A Unifying View of Loosely Time-Triggered Architectures

Guillaume Baudart, Albert Benveniste, Anne Bouillard, Paul Caspi

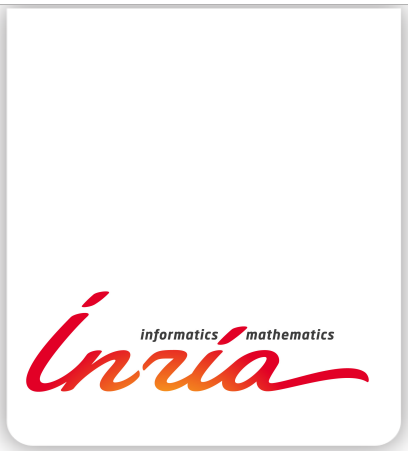
RESEARCH REPORT

N° 8494

March 2014

Project-Teams Hycomes, Parkas, and Trec

ISSN 0249-6399 ISRN INRIA/RR--8494--FR+ENG



A Unifying View of Loosely Time-Triggered Architectures

Guillaume Baudart*, Albert Benveniste†, Anne Bouillard‡, Paul Caspi§

Project-Teams Hycomes, Parkas, and Trec

Research Report n° 8494 — March 2014 — 14 pages

Abstract: Cyber-Physical Systems require distributed architectures to support safety critical real-time control. Hermann Kopetz' *Time-Triggered Architecture* (TTA) has been proposed as both an architecture and a comprehensive paradigm for systems architecture, for such systems. TTA offers the programmer a logical discrete time compliant with synchronous programming, together with timing bounds. A clock synchronization protocol is required, unless the local clocks used themselves provide the recquired accuracy. To relax the strict requirements on synchronization imposed by TTA, *Loosely Time-Triggered Architectures* (LTTA) have been proposed. In LTTA, computation and communication units are all triggered by autonomous, unsynchronized, clocks. Communication media act as shared memories between writers and readers and communication is non blocking. This is at the price of communication artifacts (such as duplication or loss of data), which must be compensated for by using some "LTTA protocol". In this paper we pursue our previous work by providing a unified presentation of the two variants of LTTA (token- and time-based), with simplified analyses. We compare these two variants regarding performance and robustness and we provide ways to combine them.

This report was prepared for a lecture in Gérard Berry's seminar series at the Collège de France, March 5, 2014; it is a corrected version of [6], which appeared at Emsoft'2010. It is dedicated to our close friend Paul Caspi who died in April 2012.

Key-words: Time-Triggered Architecture, Elastic Circuits, Loosely Time-Triggered Architecture

This research was supported in part by the European Commission under the Networks of Excellence IST-2001-34820 ARTIST and IST-004527 ARTIST2, and by IST STREP 215543 COMBEST.

* Ecole Normale Supérieure (ENS), Paris
† INRIA, Rennes, France. corresp. author: Albert.Benveniste@inria.fr
‡ Ecole Normale Supérieure (ENS), Paris
§ Formerly with Verimag, Grenoble

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Une présentation unifiée des Architectures Rythmées Souplement par le Temps

Résumé : Les infrastructures de calcul distribuées pour le contrôle des systèmes embarqués critiques requièrent des propriétés particulières destinées à préserver les caractéristiques attendues du contrôleur. Les architectures TTA (*Time-Triggered Architectures*) ont été proposées par Hermann Kopetz, à la fois comme une architecture de calcul et comme une méthodologie de conception des systèmes. TTA offre au programmeur un temps logique conforme à celui de la programmation synchrone, avec en outre un contrôle strict du temps. Il requiert un protocole de synchronisation entre les horloges du système réparti. Pour affaiblir ces hypothèses, les architectures LTTA (*Loosely Time-Triggered Architectures*) ont été proposées récemment. Dans LTTA, les calculs et les communications sont rythmées par des horloges locales, non synchronisées. Les supports de communication se comportent comme des mémoires partagées. La communication est donc non-bloquante. Ce type de communiccation crée évidemment des artefacts à combattre par un protocole dit "LTTA". Dans cet article nous présentons une approche unifiée des deux techniques connues pour ce type de protocole, reposant sur l'usage, soit de jetons, soit du temps. On compare ces deux variantes et on étudie leur performance.

Le présent rapport est une version corrigée de l'article [6], paru à Emsoft'2010. Il est dédié à notre très cher ami Paul Caspi, décédé en Avril 2012.

Mots-clés : Architectures TTA, Circuits élastiques, Architectures LTTA

CONTENTS

I Introduction 3

II LTTA and its artifacts 4

III Deployment and Semantics Preservation 5
- III-A The Synchronous Application 5
- III-B From Synchronous Application to 1-safe Petri net 5
- III-C LTTA Deployment and Semantics Preservation 6
- III-D Further Assumptions and Notations . . 6

IV Back-Pressure LTTA 7
- IV-A Preservation of synchronous semantics . 8
- IV-B Performance bounds 8
- IV-C Issues of blocking communication. . . . 8

V Time-Based LTTA 8
- V-A Preservation of synchronous semantics . 10
- V-B Performance bounds 11
- V-C Issues of blocking communication . . . 11

VI Hybrid LTT Architectures 11
- VI-A Discussion and comparison 12
- VI-B Blending the two architectures 12

VII Conclusion 12

References 12

Appendix: Model of a boolean register as a 1-safe Petri net 14

I. INTRODUCTION

Embedded electronics for safety critical systems have experienced a drastic move in the last decade, particularly in industrial sectors related to transportation (aeronautics and space, automobile, and trains, trams, or subways). In the past, each different function required its own set of sensors and actuators, its own controller, and a dedicated set of wires. This architecture, referred to as the *Federated Architecture*, has proved safe and robust by ensuring built-in partitioning between different functions. Federated Architectures could not be sustained in the late 90's, however, due to the drastic increase in number and complexity of functions and their interdependence. This lead a shift to *Integrated Architectures* [20] where several functions are hosted in a single computing unit and some functions are distributed across different computing units. Computing units as well as communication media can be standardised, thus allowing for drastic reductions in computing devices and wiring. While this move from Federated to Integrated Architectures opens new possibilities for further advances of embedded electronics in future embedded systems, it raises a number of challenging issues. The folding of different functions over shared computing units and the sharing of communication media can cause undesired interferences. Since system integration involves a mix of hardware, communication infrastructure, middleware, and software in a complex way, mismatches and failures to meet overall requirements emerges as a high risk at very late stages of system development. Since the overall system design relies on a layered view of the system, with several levels of abstraction corresponding to different computing or communication paradigms, it is not at all clear how detailed designs can indeed match system level specifications.

To address these problems as a whole, studies regarding *System Architecture* have been developed since the late 80's. Most remarkable is the *Time-Triggered Architecture* (TTA) developed by Hermann Kopetz and his school [17], [18]. TTA builds on a vision of the system in which physical time is seen as a first class citizen and as a help, not a hindrance. The Model of Computation and Communication (MoCC) of TTA is that of *strong synchrony*: the system is equipped with a discrete logical time, that is consistently maintained throughout the whole system. Strong synchrony is achieved by maintaining strictly synchronized physical clocks throughout the distributed architecture, up to a certain maximum accuracy — which in turn specifies the finest granularity of the discrete time in a TT Architecture. Having the precise MoCC of strong synchrony makes the deployment of an application easy, provided that the latter is also based on the same MoCC. Fortunately, Simulink/Stateflow and Scade, which are standard tools in these industrial sectors, are examples of formalisms obeying the synchronous MoCC. In addition, TTA offers more possibilities to address the above discussed difficulties. Firstly, time can be used as an aid in building fault tolerance services with redundancy management and fault detection and mitigation. Secondly, time also facilitates partitioning, and integrating components visible through their interfaces: Time Division Multiplexing (TDM) is a well established technique to grant a function access to communication or computing ressources. TDM is also at the very core of task scheduling.

However, the TTA approach carries cost and timing penalties that may not be acceptable for some applications. Indeed, jitter with smaller delays is preferred to fixed but longer delays for distributed control applications [1]. Also, TTA is not easily implementable for long wires (such as in systems where control intelligence is widely distributed) or for wireless communications. Finally and most importantly, re-designs are costly, due to the need for a global re-design of Time-Division multiplexing of the different functions or tasks. Hence, even for the safety critical hard real-time layers where TTA seems appropriate, it may not always be accepted.

Hence, there has been growing interest in less constrained architectures, such as the *Loosely Time-Triggered Architecture* (LTTA) [7]. LTTA is characterized by a communication mechanism, called *Communication by Sampling* (CbS), which assumes that: 1/ writes and reads are performed independently at all nodes connected to the medium, using different local clocks; and 2/ the communication medium behaves like a shared memory. See Figure 1 for an illustration.

