A unifying view of Loosely Time-Triggered Architectures [*]

Albert Benveniste
INRIA, Campus de Beaulieu
35042 Rennes cedex, France
albert.benveniste@inria.fr

Anne Bouillard
ENS Cachan (Bretagne)
Campus de KerLann
35170 Bruz, France

Paul Caspi
VERIMAG/CNRS
2 avenue de Vignate
38610 Gières, France

ABSTRACT

Cyber-Physical Systems require distributed architectures to support safety critical real-time control. Hermann Kopetz' Time-Triggered Architectures (TTA) have been proposed as both an architecture and a comprehensive paradigm for systems architecture, for such systems. To relax the strict requirements on synchronization imposed by TTA, Loosely Time-Triggered Architectures (LTTA) have been recently proposed. In LTTA, computation and communication units at all triggered by autonomous, non synchronized, clocks. Communication media act as shared memories between writers and readers and communication is non blocking. 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.

Categories and Subject Descriptors

C.0 [**General**]: Modeling of computer architecture

General Terms

Theory

Keywords

Time-Triggered Architecture, Elastic Circuits, Loosely Time-Triggered Architecture

1. INTRODUCTION

Embedded electronics for safety critical systems has 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 controller, and its dedicated set of wires. This architecture, referred to as *Federated Architecture*, has proved safe and robust by ensuring built-in partitioning between the 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 to shifting to *Integrated Architectures* [18] where several functions are hosted in a same 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 reduction in computing devices and wiring. While this move from Federated to Integrated Architectures opens new possibilities for further increase 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, mismatch and failure to meet overall requirements emerges as a high risk at that very late stage 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 clear at all how detailed design 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 [15, 16]. TTA builts on a vision of the system in which physical time is seen as a first class citizen and as a help, not an ennemy. 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 overall 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 use in these industrial sectors, are examples of formalisms

[*]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.

obeying the synchronous MoCC. In addition, TTA offers more possibilities to address the above discussed difficulties. Firstly, time can be used as a help in building fault tolerance services with its redundancy management and fault detection and mitigation. Secondly, time is also a help for partitioning, and for integrating components visible through their interfaces: Time Division Multiplexing (TDM) is a well established technique to grant a function access to communication or computing. 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 [2]. 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) [5]. LTTA is characterized by a communication mechanism, called *Communication by Sampling* (CbS), which assumes that: 1/ writings and readings 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 [20], 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 not blocking both for writes and reads. Hence, 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 [2]. However, data can be lost due to overwrites or alternatively duplicated because reader and writer are not synchronized [1, 21, 8]. 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 [5] and studied for a single writer-reader pair; [19] proposes a variation of LTTA where some master-slave re-synchronization of clocks is performed. LTT architecture of general topology was studied in [1, 21], using techniques reminiscent from back-pressure [7, 6] and elastic circuits [9]. In a different direction, [17] developed an alternative approach where upsampling is used in combination with "thick" events as a way to preserve semantics. This approach, which is more time-based as compared to [1, 21], was further developed and clarified in [8].

In this paper, we cast the two variants of LTTA in a unified framework. We simplify the analyses of [21] and we compare the respective merits of these two variants. Finally we advocate blending them for different layers of the architecture and show how this can be safely done.

The paper is organized as follows. LTTA and Communication by Sampling are presented in Section 2. This is followed in Section 3 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 4 and *time* based in Section 5. The two architectures are compared in Section 6 and we discuss why it would make sense blending them, and how this can be performed. Finally the simplifying assumption we consider is relaxed in Section 7.

2. LTTA AND ITS ARTIFACTS

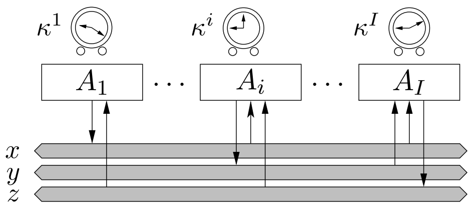

Figure 1: Communication by Sampling (CbS). For each variable x, y, or z, there is one shaded bus behaving as a shared memory, one writer and zero or more readers.

