sampling and voting in hybrid computing systems. In *HSCC*, pages 363–376, 2006.

[12] J.-Y. Le Boudec and P. Thiran. *Network Calculus*. Lecture Notes in Computer Science (LNCS). Springer Verlag, 2001.

[13] D. Potop-Butucaru, B. Caillaud, and A. Benveniste. Concurrency in synchronous systems. *Formal Methods in System Design*, 28(2):111–130, 2006.

[14] P. Potop-Butucaru and B. Caillaud. Correct-by-construction asynchronous implementation of modular synchronous specifications. *Fundamenta Informaticae*, 2006.

[15] J. Romberg and A. B. 0002. Loose synchronization of event-triggered networks for distribution of synchronous programs. In *EMSOFT*, pages 193–202, 2004.

[16] Same Authors. Full paper. IRISA Research Report Nr 1854. http://www.irisa.fr/distribcom/benveniste/pub/PI1854.pdf, June 2007.

[17] A. Sangiovanni-Vincentelli. Quo Vadis SLD? Reasoning about trends and challenges of System Level Design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[18] N. Scaife and P. Caspi. Integrating model-based design and preemptive scheduling in mixed time- and event-triggered systems. In *ECRTS*, pages 119–126, 2004.

[19] C. Sofronis, S. Tripakis, and P. Caspi. A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling. In *EMSOFT*, pages 21–33, 2006.

[20] S. Tripakis, C. Sofronis, N. Scaife, and P. Caspi. Semantics-preserving and memory-efficient implementation of inter-task communication on static-priority or edf schedulers. In *EMSOFT*, pages 353–360, 2005.