LTT Architectures are widely used in embedded systems industries. The authors are personally aware of cases in aeronautics [23], nuclear, automation, and rail industries where the LTTA architecture with limited clock deviations has been used with success. It is indeed the architecture of choice for railway systems, in which tracks are used as the communication medium and computing systems are carried by the trains and work autonomously.

By not requiring any clock synchronization, LTTA is non blocking both for writes and reads. Hence, the risk of failure propagation throughout the distributed computing system is reduced and latency is also reduced, albeit at the price of increased jitter and drift [1]. However, data can be lost due to overwrites or duplicated because readers and writers are not synchronized [3], [24], [10]. Issues regarding the use of LTTA for distributed continuous control are discussed in [4]. If, as in safety critical applications that involve discrete control for operating modes or protection handling, data loss is not permitted, then special techniques must be developed to preserve the semantics of the specification.

The LTT bus based on CbS was first proposed in [7] and studied for a single writer-reader pair; [22] proposes a variation of LTTA where some master-slave re-synchronization of clocks is performed. The LTTA Architecture of general topology was studied in [3], [24], using techniques reminiscent of Back-Pressure [9], [8] and elastic circuits [11]. In a different direction, [19] developed an alternative approach where up-sampling is used in combination with "thick" events as a way to preserve semantics. This approach, which is more time-based as compared to [3], [24], was further developed and clarified in [10].

In this paper, we cast the two variants of LTTA in a unified framework. We simplify the analyses of [24] and we compare the respective merits of the two variants. Finally, we advocate blending them for different layers of the architecture and show how this can be done safely.

The paper is organized as follows. LTTA and Communication by Sampling are presented in Section II. This is followed in Section III by the definition of a synchronous application and the problem of the preservation of synchronous semantics at LTTA deployment. The next two sections are the core of this paper. The two variants of LTTA are developed: *Back-Pressure* based in Section IV and *time* based in Section V. The two architectures are compared in Section VI and we discuss why it would make sense to blend them, and how this can be done.

II. LTTA and its artifacts

LTTA relies on *Communication by Sampling*, which is illustrated in Figure 1 and formalized as the following set of assumptions:

Assumption 1:

1) *Each computing unit is triggered by its own local clock.*
2) *The communication medium behaves like a collection of shared memories, one for each variable; memory updates are assumed atomic;*

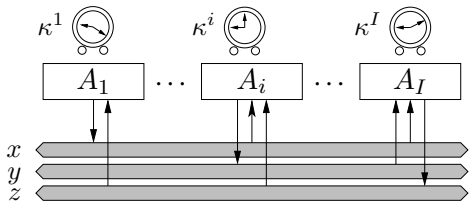


Figure 1. Communication by Sampling (CbS). For each variable x, y, or z, there is one shaded bus behaving as a shared memory, one writer and zero or more readers.

3) *Writes and reads are performed independently at all nodes connected to the medium, using different, non synchronized, local clocks.*

For some of the designs we propose, we will need to complement Assumption 1 with the following:

Assumption 2: Updates of every variable are visible to every node.

Due to the inherent robustness of physical systems against slight variations in sampling, direct use of CbS is suitable for continuous feedback control. Discrete systems such as finite state machines are generally not robust against duplication or loss of data. Combinational functions are generally not robust, and, even worse, sequential functions (with dynamically changing state) may react to such artifacts by diverging from their nominal behaviour. The issue is illustrated on Figure 2 for the case of combinational functions. We show here the

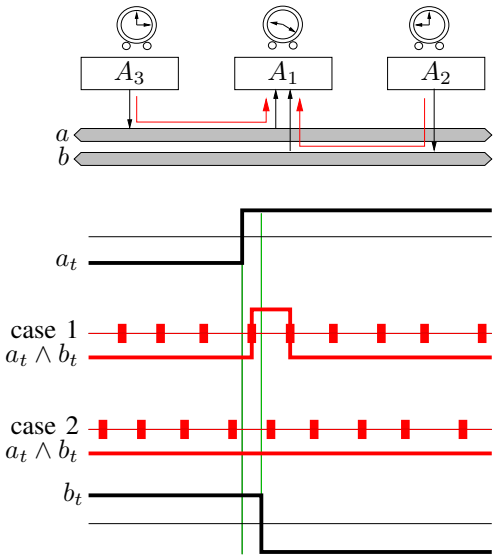


Figure 2. Sensing multiple signals, for distributed clocks subject to independent drifts and jitters. Referring to Figure 1, we show the case of A_1 reading two boolean inputs originating from A_2 and A_3, respectively, and computing their conjunction. Cases 1 and 2 correspond to two different outcomes for the local clock of A_1.

case of A_1 reading two boolean inputs a_t and b_t originating from A_2 and A_3, respectively, and computing their conjunction $a_t \wedge b_t$. Cases 1 and 2 correspond to two different outcomes

for the local clock of A_1. Observe that the result takes three successive values F, T, and F for case 1, whereas case 2 yields the constant value F. The origin of the problem is that events attached to different signals can be separated by arbitrary small time intervals — in Figure 2 the problem comes from the very close jumps of a_t and b_t. Increasing the clock rate cannot prevent this from occurring.

Due to such artifacts, the semantics of an application may not be preserved when deployed directly on an LTTA Architecture and protocols are needed to ensure correct functioning.

III. Deployment and Semantics Preservation

In this section we formalize the problem of preserving the semantics of a synchronous application when deploying it on an LTT Architecture.

A. The Synchronous Application

We are given an underlying set Z of *variables*. Our specifications for discrete controllers are modeled using dataflow diagrams. That is, they consist of a network of computing nodes of the following form:

$$\mathrm{N}: \left\{ \begin{array}{rcl} X_k & = & f(X_{k-1}, u_k^1, \ldots, u_k^p) \\ y_k & = & g(X_{k-1}, v_k^1, \ldots, v_k^q) \end{array} \right. \quad (1)$$

In (1), k is the discrete time index, $u^1, \ldots, u^p, v^1, \ldots, v^q \in Z$ are the *input variables* of the node, $X \subseteq Z$ is the tuple of *state variables* of the node, $y \in Z$ is the output variable of the node, and f, g are functions. This model can capture multiple-clocked systems simply by extending the domains of variables with a special value denoting absence. Absence can be reasoned about using type systems such as clock calculi of Lustre or Signal. Nodes $\mathrm{N}_1, \ldots, \mathrm{N}_n$ can be composed by output-to-input connections to form *systems*, i.e., networks of nodes, denoted by $S = \mathrm{N}_1 \parallel \ldots \parallel \mathrm{N}_n$. Systems can be further composed in the same way, denoted by

$$S_1 \parallel \ldots \parallel S_I. \quad (2)$$

Node (1) is abstracted as the following labeled directed graph:

$$\mathcal{G}(\mathrm{N}): \left\{ \begin{array}{ccccccc} X \overset{\mathrm{UD}}{\leftarrow} X & , & X \leftarrow u^1 & , & \ldots & , & X \leftarrow u^p \\ y \overset{\mathrm{UD}}{\leftarrow} X & , & y \leftarrow v^1 & , & \ldots & , & y \leftarrow v^q \end{array} \right. \quad (3)$$

A branch $y \overset{\mathrm{UD}}{\leftarrow} X$ indicates that y depends on X through a Unit Delay, whereas a branch $y \leftarrow v$ indicates a direct dependency. Systems $S = \mathrm{N}_1 \parallel \ldots \parallel \mathrm{N}_n$ are abstracted as the union of the associated graphs $\mathcal{G}(S) = \mathcal{G}(\mathrm{N}_1) \cup \cdots \cup \mathcal{G}(\mathrm{N}_n)$, and the same holds, inductively, for $\mathcal{G}(S)$ when $S = S_1 \parallel \ldots \parallel S_I$. Combinational loops are prohibited:

Assumption 3: We require that no loop exists in $\mathcal{G}(S)$ involving branches not labeled by delay symbols UD.

Referring to a composition $S = S_1 \parallel \ldots \parallel S_I$, consider $\mathcal{G}(S) = \mathcal{G}(S_1) \cup \cdots \cup \mathcal{G}(S_I)$. Using Assumption 3, erasing, in $\mathcal{G}(S)$, branches labeled by delay symbols UD yields a partial order denoted by $\preceq$. Every total order that is an extension of $\preceq$ yields a correct scheduling of the atomic computation steps needed to complete a reaction of S.