LTTA relies on *Communication by Sampling*, which is illustrated on 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;*
3. *Writings and readings 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 to 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 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

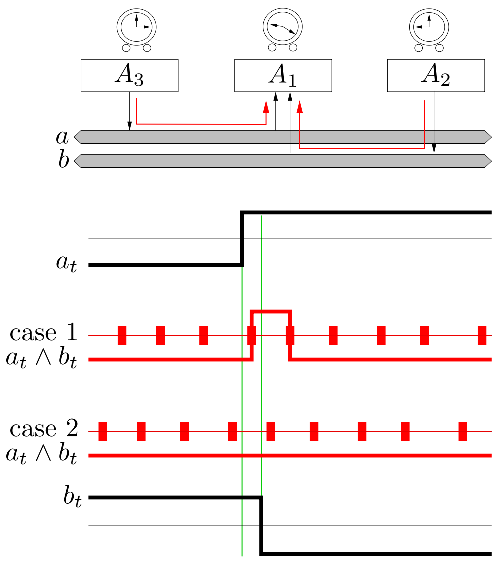

Figure 2: Sensing multiple signals, for distributed clocks subject to independent drifts and jitters. Referring to Figure 1, we show the case of A_1 reading two boolean inputs originating from A_2 and A_3, respectively, and computing their conjunction. Cases 1 and 2 correspond to two different outcomes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

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 for a_t and b_t. Increasing the clock rate cannot prevent this from occurring.

Due to the above artifacts, the semantics may not be preserved while deploying an application over an LTT architecture and protocols are needed to cope with this issue. These are presented and studied in Section 4 and subsequent ones.

3. DEPLOYMENT AND SEMANTICS PRESERVING

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

3.1 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:

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

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 Lustre or Signal clock calculi. "Absence" values are not published.

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

$$S_1 \parallel \ldots \parallel S_I. \tag{2}$$

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

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

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 decomposition $S = S_1 \parallel \ldots \parallel S_I$, 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 $\preceq$. For z a vertex of $\mathcal{G}(S)$, define its *level* $\ell(z)$ as being

$$\begin{array}{l} \text{the largest index } \ell \geq 0 \text{ such that a chain} \\ z_0 \prec z_1 \prec \cdots \prec z_\ell = z \text{ exists in } \mathcal{G}(S). \end{array} \tag{4}$$

The level is illustrated on Figure 3.

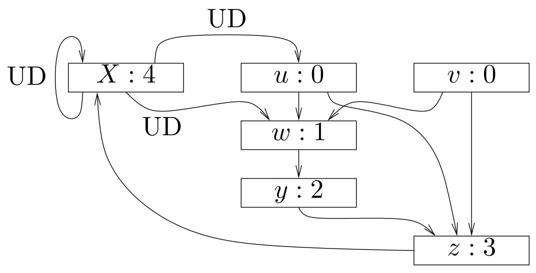

Figure 3: Illustrating the level of a node.

To simplify our study, we consider the following additional assumption (this assumption will be relaxed in section 7):

ASSUMPTION 4. *Every communication between different sites is subject to a unit delay or more.*

This corresponds to a level 0 for all input nodes of the different systems, see formula (4) and Figure 3.

3.2 LTTA Deployment and Semantics Preserving

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 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 need to schedule the computation of the different variables by respecting partial order $\preceq$. This leaves room for computing different variables at different clock ticks.

All the above designs implement correctly 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 extensively discussed in Section 2, CbS communication by itself will not offer this. In the next two sections we propose two different protocols on top of the CbS infrastructure to ensure semantics preserving. We now complete this section with further assumptions and notations.