To simplify our study, we consider the following additional assumption (this assumption will be relaxed in a later paper):

Assumption 4: Every communication between different sites is subject to a unit delay or more.

B. From Synchronous Application to 1-safe Petri net

In this section we recall a result belonging to the "folklore" of synchronous programming, see [5], namely: synchronous programs can be mapped to 1-safe Petri nets while preserving flow semantics. Recall that a *Petri net* [21] is a tuple $\mathcal{N} = (P, T, \rightarrow, M_0)$, where P is a set of *places*, T is a set of *transitions* such that $P \cap T = \emptyset$, $\rightarrow \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. Write $n \rightarrow m$ to mean $(n, m) \in \rightarrow$. For $n \in P \cup T$, ${}^\bullet n = \{m \mid m \rightarrow n\}$ and $n^\bullet = \{m \mid n \rightarrow m\}$ denote the *preset* and *postset* of node n. A *marking* is a map $M : P \mapsto \mathbb{N}$ and, in the tuple defining $\mathcal{N}$, M_0 is the *initial marking*. *Firing* transition $t \in T$ from marking M requires $M(p) > 0$ for every $p \in {}^\bullet t$ and yields the new marking M' such that $M'(p) = M(p) - 1$ for $p \in {}^\bullet t \setminus t^\bullet$, $M'(p) = M(p) + 1$ for $p \in t^\bullet \setminus {}^\bullet t$, and $M'(p) = M(p)$ otherwise. A marking M is *reachable* if there exists a finite sequence of firings, starting at M_0 and ending at M. A Petri net $\mathcal{N}$ is 1-*safe* if $M(p) \leq 1$ for all p at every reachable marking including M_0. Say that $\mathcal{N}$ is an *event graph* if ${}^\bullet p$ and $p^\bullet$ are singletons for every place p. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two nets such that $P_1 \cap P_2 = \emptyset$. Define their *product* $\mathcal{N}_1 \times \mathcal{N}_2$ to be the net $\mathcal{N}$ such that $P = P_1 \cup P_2$, $T = T_1 \cup T_2$, $\rightarrow = \rightarrow_1 \cup \rightarrow_2$, and $M_0 = M_{1,0} \cup M_{2,0}$, that is, $\mathcal{N}_1 \times \mathcal{N}_2$ is obtained by superimposing shared transitions of the two nets—places are private. Finally, one can associate a holding time to a place which is the time a token must spend in the place before contributing to the enabling of transitions.

Using instantaneous broadcast: We first consider a mapping in which parallel composition of synchronous systems is realized through instantaneous broadcast. No distinction is made between the reading and the writing of a variable. Equivalently, output-to-input connections are considered instantaneous. Referring to the synchronous application introduced in Section III-A, for x and y two variables, the two 1-safe event graphs $\mathcal{N}_{xy}^{\mathrm{UD}}$ and $\mathcal{N}_{xy}$ shown in Figure 3 correspond, respectively, to the dependencies $y \overset{\mathrm{UD}}{\leftarrow} x$ and $y \leftarrow x$.

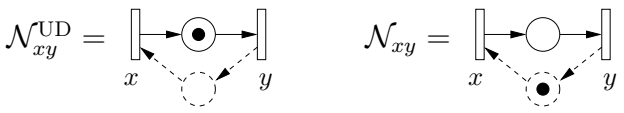


Figure 3. The two nets $\mathcal{N}_{xy}^{\mathrm{UD}}$ and $\mathcal{N}_{xy}$. Rectangles figure transitions, circles figure places, and initial tokens are depicted by black bullets.

Consider a computing node N of the form (1) and its associated graph $\mathcal{G}(\mathrm{N})$ as in (3). To N we associate the 1-safe event graph $\mathcal{N}(\mathrm{N})$ equal to

$$\mathcal{N}_{XX}^{\mathrm{UD}} \times \mathcal{N}_{Xy}^{\mathrm{UD}} \times \left(\prod_{k=1}^{p} \mathcal{N}_{u^k X} \right) \times \left(\prod_{\ell=1}^{q} \mathcal{N}_{v^\ell y} \right) \quad (4)$$

Shortest firing sequences of $\mathcal{N}(\mathrm{N})$ starting from and returning to the initial marking—we call them *big-steps*—are in one-to-one correspondence with the linear extensions of partial order $\preceq$ introduced in Section III-A. Thus, big-steps capture sequential executions of the synchronous system N.

For $S{=}\mathrm{N}_1 \parallel \ldots \parallel \mathrm{N}_I$, define $\mathcal{N}(S){=}\mathcal{N}(\mathrm{N}_1){\times}\ldots{\times}\mathcal{N}(\mathrm{N}_I)$ by using (4). And, then, for $S = S_1 \parallel \ldots \parallel S_I$ a system as in Section III-A, we define, inductively, $\mathcal{N}(S) = \mathcal{N}(S_1){\times}\ldots{\times}\mathcal{N}(S_I)$. Again, big-steps of $\mathcal{N}(S)$ capture sequential executions of synchronous system S. Big-steps start from the beginning of a reaction and terminate at its end. Considering all firing sequencess of $\mathcal{N}(S)$ yields the *Kahn Process Network* [16] executions of S. For σ a firing sequence of $\mathcal{N}(S)$, let $\overline{\sigma}$ be any *synchronous completion* of it, i.e., any shortest firing sequence extending σ and returning to the initial marking. Such $\overline{\sigma}$ is thus the concatenation of big-steps, i.e., represents a finite sequence of successive reactions of synchronous system S. Hence, executing $\mathcal{N}(S)$ as a net preserves the individual flows of S, i.e., the sequences of values taken by an arbitrary variable x of S.

Using explicit links: In this setting, we consider explicit models for the output-to-input links of the synchronous application. That is, node N introduced in (1) is rewritten

$$\mathrm{N}: \left\{ \begin{array}{rcl} X_k & = & f(X_{k-1}, (r_u)_k^1, \ldots, (r_u)_k^p) \\ (w_y)_k & = & g(X_{k-1}, (r_v)_k^1, \ldots, (r_v)_k^q) \end{array} \right. \tag{5}$$

by distinguishing the reading of an input (here r_u, r_v) from its writing (here w_y). This allows modeling the link $w_y \mapsto r_y$ explicitly, with or without a delay in it. See Figure 4, where the two nets $\mathcal{N}^{\mathrm{UD}}_{w_y r_y}$ and $\mathcal{N}_{w_y r_y}$ correspond, respectively, to the links:

$$\begin{array}{lll} (r_y)_k = (w_y)_{k-1} & \text{giving the dependency} & r_y \overset{\mathrm{UD}}{\leftarrow} w_y \\ (r_y)_k = (w_y)_k & \text{giving the dependency} & r_y \leftarrow w_y \end{array}$$

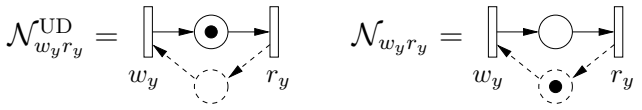


Figure 4. The two nets $\mathcal{N}^{\mathrm{UD}}_{w_y r_y}$ and $\mathcal{N}_{w_y r_y}$.

Now, let us investigate the consequences of Assumption 4 on net $\mathcal{N}(S)$. Consider first the case in which $S = \mathrm{N}_1 \parallel \ldots \parallel \mathrm{N}_I$. To capture the fact that any variable x is transmitted, from some node j to some node i, with a positive delay, we create a net $\mathcal{N}^{\mathrm{UD}}_{w_x r_x}$ between node j and node i. Thus, all nets modeling links are of the "$\mathcal{N}^{\mathrm{UD}}$" form. Other nets may be of one of the two forms of Figure 4. For the sake of simplicity, in the following we will write $\mathcal{N}_{ji}$ rather than $\mathcal{N}^{\mathrm{UD}}_{ji}$ for a link from node j to node i.

C. LTTA Deployment and Semantics Preservation

Deploying the application $S_1 \parallel \ldots \parallel S_I$ of formula (2) over a strictly synchronous architecture is straightforward. In such an architecture, computing units and communication media are all triggered by a unique periodic global clock. The different computing units compute in lock steps — we call them *reactions* — according to the global clock. Each node is assigned some computing unit for its execution. Then, the computation of the different variables is scheduled within each reaction, by respecting the partial order $\preceq$. In this case, variables are updated at each reaction. We can instead cluster a (possibly variable) number of successive clock ticks together to form *macro-reactions* and update the variables once in each macro-reaction. Now, inside each macro-reaction, one needs to schedule the computations of the different variables by respecting the partial order $\preceq$. This leaves room for computing different variables at different clock ticks. All the above designs correctly implement the application function, which consists in mapping input streams to output streams. We say that *application semantics is preserved.* The question is: if, instead, deployment is performed over the LTT Architecture of Figure 1, how can application semantics be preserved?

As discussed in Section II, CbS communication by itself is not sufficient. In the next two sections we propose two different protocols on top of the CbS infrastructure to ensure semantics preservation. We now complete this section with further assumptions and notations.

D. Further Assumptions and Notations

The following assumptions will also be considered, see Figure 1:

Assumption 5: Local clocks $(\kappa^i)_{i\in I}$ possess lower and upper bounds $T_{\min}$ and $T_{\max}$ for the time interval between any two successive ticks κ^i_{k-1} and κ^i_k: $\forall k \in \mathbb{N}$,

$$T_{\min} < \kappa^i_k - \kappa^i_{k-1} < T_{\max}\,, \tag{6}$$

and communication delays are bounded by $\tau_{\min}$ and $\tau_{\max}$,

$$0 \leq \tau_{\min} < \tau < \tau_{\max}. \tag{7}$$

Assumption 6: For each computing unit, executions take at most one clock cycle and a computing unit which starts executing freezes its input data.

We stress that communicated variables are not updated at each reaction of the application, but only when required by the application. Consequently, a processor does not know a priori which variable update it is supposed to see at a given reaction.

Regarding notations, x, y, z, X, etc. denote variables. Nodes are indexed by $1, \ldots, n$. We can always group the output variables of a node into one tuple, which we regard again as a single variable. Thus, *without loss of generality, we can assume that each node writes into a single variable labeled with the index of the node.*

In the following sections we present two protocols that were proposed on top of CbS communication in order to ensure that the deployment of synchronous applications over resulting LTT Architectures is semantics preserving. The first protocol is an adaptation of elastic circuits in hardware, and the second one is a relaxed, time based, adaptation of the original TTA. For each of them we indicate the required assumptions.

IV. Back-Pressure LTTA

Assumptions: Throughout this section, Assumptions 1 and 6 must hold regarding the architecture. The application for deployment satisfies Assumptions 3 and 4. Assumptions 2 and 5 are not required in this section.

Elastic circuits were proposed in [12], [15], [11] as a semantic preserving architecture in which Kahn Process Network [16] type of execution is performed using bounded buffers. This is achieved by relying on a mechanism of *Back-Pressure* [8] by which reads from a buffer by a node is acknowledged to the writer using a reversed virtual buffer. Petri net $\mathcal{N}_{ji}$ of Figure 5 depicts how a link $j \to i$ with a 1-buffer is implemented in an elastic circuit for running a synchronous application with a 1-delay communication. *Back-Pressure* places and arcs of this net are dashed, to distinguish them from the corresponding *direct* places and arcs, which are solid — solid and dashed places and arcs both obey the usual net semantics. Only direct places model data communication, Back-Pressure ones are there to prevent from buffer overflow at the link from j to i.

In our study, however, we cannot make direct use of elastic circuits since the activation of nodes in elastic circuits is triggered by tokens, not by autonomous non-synchronized clocks as in LTTA. To adapt to the constraints of LTTA, the authors of [24] have proposed to enhance elastic circuits with a *skipping mechanism* that we present now under the name of *Back-Pressure* LTT Architecture. The model of this architecture is developed using Petri nets, which we assume 1-safe throughout this section. Some places in the nets will be filled with a color, to indicate that a holding time will be subsequently associated with it. Also, some slight deviation from Petri net semantics will be needed to capture priority arising in certain conflicts.

Principle 1: To summarize the essence of this protocol, synchronization is ensured by using tokens for slowing down faster nodes.

Modeling the links: Figure 5 depicts net $\mathcal{N}_{ji}$ associated to each directed link $j \to i$ of the architecture. This net assumes a 1-buffer on each link—see the discussion at the end of Section III-B for a justification of this.

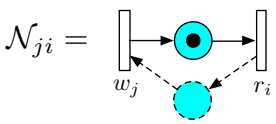


Figure 5. Back-Pressure net $\mathcal{N}_{ji}$ associated to each directed link $j \to i$ of the architecture. For performance studies of Section IV-B, the holding time of the blue places models transmission delay.

Modeling the nodes: Reactions at node i are captured by the net $\mathcal{N}_i$ shown in Figure 6, which is composed of a two-step "read; write" together with a local skipping mechanism. Inter-tick time is captured by a holding time (Assumption 5) associated to each yellow place. The following holds regarding

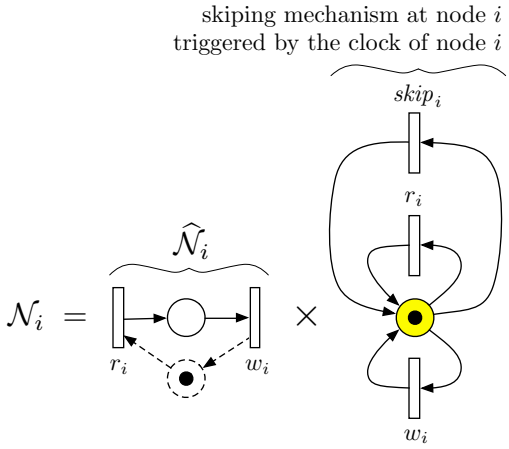


Figure 6. Net $\mathcal{N}_i$ showing the mechanism of skipping at node i. For performance studies of Section IV-B, inter-tick time is captured by the holding time of yellow places.

this skipping mechanism, which was proposed in [24] in order to avoid computing units getting blocked — blocking is replaced by skipping:

The skipping mechanism is triggered by the local clock at node i. (8)

Transitions with labels r_i and w_i have priority over transition with label $skip_i$. (9)

In order for Figure 6 to capture feature (8), we take the convention that an enabled transition is fired immediately. Indeed, the transition $skip_i$ is enabled as soon as the token is available in the yellow place. Thus, if no other transition is available, the node must fire the $skip_i$ transition when the holding time elapsed. Since the holding time of yellow places models inter-tick time, it corresponds to the ticks of κ^i.

The composition indicated in Figure 6 by the symbol $\times$ is by superimposing transitions having same label, thus forcing the synchronisation of the corresponding transitions in the composed nets. In particular, when clock κ^i of node i has a tick, then action r_i or w_i is fired if enabled, and otherwise $skip_i$ is fired, expressing that node i keeps silent at that tick.

The network: Referring to Figures 5 and 6, consider the following two product nets

$$\widehat{\mathcal{N}} = \left(\prod_i \widehat{\mathcal{N}}_i\right) \times \left(\prod_{j\to i} \mathcal{N}_{ji}\right) \quad (10)$$

$$\mathcal{N} = \left(\prod_i \mathcal{N}_i\right) \times \left(\prod_{j\to i} \mathcal{N}_{ji}\right) \quad (11)$$

where the product is obtained by superimposing transitions having identical labels.

Net $\widehat{\mathcal{N}}$ (without the skipping mechanisms) yields the *elastic circuit* implementing the original synchronous application according to the Kahn Process Network semantics — which is known to preserve synchronous semantics. Observe that net $\widehat{\mathcal{N}}$ exhibits no conflict and is thus an *event graph* (also sometimes called *marked graph*). With our assumption of single-delay communications, net $\widehat{\mathcal{N}}$ is indeed 1-safe.

On the other hand, net $\mathcal{N}$ is the *Back-Pressure net* modeling our Back-Pressure LTTA, which involves the skipping mechanism. In the remainder of this section, we first analyse the preservation of synchronous semantics by $\mathcal{N}$ and then we study its performance.

A. Preservation of synchronous semantics

This result was first proved in [24], for more general architectures. We give here a very simple and direct proof, by explicitly using the associated elastic circuit $\widehat{\mathcal{N}}$. The following theorem makes no use of any argument related to timing characteristics, with the exception of the following weak fairness condition:

> No transition can be enabled forever without firing. (12)

Theorem 1: Net $\mathcal{N}$ preserves synchronous semantics.

Proof: Net $\widehat{\mathcal{N}}$ is indeed an elastic circuit which is known to implement a synchronous program with Kahn Network Process semantics; 1-buffers can be used on the links since direct links all have a logical 1-delay by Assumption 4. Observe that this first property only relies on assumptions 3–6; it does not use Assumption 1 nor conditions (6,7) regarding clocks and communication delays.