3.3 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: $\forall k \in \mathbb{N}$,*

$$T_{\min} < \kappa^i_k - \kappa^i_{k-1} < T_{\max}\,, \tag{5}$$

and communication delays are bounded from above:

$$\tau < \tau_{\max}. \tag{6}$$

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 preserves the semantics. The first protocol is an adaptation of elastic circuits in hardware, and the second one is a softening, time based, adaptation of the original TTA. For each of them we indicate the needed assumptions.

4. 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.

Elastic circuits were proposed in [10, 13, 9] as a semantic preserving architecture in which Kahn Process Network [14] type of execution is performed using bounded buffers. This is achieved by relying on a mechanism of *back-pressure* [6] by which readings from a buffer by a node is acknowledged to the writer using a reversed virtual buffer. Petri net $\mathcal{N}_{ji}$ of Figure 4 depicts how a link $j \to i$ with a 2-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 $j \to i$.

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 quasi periodic clocks as in LTTA. To adapt to the constraints of LTTA, the authors of [21] 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 slight deviation from Petri net semantics will be needed to capture priority arising in certain conflicts.

Modeling the links. Figure 4 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.

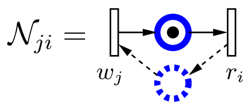

Figure 4: Back-pressure net $\mathcal{N}_{ji}$ associated to each directed link $j \to i$ of the architecture. For performance studies of Section 4.2, the time spent in the place depicted in thick blue models transmission delay.

Modeling the nodes. Reactions at node i are captured by the net $\mathcal{N}_i$ shown on Figure 5, which is composed of a two-step "read; write" together with a local skipping mechanism. The following holds regarding this skipping mechanism, which was proposed in [21] 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. (7)

Transitions with labels r_i and w_i have priority over transition with label $skip_i$. (8)

Observe that, while net $\widehat{\mathcal{N}}_i$ in isolation evolves according to a purely logical time, net $\mathcal{N}_i$ is triggered by the local clock κ^i of the considered node.

The composition indicated on figure 5 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 4 and 5, consider the following two product nets

$$\widehat{\mathcal{N}} = \left(\prod_i \widehat{\mathcal{N}}_i\right) \times \left(\prod_{j\to i} \mathcal{N}_{ji}\right) \tag{9}$$

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

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 applica-

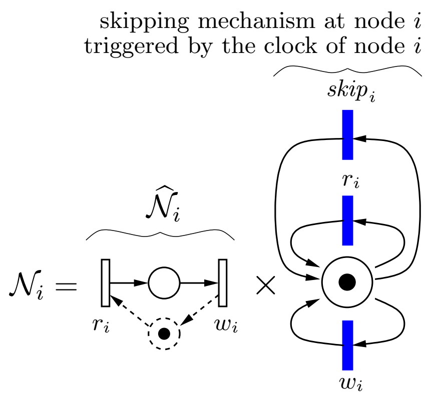
skipping mechanism at node i
triggered by the clock of node i
$skip_i$
$\widehat{\mathcal{N}}_i$
$\mathcal{N}_i =$
r_i
w_i
$\times$
r_i
w_i

Figure 5: Net $\mathcal{N}_i$ showing the mechanism of skipping at node i. For performance studies of Section 4.2, inter-tick time is captured by a non-zero duration attached to blue transitions.

tion 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.

4.1 Preservation of synchronous semantics

This result was first proved in [21], for more general architectures. We give here a very simple and direct proof, by explicitly using the associated elastic circuit $\widehat{\mathcal{N}}$.

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; 2-bounded buffers can be used on the links since direct links all have a logical 1-delay by Assumption 4.[1] Observe that this first property only relies on assumptions 3–6; it does not use Assumption 1 nor conditions (5,6) 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 is assumed:

it is not possible that transition $skip_i$ of the skipping mechanism fires repeatedly for ever. (11)

Using (11), 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. $\diamond$

Observe that fairness condition (11) is indeed much weaker than the conjunction of (7) and Assumption 5.

4.2 Performance bounds

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

[1] This is a well known result. For the sake of completeness, we provide in appendix A a detailed proof of this.