Let $\mathcal{L}_{\mathcal{N}}$ be the language of net $\mathcal{N}$, i.e., the set of all its firing sequences, and similarly for $\mathcal{L}_{\widehat{\mathcal{N}}}$. The following fairness condition follows from (12):

> It is not possible that transition $skip_i$ of the skipping mechanism fires repeatedly for ever. (13)

Using (13), the projection of the language $\mathcal{L}_{\mathcal{N}}$ over alphabet $\{r_i, w_i \mid i = 1, \ldots, n\}$ coincides with the language $\mathcal{L}_{\widehat{\mathcal{N}}}$, which proves the preservation of synchronous semantics. ◇

Observe that fairness condition (13) is indeed much weaker than the conjunction of (8) and Assumption 5.

B. Performance bounds

Assumptions: Assumption 5 is in force for the derivation of performance bounds.

Let us first focus on elastic circuit $\widehat{\mathcal{N}}$ defined in (10). Assume the following conditions for this elastic circuit — they are similar to (6) and (7):

(6') there exist lower and upper bounds $\overline{T}_{\min}$ and $\overline{T}_{\max}$ for the interval between any two successive firings of a transition related to node i (it can be a read r_i or a write w_i); this is captured by assigning these bounds to the holding time of the yellow places in Figure 5.

(7') the holding time of direct or Back-Pressure places of $\mathcal{N}_{ji}$ is bounded by $\overline{\tau}_{\min}$ and $\overline{\tau}_{\max}$, for any link $j \to i$ (direct places are the solid ones in Figure 5, whereas Back-Pressure places are dashed).

Following classical results on event graphs [14] (Chapter 6.7, p.247), [2] (Chapter 2.5) or (max, plus) algebras [13] (Chapters 21-26), the worst case throughput $\lambda_{\widehat{\mathcal{N}}}$ of net $\widehat{\mathcal{N}}$ is given by the minimal ratio: number of tokens/time over all cycles of the event graph, that is:

$$1/\lambda_{\widehat{\mathcal{N}}} \quad = \quad 2\max\left(\overline{T}_{\max}, \overline{\tau}_{\max}\right) \tag{14}$$

Next, net $\mathcal{N}$ consists in adding, at each node i of net $\widehat{\mathcal{N}}$, the skipping mechanism shown on Figure 5. Now, assume that conditions (6) and (7) hold for net $\mathcal{N}$, namely:

(6) there exist lower and upper bounds $T_{\min}$ and $T_{\max}$ for the interval between two successive firings of the skipping mechanism at any node;

(7) the holding time associated to any place of $\mathcal{N}_{ji}$ is bounded by $\tau_{\min}$ and $\tau_{\max}$ for any link $j \to i$.

We claim that, when synchronizing with all the local skipping mechanisms, net $\widehat{\mathcal{N}}$ inherits the following values for its bounds $\overline{T}_{\max}$ and $\overline{\tau}_{\max}$ mentioned in (6') and (7'):

$$\overline{T}_{\max} \quad = \quad T_{\max} \tag{15}$$

$$\overline{\tau}_{\max} \quad = \quad T_{\max} + \tau_{\max} \tag{16}$$

Indeed, bound (15) is realized by the node i with the slowest clock, since this node does not need to skip. Bound (16) is reached when the latest token reaches an input place of node i but net $\mathcal{N}_i$ fired just before. Combining (6'), (7'), (14), (15) and (16) yields:

Theorem 2: The worst case throughput $\lambda_{\mathcal{N}}$ of net $\mathcal{N}$ is

$$1/\lambda_{\mathcal{N}} = 2\left(T_{\max} + \tau_{\max}\right)$$

Recall that Theorem 1 only requires fairness condition (13) and Theorem 2 only requires the upper bounds in (6,7), but not the lower bounds. Performance results are provided in [24] for more general architectures (with arbitrary buffer sizes). However, the proof we give here is much more straightforward.

C. Issues of blocking communication.

The skipping mechanism ensures that computing nodes themselves never get blocked due to the failure of other nodes or communication link. However, net $\widehat{\mathcal{N}}$ involves blocking communication between the different computing nodes of the architecture. This means that, *when focusing on the effective communication of fresh data at a given node, blocking does still occur in net $\mathcal{N}$*. This observation actually motivated considering the alternative, time-based, LTT Architecture that we describe and analyse in the next section.

V. Time-Based LTTA

Assumptions: Assumptions 1–6 hold throughout this section, for both the preservation of semantics and the calculation of performance bounds.

By contrast with Principle 1 used in Back-Pressure LTTA, the following principle is followed in Time-Based LTTA:

Principle 2: Synchronization is ensured by using time for slowing down faster nodes and tokens for speeding up slower ones.

Time-Based LTTA relies on an original idea of P. Caspi [19], [10]. The aim of this protocol is to ensure a clean alternation of writing and reading phases throughout the architecture, see Figure 7. In this figure, we show the first reading phase $\mathbf{R}_0$ and writing phase $\mathbf{W}_0$; since $\mathbf{R}_0$ consists in reading initial values sitting in local memories, it takes no time. We also show later alternating phases. The pairs of phases $\mathbf{W}_{k-1}, \mathbf{R}_k$ are used in the proof of forthcoming Theorem 3 for selecting the two integers p and q arising in the time-based LTTA protocol shown on Figures 11 and 10.

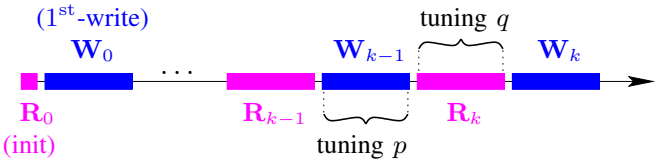


Figure 7. Alternating phases of reads and writes. The right arrow figures the time line and k is the reaction index.

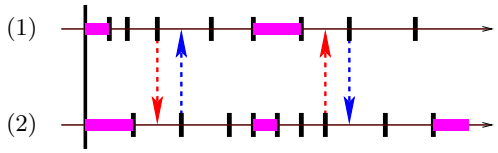


Figure 8. Synchronization principle used in time-based LTTA.

The synchronization principle used in time-based LTTA is illustrated on Figure 8, which shows two time lines, for two communicating nodes (1) and (2). Ticks of the local clocks are depicted by the short thick vertical bars (note the jitter, clocks are not periodic). Magenta rectangles depict reading-and-computing periods; they correspond to r_{k-1}, r_k, r_{k+1} in Figure 7. At the beginning, node (1) is the fastest. Thus, it waits for a certain amount of ticks and then it writes its computed value; this is indicated by the red dashed arrow pointing to (2). Upon noticing this publication (after transmission through the network), node (2) responds with its own publication, indicated by the blue dashed arrow pointing to (1). Meanwhile, node (1) remains suspended to ensure that node (2) has enough time to publish its own fresh value, if any (fresh values need not be produced at every reaction of the synchronous application). Then it can repeat reading-and-computing. In the second round, node (2) is the fastest and publishes first. And so on.

Observe that the two publications in blue are not based on time. They rather react to a publication by another node. The key observation is that fast nodes slow down by waiting a number q of ticks of their local clocks before writing their value, whereas slow nodes accelerate by actively synchronizing on fast nodes' publications. Then, every node waits for p ticks of their local clocks, to ensure that all publications have been made by other nodes before reading a new value. The key issue is to tune p and q to the smallest values that correctly ensure the alternation shown in Figure 7. The architecture implementing this protocol is detailed next using again the same Petri net framework.

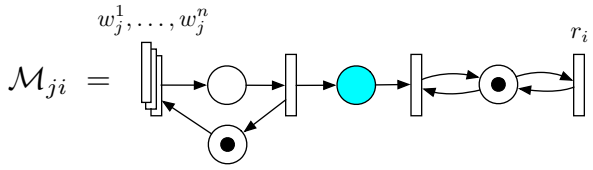


Figure 9. Net $\mathcal{M}_{ji}$ modeling directed CbS link from node j to node i. Note the total lack of synchronization. Compare with net $\mathcal{N}_{ji}$ of Figure 5.

Modeling the links: Figure 9 shows net $\mathcal{M}_{ji}$, which models CbS communication for directed link from node j to node i—note the splitting of transition w_j into mutually exclusive transitions $w_j^1 \ldots w_j^{q_j}$, motivated by the forthcoming consideration of product (17). In this net, reads and writes can occur concurrently and asynchronously. The left-most places act as an edge detector to detect the publication of a new value. Whenever one of the w_j^q transition is fired, the token moves to the top place and then instantaneously returns to the bottom place. This alternation puts a token in the central blue place which models network communication. A holding time τ is associated to this place to model transmission delay (see Figure 5).

Note that, a priori, an unbounded number of tokens may appear in the central place. It is the role of the node controllers, which control the firing of transitions w_j^q and r_i, to avoid this. Furthermore, the communication medium does not guarantee that data are delivered in due order. Indeed, since the transmission delay can vary between two publications, it is possible that a token published after another is delivered beforehand. Again, the role of the node controllers is to prevent this from happening.