Let us first focus on elastic circuit $\widehat{\mathcal{N}}$ defined in (9). Assume the following conditions for this elastic circuit — they are similar to (5) and (6):

(5') there exist lower and upper bounds $\overline{T}_{\min}$ and $\overline{T}_{\max}$ for the interval between any two successive firings of a black transition related to node i (it can be a read r_i or a write w_i); this is captured by assigning these bounds for the duration of the two transitions w_j and r_i in Figure 4.

(6') the time spent in direct or back-pressure places of $\widehat{\mathcal{N}}_{ji}$ is bounded by $\overline{\tau}_{\max}$, for any link $j \to i$ (direct places are the solid ones in figure 4, whereas back-pressure places are dashed). Corresponding bound is assigned to the time spent in the place depicted in thick blue in Figure 4.

Following classical results on event graphs [12] (Chapter 6.7, p247), [3] (Chapter 2.5) or (max,plus) algebras [11] (Chapters 21-26), 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:

$$1/\lambda_{\widehat{\mathcal{N}}} \; = \; 2\left(\overline{T}_{\max} + \overline{\tau}_{\max}\right) \tag{12}$$

Next, net $\mathcal{N}$ consists in adding, at each node i of net $\widehat{\mathcal{N}}$, the skipping mechanism shown on figure 4. Now, assume that conditions (5) and (6) hold for net $\mathcal{N}$, namely:

(5) 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;

(6) the time spent in any place of $\mathcal{N}_{ji}$ is bounded by $\tau_{\max}$ 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 (5') and (6'):

$$\begin{aligned} \overline{T}_{\max} &= T_{\max} &(13)\\ \overline{\tau}_{\max} &= T_{\max} + \tau_{\max} &(14) \end{aligned}$$

Indeed, bound (13) is reached by node i having slowest clock, since this node does not need to skip. Bound (14) is reached when the latest token reaches an input place of node i but net $\mathcal{N}_i$ fired just before.

Combining (5',6'), (12), and (13,14) yields:

THEOREM 2. *The worst case throughput $\lambda_{\mathcal{N}}$ of net $\mathcal{N}$ is*

$$1/\lambda_{\mathcal{N}} = 4T_{\max} + 2\tau_{\max}$$

Recall that Theorem 1 only requires fairness condition (11) and Theorem 2 only requires the upper bounds in (5,6), but not the lower bounds. Performance results are provided in [21] for more general architectures (with arbitrary buffer sizes). However, the proof we give here is much more straightforward. General architectures are analysed in Section 7.1.

4.3 Issues of blocking communication.

The skipping mechanism ensures that computing nodes themselves never get blocked due to the failure of other nodes or communication. However, net $\widehat{\mathcal{N}}$ exhibits blocking read 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 propose and analyse in the next section.

5. TIME-BASED LTTA

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

Time-based LTTA relies on an original idea of P. Caspi [17, 8]. Aim of this protocol is to ensure a clean alternation of writing and reading phases throughout the architecture, see Figure 6. The synchronization principle used in time-based LTTA is illustrated on Figure 7.

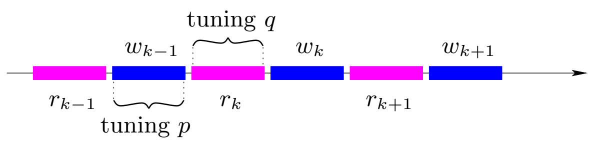

Figure 6: Alternating phases of reads and writes. The right arrow figures the time line and k is the reaction index.

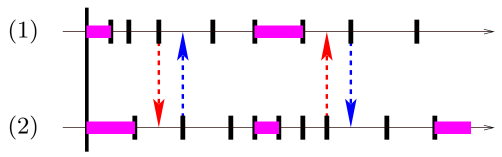

Figure 7: Synchronization principle used in time-based LTTA.

Figure 7 shows two time lines, for two communicating nodes (1) and (2). Ticks of the local clocks are figured by the short thick vertical bars (note the jitter). Magenta rectangles depict reading-and-computing periods; they correspond to r_{k-1}, r_k, r_{k+1} in Figure 6. At the beginning, node (1) is the fastest. Thus, it waits for a certain amount of ticks and then it publishes its computed value; this is indicated by the red dashed arrow pointing to (2). Upon noticing this publication, node (2) responds with its own publication, indicated by the blue dashed arrow pointing to (1). Meanwhile, node (1) keeps frozen to make sure that node (2) had 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 upon time. They rather react to noticing a publication by another node. The key observation is that fast nodes slow down by waiting a number p of ticks of their local clocks, whereas slow nodes accelerate by actively synchronizing over fast nodes' publications. To summarize, in order to achieve sufficient synchronization to preserve synchronous semantics, tokens are used for speeding-up only whereas time is used for slowing-down. The key issue is to tune p to the smallest value that is sufficient to ensure Figure 6 (actually we will use different waiting times for the writing and reading periods). The architecture implementing this protocol is detailed next, by successively describing its links and nodes, using again the same Petri net framework.

Modeling the links. Figure 8 shows net $\mathcal{M}_{ji}$, which models CbS communication for directed link from node j to node i — note the *read arc*[2] ingoing to transition r_i. In this net, reads and writes can occur concurrently and asynchronously.

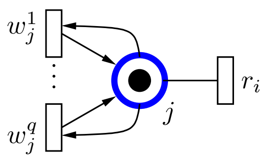

Figure 8: Net $\mathcal{M}_{ji}$ modeling directed CbS link from node j to node i. Note the non-directed *read arc* adjacent to transition r_i. Compare with net $\mathcal{N}_{ji}$ of figure 4. For colors and thickness of circles, see figures 4 and 5.

Modeling the nodes. In time-based LTTA the synchronization is local to each node and is a mix of time- and token-based synchronization. It involves the nets shown in figures 9 and 10, which are commented next.

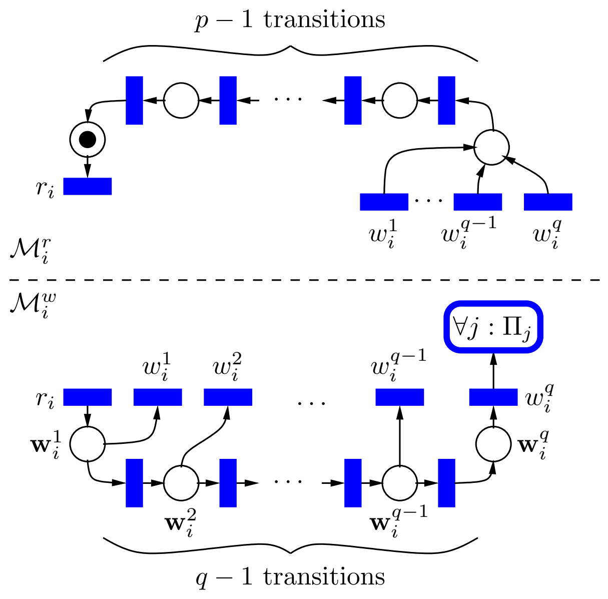

Figure 9: Half token ring for reads ($\mathcal{M}_i^r$, top) and writes ($\mathcal{M}_i^w$, bottom) by node i. For colors and thickness of circles, see figures 4 and 5.

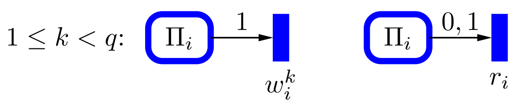

Figure 10: Nets $\mathcal{P}_{ik}^w, k \in \{1, \ldots, q-1\}$ (left) and $\mathcal{P}_i^r$ (right). For colors and thickness of circles, see figures 4 and 5.