Modeling the nodes: In time-based LTTA synchronization is local to each node and is a mix of time- and token-based synchronization. It involves the nets shown in Figures 10 and 11, which are explained below.

The protocol instance running at node i is modeled by the net

$$\mathcal{M}_i \quad = \quad \mathcal{P}_i \times \mathcal{M}_i^r \times \mathcal{M}_i^w \times \left(\prod_{j \neq i} \mathcal{P}_j\right) \qquad (17)$$

where the product is by superimposing transitions with identical label. Referring to the right hand side of (17), the role of the different components of $\mathcal{M}_i$ is as follows:

Net $\mathcal{P}_i$ records the detection of publication by another node since the last local read (firing of transition r_i). The red color indicates that places hold a boolean status, not a token. The blue color indicates that a latency will be subsequently associated to the place π_i to model a publication delay between the setting and reading times. Note that these "boolean registers" can be cast into the same Petri net framework and can be seen as macros, used here for readability—see Appendix A.

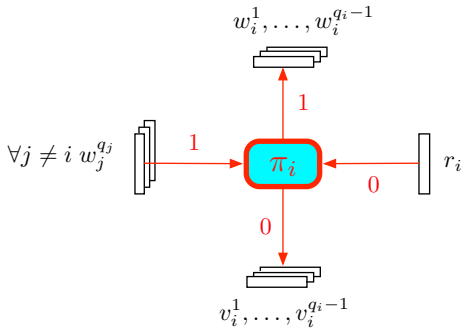


Figure 10. Net $\mathcal{P}_i$. Place π_i of this net latches the publications made by other nodes, for use in the control of node i.

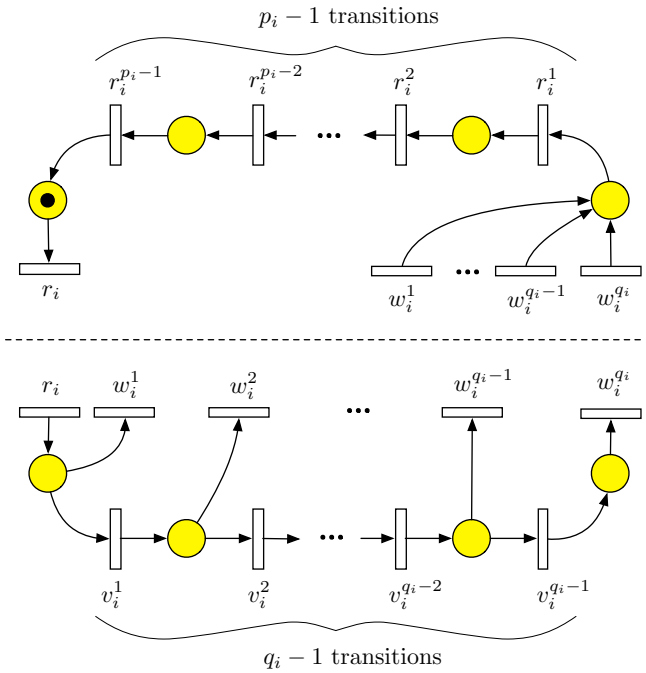


Figure 11. Half token ring for reads ($\mathcal{M}_i^r$, top) and writes ($\mathcal{M}_i^w$, bottom) by node i. Yellow places model inter-tick time (see Figures 6).

The incoming arrow

$$r_i \xrightarrow{0} \pi_i$$

indicates that firing transition r_i resets the boolean status in π_i to 0, to begin a new publication round; similarly, the incoming arrow

$$w_{q_j}^q \xrightarrow{1} \pi_i$$

for every $j{\neq}i$, indicates that firing transition w_j^q sets the boolean status in π_i to 1. It corresponds to the publication of another node that is first to publish. Vice-versa, the outgoing arrow

$$\pi_i \xrightarrow{1} w_i^q$$

for every $1{\leq}q{<}q_i$, indicates that firing transition w_i^q is guarded by the condition that the boolean status in π_i equals 1, i.e, a node already published a value. Observe that the action of firing transition $w_i^{q_i}$ is not guarded since there is no alternative to it. Similarly, the outgoing arrow

$$\pi_i \xrightarrow{0} v_i^q$$

for every $1{\leq}q{<}q_i$, indicates that firing transition v_i^q is guarded by the condition that the boolean status in π_i equals 0. It means that transitions v_i^q are fired whenever no publication by another node has been seen by node i.

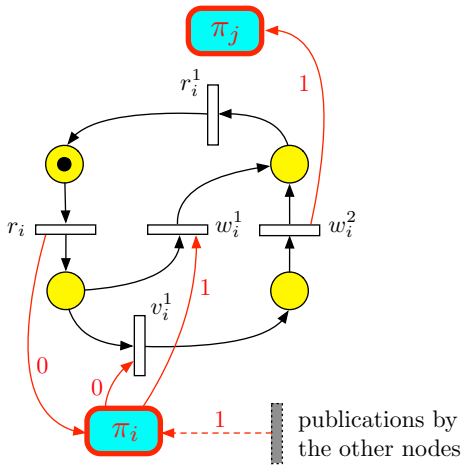


Figure 12. Result of performing (17) for the case of two or more nodes, $j{\neq}i$, and $p_i = q_i = 2$.

Net $\mathcal{M}_i^r$, top of Figure 11, models the reading part of the round. The progress of this net cannot be preempted, thus it corresponds to a simple counter stopping at value p. On the other hand net $\mathcal{M}_i^w$, bottom of Figure 11, models the writing part of the round. The longest path in this net can be preempted if a faster node publishes beforehand, i.e., if the boolean status π_i equals 1. Finally, publications by node i are broadcast to the status place of other nodes $\prod_{j\neq i} \mathcal{P}_j$.

We show in Figure 12 the instanciation of (17) for the case of two or more nodes and $p = q = 2$.

The network: The overall net modeling time-based LTTA is given by

$$\mathcal{M} \quad = \quad \left(\prod_i \mathcal{M}_i \right) \times \left(\prod_{j \to i} \mathcal{M}_{ji} \right) \tag{18}$$

Note that the time-based LTT Architecture involves no skipping mechanism.

A. *Preservation of synchronous semantics*

Let $\mathcal{L}_{\widehat{\mathcal{N}}}$ be the language of net $\widehat{\mathcal{N}}$ defined in (10) and similarly for $\mathcal{L}_{\mathcal{M}}$. Project $\mathcal{L}_{\mathcal{M}}$ over the subalphabet

$$\{r_i, w_i^1, \ldots, w_i^q \mid i = 1, \ldots, n\}$$

by erasing transitions not belonging to this set. Then, identify, in this projected language, the conflicting transitions w_i^k for $k = 1, \ldots, q$ by renaming them all w_i and finally call the resulting language $\widehat{\mathcal{L}}_{\mathcal{M}}$.

We then need to translate to net $\mathcal{M}$ assumptions 1–6 regarding the architecture. Assumption 1 is already taken into account by the net $\mathcal{M}_{ji}$ of Figure 9. Assumptions 3 and 4 are taken into account by having the tokens as in Figures 9 and 11. Assumption 6 is side information that needs not be reflected in the net. Finally, conditions (6) and (7) are reformulated in terms of the net $\mathcal{M}$:

(6') Having a holding time satisfying inequalities (6) for all places of nets $\mathcal{M}_i^r$ and $\mathcal{M}_i^w$ for every $i = 1 \dots n$, and

(7') (a) Referring to Figure 9, having a holding time satisfying inequality (7) for the blue place;

(b) Referring to Figure 10, having a latency satisfying inequality (7) between the setting and reading times, for all publication places π_i, for $i = 1 \dots n$.

If the above assumptions regarding net $\mathcal{M}$ are satisfied, then the following theorem holds, which expresses the preservation of synchronous semantics:

Theorem 3: The following conditions on integers p and q ensure that $\widehat{\mathcal{L}}_{\mathcal{M}} = \mathcal{L}_{\widehat{\mathcal{N}}}$:

$$p > \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \tag{19}$$

$$q > \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p\left(\frac{T_{\max}}{T_{\min}} - 1\right) \tag{20}$$

Proof: Keeping in mind Figure 7, the conclusion of Theorem 3 (namely, that $\widehat{\mathcal{L}}_{\mathcal{M}} = \mathcal{L}_{\widehat{\mathcal{N}}}$ holds) is an immediate consequence of the following two properties, which hold for every node $i = 1 \dots n$:

Property 1 (writes): The $(k-1)$th firing of one of the (conflicting) transitions $w_i^1 \dots w_i^q$ occurs only after transitions $r_j, j = 1 \dots n$ have all been fired $k-1$ times.

Property 2 (reads): The kth firing of transition r_i occurs only after after all places $j = 1 \dots n$ have been written $k-1$ times.

We prove the above two properties by induction over k. Thus, we assume they are true up to $k-1$.

Suppose the first $(k-1)$th write by some node occurs at real-time t. Then we claim that the last $(k-1)$th write by some (other) node occurs at the latest at time

$$\min\left(t + qT_{\max}\,,\; t + \tau_{\max} + T_{\max}\right) \tag{21}$$

The first term in the min corresponds to an "autistic" node i that sees no publication and thus writes by firing transition w_i^q after having performed its $(k-1)$th reading r_i, which must have occured before t by the induction hypothesis. To derive the second term in the min, pick a node that is the "last to awake": this node just missed the publication by the earliest node, which was made available at latest at $t + \tau_{\max}$. It can notice this publication at the latest within one period $T_{\max}$ and then it fires by publishing the result of its computation based on its $(k-1)$th read r_i.

Note that the result is a minimum between two values. Indeed, if $\tau_{\max} + T_{\max} < qT_{\max}$, we have $\tau_{\max} < (q-1)T_{\max}$. In this case, the slowest node should notice the first broadcast at $t + (q-1)T_{\max}$ (at the $(q-1)$th activation of the node). On the other hand, if $\tau_{\max} > (q-1)T_{\max}$, the node writes its value before noticing the first publication.

Using (7'-a), publications of values from the $(k-1)$th round over the channel are available for reading at latest at date (21)$+\tau_{\max}$, that is:

$$\min\left(t + qT_{\max}\,,\; t + \tau_{\max} + T_{\max}\right) + \tau_{\max} \tag{22}$$

On the other hand, the earliest kth read cannot occur before $t + pT_{\min}$. Hence condition (19), which expands to $pT_{\min} > 2\tau_{\max} + T_{\max}$, ensures that property 2 remains valid at the kth round. Then, the latest kth read cannot occur later than (21)$+T_{\max}$, that is

$$\begin{aligned} & t + \min\left(qT_{\max}\,,\; \tau_{\max} + T_{\max}\right) + pT_{\max} \\ \leq\; & t + \tau_{\max} + (p+1)T_{\max} \end{aligned} \tag{23}$$

Finally, the earliest kth write cannot occur earlier than $qT_{\min}$ after the earliest kth read. Hence it cannot occur before $t + pT_{\min} + qT_{\min}$. This value is then published no earlier than $\tau_{\min}$ later. Hence condition (20), which expands as $(p+q)T_{\min} + \tau_{\min} > \tau_{\max} + (p+1)T_{\max}$, ensures that property 1 remains valid at the kth round. ◇

B. Performance bounds

Performance bounds are easily derived from the conditions of Theorem 3, which are assumed to be in force:

Theorem 4: The Worst case throughput $\lambda_{\mathcal{M}}$ of net $\mathcal{M}$ is given by $1/\lambda_{\mathcal{M}} = (p_\star + q_\star)T_{\max}$, where $p_\star$ and $q_\star$ are the optimal values for p and q according to inequalities (19) and (20).

C. Issues of blocking communication

Net $\mathcal{M}_{ji}$ exhibits no blocking read. On the other hand, net $\mathcal{M}_i$ defined in (17) possesses a circuit shown in red in Figure 12. This circuit, however, involves no synchronization outside node i. It can therefore never be blocked. Hence, net $\mathcal{M}$ is free from blocking communication between different nodes. If a node or communication link fails by becoming silent, then the nodes reading from that node or that link will just proceed with their local computations using old data from the local CbSbuffer, in combination with fresh data from live links and nodes. Time-Based LTTA is thus fully non-blocking, although the application enters a degraded mode in the case of silent failure of a node or link, by using outdated data from its failed input links.[1]

VI. Hybrid LTT Architectures

In this section we compare the above two types of LTT Architectures and study their blending.

[1]The semantics of the original program is not preserved in the degraded mode.

A. Discussion and comparison

Throughput: Let us first start with some comparisons regarding throughput. Firstly, the lower bound for throughput in the Back-Pressure architecture is always given by Theorem 2, that is $1/\lambda_{\mathcal{N}} = 2\,(T_{\max} + \tau_{\max})$. On the other hand, from Theorem 4, performance of the time based architecture depends on the type of situation:

For moderately distributed real-time control: delay $\tau_{\max}$ and jitter $T_{\max} - T_{\min}$ are non-zero but small relative to the nominal period $T_{\max}$. In this case, Theorem 3 yields $p_\star = q_\star = 2$, and thus $1/\lambda_{\mathcal{M}} = 4T_{\max}$. Note that transmission delays do not matter as long as they remain small.

For widely distributed real-time control: i.e., when communications are distant, reflected by $T_{\max}/T_{\min} \approx 1$ and $\tau_{\max}/\tau_{\min} \approx 1$ but $\Delta =_{\text{def}} \tau_{\max}/T_{\min} \gg 1$, we end up having $p_\star \approx 2\Delta, q_\star \approx 1 \ll \Delta$ and thus $1/\lambda_{\mathcal{M}} \approx 2\tau_{\max} \approx 1/\lambda_{\mathcal{N}}$. Hence performances of Back-Pressure LTTA are comparable to the ones of Time-Based LTTA for distant communications.

Robustness: Recall that net $\widehat{\mathcal{N}}$ is an abstraction for communication in Back-Pressure architecture. This net is subject to blocking communication. This means that if one node gets stuck, then all nodes will keep skipping forever, thus computing with outdated constant values and outputting nothing. The overall application is then stuck, even though not all computing nodes are blocked. This implies that Time-Based monitoring must be added on top of the architecture, e.g., by means of watchdogs that can be used by neighbor nodes to detect the fail-stop of one node. This, however, causes slow-down and loss of performance.

In contrast, the Time-Based protocol can still survive in degraded mode without any slow-down when a node has experienced fail-stop. Outdated values will be used by neighboring nodes but the rest of the system continues to function.

Flexibility: Back-Pressure architectures are very flexible since semantics preservation does not depend on their timing characteristics. In particular, hardware characteristics can be changed without retuning the Back-Pressure protocol. The same holds regarding the addition or removal of nodes and links in the application or architecture.

Summary: Since the two architectures are similar in timing performance, the preferred architecture depends on the relative importance of robustness versus flexibility for the application at hand. Since complex applications typically combine both cases for different parts of the system, it makes sense to consider blending the two architectures. How can this be done?

B. Blending the two architectures

In the architecture, partition the links into Time-Based ones and Back-Pressure based ones. Nodes that are adjacent to at least one Time-Based link are marked Time-Based as well and are implemented according to net $\mathcal{M}_i$ defined in (17). Other nodes are marked Back-Pressure based and are implemented according to net $\mathcal{N}_i$ of Figure 6. For the links, several cases occur. Homogeneous links, for which all adjacent nodes are of the same kind, are implemented according to net $\mathcal{N}_{ji}$ of Figure 5 or net $\mathcal{M}_{ji}$ of Figure 9, depending on the kind. Now, heterogeneous links, which are adjacent to nodes of different kinds, must be slightly adapted. For a heterogeneous link $j \to i$ ending at a node i marked Time-Based, its associated net is obtained by equipping net $\mathcal{N}_{ji}$ with a skipping mechanism at its sink transition labeled r_i; this mechanism ensures that node i can perform reads at its own pace, according to the Time-Based protocol. Symmetrically, for a heterogeneous link $j \to i$ originating from a node j marked Time-Based, its associated net is obtained by equipping net $\mathcal{N}_{ji}$ with a skipping mechanism at its source transition labeled w_j; this mechanism ensures that node j can perform its writes at its own pace, according to the Time-Based protocol. In the resulting hybrid architecture, clocks of Time-Based nodes and delays of Time-Based links are subject to the conditions of Theorem 3. In contrast, no condition is required for the clocks and delays of the Back-Pressure part — of course, the performance depends on them.

VII. Conclusion

H. Kopetz' TTA was the first proposal for a MoCC-based architecture suited to distributed hard real-time systems involving feedback control. Of course, as explained in, e.g., [18], TTA cannot be used as the single architectural paradigm in a complex, multi-layered, embedded system.