Focus on the half token-ring $\mathcal{M}_i^w$ of Figure 9, which controls writes by node i. Label "$\forall j : \Pi_j$" indicates that a token is put in special place labeled Π_j for every j. Observe that the places labeled $\mathbf{w}_i^j, j = 1 \ldots q-1$, possess two conflicting

[2]This means that transition r_i can fire if and only if there is a token in place j, but this token is not consumed by transition r_i.

transitions in their postset, namely a private one (without label) and w_i^j, for $j = 1 \dots q-1$, respectively. The following convention applies regarding these conflicts:

enabled transition $w_i^1, w_i^2, \dots$, or w_i^{q-1}, has priority over the conflicting private transition. (15)

The special square-round shaped place Π_j exists for all j. We take the special convention that *tokens are overwritten* (not added) in the special place Π_j, so this place holds 0 or 1 token. Place Π_j stores the information that some node has published after having spent enough ticks.

Focus next on figure 10. Publications are controlled by the nets $\mathcal{P}_{ik}^w$, $k \in \{1, \dots, q-1\}$ (left) and $\mathcal{P}_i^r$ (right). The labels on the arcs indicate the amount of tokens needed in the preset of this arc for the postset transition to fire — label 0, 1 means "0 or 1". Each firing consumes the token, if any. Thus the special place Π_i has its token consumed by node i when writing strictly before last stage q. In contrast, when writing at last, the special place Π_i is reset whatever its content is — this is to avoid a node self-reacting to its own publication.

The protocol sitting at node i is then modeled by the net

$$\mathcal{M}_i \;=\; \mathcal{M}_i^r \times \mathcal{M}_i^w \times \mathcal{P}_i^r \times \left(\prod_{k=1}^{q-1} \mathcal{P}_{ik}^w \right) \tag{16}$$

where the product is by superimposing nodes (*both transitions and places*) with identical label.[3] Nodes with no label are considered private. Regarding triggering policy in net $\mathcal{M}_i$, the following holds:

When enabled, transitions are triggered by the local clock of node i. (17)

We show in Figure 11 the result of performing product (16) for the case of two nodes $i = 1, 2$, and $p = q = 2$.

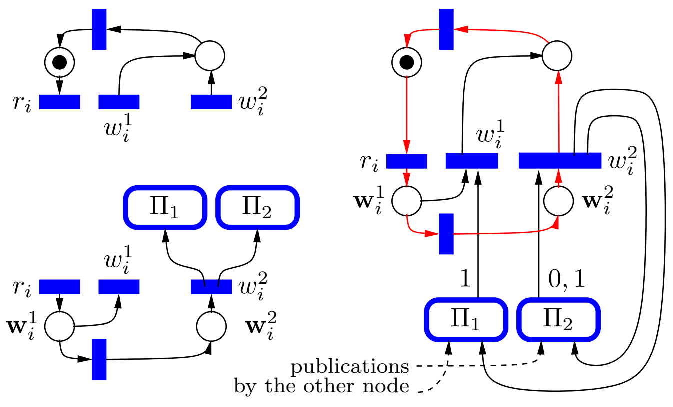

Figure 11: Result of performing (16) for the case of two nodes $i = 1, 2$, and $p = q = 2$. Left: counterpart of Figure 9. Right: the product (16). For colors and thickness of circles, see figures 4 and 5.

The role of the different components of $\mathcal{M}_i$ is as follows: considering the right hand side of (16), product $\mathcal{M}_i^r \times \mathcal{M}_i^w$ is responsible for slow-down based on local clocks (through the combination of (17) and the additional internal transitions), and product $\mathcal{P}_i^r \times (\prod_{k=1}^{q-1} \mathcal{P}_{ik}^w)$ implements speed-up by synchronizing over publications by other nodes.