LTTA was proposed as a relaxed version of TTA for the very same layers. In fact, the objective of LTTA is to offer an abstraction that emulates TTA. This paper unifies the work done on LTTA [24], [10], by proposing a single framework in which the two existing variants of LTTA can be cast. This study reveals that Back-Pressure based LTTA is more flexible but less robust against failures than Time-Based LTTA. It therefore makes sense to use different versions of LTTA for different parts of the system. We thus propose a way of blending the two architectures while maintaining the essential properties of preservation of semantics. Further work is needed, regarding time based LTTA, to address heterogeneous infrastructures where ensuring the global bounds arising in Assumption 5 may be a problem.

Acknowledgement: The authors are indebted to Alberto Sangiovanni-Vincentelli for inspiring discussions, to Benoît Caillaud for the precise form of nets $\mathcal{P}_i$ and to Timothy Bourke for his kind and valuable remarks.

References

[1] K.-E. Årzén, "Timing analysis and simulation tools for real-time control," in *Formal Modeling and Analysis of Timed Systems*, ser. LNCS, P. Pettersson and W. Yi, Eds. Springer, 2005, vol. 3829.

[2] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity: An Algebra for Discrete Event Systems*. Wiley, 1992.

[3] A. Benveniste, P. Caspi, M. di Natale, C. Pinello, A. Sangiovanni-Vincentelli, and S. Tripakis, "Loosely Time-Triggered Architectures based on Communication-by-Sampling," in *EMSOFT*, 2007, pp. 231–239.

[4] A. Benveniste, "Loosely Time-Triggered Architectures for Cyber-Physical Systems," in *DATE*, 2010.

[5] A. Benveniste and G. Berry, "The synchronous approach to reactive and real-time systems," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1270–1282, 1991.

[6] A. Benveniste, A. Bouillard, and P. Caspi, "A unifying view of loosely time-triggered architectures," in *EMSOFT*, L. P. Carloni and S. Tripakis, Eds. ACM, 2010, pp. 189–198.
[7] A. Benveniste, P. Caspi, P. L. Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis, "A protocol for loosely time-triggered architectures." in *EMSOFT*, 2002, pp. 252–265.
[8] L. P. Carloni, "The role of back-pressure in implementing latency-insensitive systems," *Electronic Notes in Theorical Computer Science*, vol. 146, no. 2, pp. 61–80, 2006.
[9] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1059–1076, 2001.
[10] P. Caspi and A. Benveniste, "Time-robust discrete control over networked loosely time-triggered architectures," in *IEEE Control and Decision Conference*, December 2008.
[11] J. Cortadella and M. Kishinevsky, "Synchronous elastic circuits with early evaluation and token counterflow," in *DAC*, 2007, pp. 416–419.
[12] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1904–1921, 2006.
[13] R. Cuninghame-Green, *Minimax Algebras*, ser. Lecture notes in economics and mathematical systems. Springer Verlag, 1979, vol. 166.
[14] M. Gondran and M. Minoux, *Graphs, Dioids and Semirings*, ser. Interface series. Springer Verlag, 2008, vol. 41.
[15] K. C. Gorgônio, J. Cortadella, F. Xia, and A. Yakovlev, "Automating synthesis of asynchronous communication mechanisms," *Fundam. Inform.*, vol. 78, no. 1, pp. 75–100, 2007.
[16] G. Kahn, "The semantics of a simple language for parallel programming," in *Information Processing 74, Proceedings of IFIP Congress 74*. Stockholm, Sweden: North-Holland, 1974.
[17] H. Kopetz, *Real-Time Systems*. Kluwer Academic Publishers, 1997.
[18] ——, "The complexity challenge in embedded system design," in *ISORC*, 2008, pp. 3–12.
[19] C. Kossentini and P. Caspi, "Approximation, sampling and voting in hybrid computing systems." in *HSCC*, 2006, pp. 363–376.
[20] M. di Natale and A. Sangiovanni-Vincentelli, "Moving from Federated to Integrated Architectures: the role of standards, methods, and tools," *Proc. of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.
[21] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[22] J. Romberg and A. Bauer, "Loose synchronization of event-triggered networks for distribution of synchronous programs." in *EMSOFT*, 2004, pp. 193–202.
[23] P. Traverse, I. Lacaze, and J. Souyris, "Airbus fly-by-wire: A total approach to dependability," in *IFIP World Congress, Toulouse*. IFIP, 2004.
[24] S. Tripakis, C. Pinello, A. Benveniste, A. L. Sangiovanni-Vincentelli, P. Caspi, and M. D. Natale, "Implementing synchronous models on loosely time triggered architectures," *IEEE Trans. Computers*, vol. 57, no. 10, pp. 1300–1314, 2008.

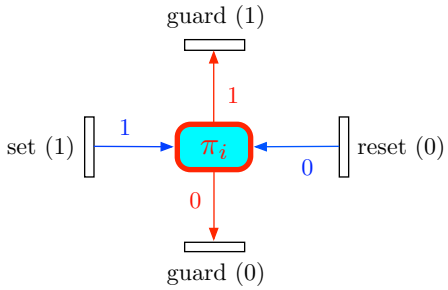


Figure 13. A boolean register. One can test the value (guard) or modify the value (set and reset).

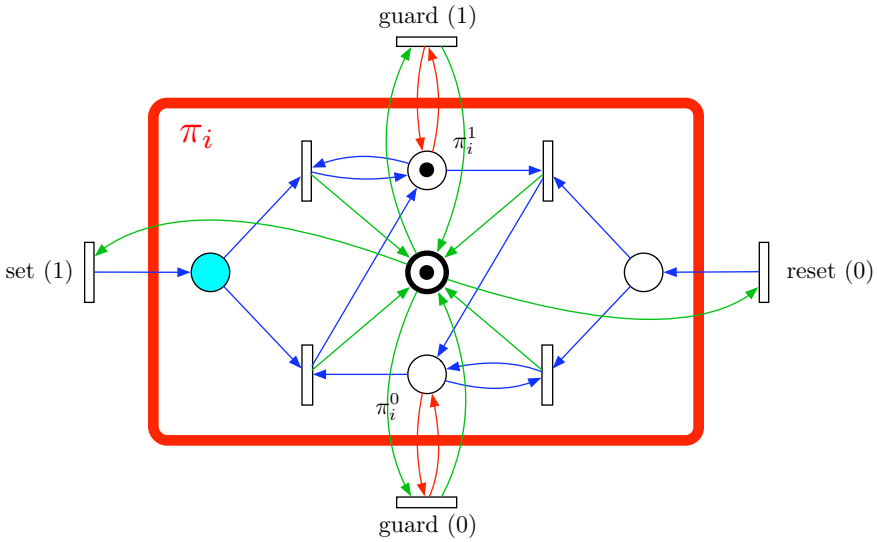


Figure 14. Petri net model of a boolean register.

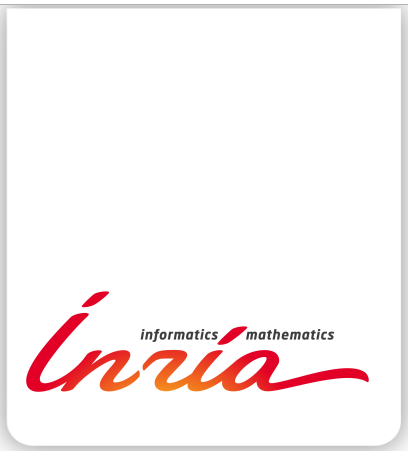
APPENDIX
MODEL OF A BOOLEAN REGISTER AS A 1-SAFE PETRI NET

We show here how to model boolean registers π_i (see Section V) as Petri nets. Figure 13 shows an example of a boolean register. Note that there are two kinds of transitions. Guards test the value of the register (1 or 0), whereas set and reset transitions modify the value of the register. We show only one transition of each kind but there may be several guards, and several set and reset transitions.

Figure 14 presents the model of the register as a Petri net. The idea is to rely on two mutually exclusive places π_i^1 and π_i^1. If the token is in place π_i^1 the value of the register is true. On the other hand, if the token is in place π_i^0 the value of the register is false. Thus, the guards simply test the presence of a token in these places (see the red edges).

For set and reset transitions we need to consider two cases. Consider a set transition (reset transitions are handled in the same way). If the token is in place π_i^1 the transition consumes and replaces the token. On the other hand, if the token is in the place π_i^0, the transition consumes the token and places it in the π_i^1 place (see the blue edges). Note that the left most place is blue. Indeed we can associate to this place a holding time to model the publication delay (see Section V-A, assumption (7')(b)). The left most place can remain instantaneous since resets are done locally and do not transit through the network.

The last problem is to avoid concurrent accesses to the register. This is the role of the green edges. The central thick place acts like a lock and ensures atomic accesses to the register.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399