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

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

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

5.1 Preservation of synchronous semantics

Let $\mathcal{L}_{\widehat{\mathcal{N}}}$ be the language of net $\widehat{\mathcal{N}}$ defined in (9) and similarly for $\mathcal{L}_{\mathcal{M}}$. Project $\mathcal{L}_{\mathcal{M}}$ over subalphabet

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

by erasing transitions not belonging to this set. Then, identify, in this projected language, the conflicting transitions w_i^k for $k = 1, \dots, q$ by renaming them all w_i and we 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 and conditions (5) and (6) regarding clocks and communication delays. Assumption 1 is already taken into account by having the read arc in the net $\mathcal{M}_{ji}$ of Figure 8. Assumptions 3 and 4 are taken into account by having the tokens as in figures 8 and 9. Assumption 6 is a side information that needs not be reflected in the net. Finally, conditions (5) and (6) are reformulated in terms of the net $\mathcal{M}$:

(5') having a firing duration satisfying inequalities (5) for all transitions of nets $\mathcal{M}_i^r$ and $\mathcal{M}_i^w$ for every $i = 1 \dots n$, and

(6') having a sojourn time satisfying inequalities (6) for all places capturing node-to-node communication, namely:

- the place of net $\mathcal{M}_{ji}$ for each link $j \to i$;
- publication places Π_i, for $i = 1 \dots n$.

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

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

$$p \;\geq\; \frac{\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} \tag{19}$$

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

PROOF. 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 $i = 1 \dots n$, see Figure 6:

PROPERTY 1 (READS). *The kth firing of transition r_i occurs only after after all places $j \in \{1 \dots n\}$ as in Figure 9 have been written $k-1$ times.*

PROPERTY 2 (WRITES). *The kth 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 times.*

[3] Said differently, the product net is obtained by taking the union of the different nets, seen as directed labeled bi-partite graphs.

We prove the above two properties by induction over $k > 0$. Assume they are true up to $k-1$.

Suppose the first $(k-1)$st writing by some node occurs at real-time t. Then we claim that the last $(k-1)$st writing by some node occurs at 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 $(k-1)$st reading r_i, which must have occured before t by induction hypothesis. To derive the second term in the min, pick a node that is "latest 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 latest within one period $T_{\max}$ and then it fires.

Then, the earliest kth reading cannot occur before $t + pT_{\min}$. Hence condition (19), which expands as $pT_{\min} \geq \tau_{\max} + T_{\max}$, ensures that property 1 keeps valid at the kth round. On the other hand, the latest kth reading cannot occur later than

$$\begin{aligned} & t + \min\left(qT_{\max}\,,\; \tau_{\max} + T_{\max}\right) + pT_{\max} \\ \leq\;\; & t + \tau_{\max} + (p+1)T_{\max} \end{aligned} \tag{22}$$

Finally the earliest kth writing cannot occur earlier than $qT_{\min}$ after the earliest kth reading. Hence it cannot occur before $t + pT_{\min} + qT_{\min}$. Hence condition (20), which expands as $qT_{\min} \geq \tau_{\max} + p(T_{\max} - T_{\min})$, ensures that property 2 keeps valid at the kth round. $\diamond$

5.2 Performance bounds

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

THEOREM 4. *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).*

5.3 Issues of blocking communication

Since net $\mathcal{M}_{ji}$ has a read arc, it exhibits no blocking read. On the other hand, net $\mathcal{M}_i$ defined in (16) possesses a circuit shown in red in Figure 11. 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 getting silent, then the nodes reading from that node or that link will just proceed to their local computations using old data provided as backups by CbS communication, 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 case of silent failure of a node or link, by using outdated data from its failed input links.[4]

6. HYBRID LTT ARCHITECTURES

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

6.1 Discussion and comparison

[4]Strict semantics preserving fails to hold in the degraded mode.

Throughput

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

For low-level safety-critical real-time control, delay and jitter are non-zero but small relative to nominal period. This then yields $1/\lambda_{\mathcal{M}} = 4T_{\max}$. Observe that this outperforms back-pressure architecture and that transmission delays do not matter as long as they remain small.

Another situation of interest is when communications are distant, reflected by $\frac{T_{\max}}{T_{\min}}$ close to 1 but $\frac{\tau_{\max}}{T_{\min}}$ significantly larger than 1, so that the latter dominates the former. Then we have $1/\lambda_{\mathcal{M}} \approx 2(\frac{\tau_{\max}}{T_{\min}})^2\tau_{\max}$, which is close to $1/\lambda_{\mathcal{N}}$.

Robustness

Recall that net $\widehat{\mathcal{N}}$ is the right 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 for ever, thus computing with outdated constant values and outputting nothing. The overall application is then stuck, despite computing nodes are not 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, time based architecture can still survive in degraded mode without any slow-down in case a node has experienced fail-stop. Outdated values will be used by its neighboring nodes but the rest of the system is still at work.

Flexibility

Back-pressure architectures are very flexible since semantics preserving 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 adding or removal of nodes and links in the application or architecture.

Summary

Since the two architectures are similar in performance regarding timing, 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 performed?

6.2 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 (16). Other nodes are marked back-pressure based and are implemented according to net $\mathcal{N}_i$ of Figure 5.

For the links, several cases occur. Homogeneous links, for which all adjacent nodes are of the same kind as that of this link, are implemented according to net $\mathcal{N}_{ji}$ of Figure 4 or net $\mathcal{M}_{ji}$ of Figure 8, depending on the case. 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 its reads at its own pace, according to 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 reads at its own pace, according to 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.

7. EXTENSION TO GENERAL ARCHITECTURES

In this section we indicate how to modify the results of the previous sections when Assumption 4 is relaxed.

7.1 Back-Pressure LTTA

Corresponding study was already performed in [21]. We, however, like to extend our simpler analysis to this case as well. To this end, we simply replace net $\mathcal{N}_{ji}$ of Figure 4 by the net with same name in Figure 12. Observe that the link of Figure 12 captures pipelining. Buffer size must be not smaller than the number of delays. Having done this we can again carry on the study of section 4. Theorem 1 is still valid. On the other hand, Theorem 2 regarding performance must be reformulated as follows.

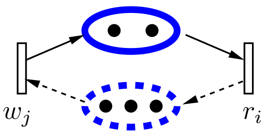

Figure 12: Back-pressure net $\mathcal{N}_{ji}$ associated to a directed link $j \to i$ of the architecture, for the case of a buffer size 5 and 2 delays. For colors and thickness of circles, see figures 4 and 5.

Consider net $\widehat{\mathcal{N}}$ defined as in (9) but with links $\mathcal{N}_{ji}$ redefined according to the principles of Figure 12. In this figure, the two places (depicted in thick blue lines) are assigned a delay $\tau_{\max}$ — this is to capture worst transmission delay, for both data and back-pressure information. Construct net $\widehat{\mathcal{N}}$ according to formula (9). For each circuit σ of net $\widehat{\mathcal{N}}$, define

$$\kappa(\sigma) \ =_{\rm def} \ \frac{\mathbf{T}(\sigma) \times \overline{T}_{\max} + \mathbf{P}(\sigma) \times \overline{\tau}_{\max}}{\text{number of tokens in } \sigma}, \qquad (23)$$

where

$$\begin{array}{lll} \mathbf{T}(\sigma) & =_{\rm def} & \text{number of (read or write) transitions in } \sigma \\ \mathbf{P}(\sigma) & =_{\rm def} & \text{number of thick places in } \sigma \end{array}$$

and $\overline{T}_{\max}$ and $\overline{\tau}_{\max}$ are defined in (13,14). Then, let $\kappa(\widehat{\mathcal{N}})$ be the maximum of all $\kappa(\sigma)$ for σ ranging over all circuits of $\widehat{\mathcal{N}}$. Using the same references as for deriving (12), the following theorem holds, regarding performance:

THEOREM 5. *The worst case throughput $\lambda_{\mathcal{N}}$ of net $\mathcal{N}$ is $1/\lambda_{\mathcal{N}} = \kappa(\widehat{\mathcal{N}})$. The throughput can be computed in $O((n+m)^3)$ where n is the number of nodes in the network and m is the total number of tokens of the event graph.*

7.2 Time-Based LTTA

In this section we relax Assumption 4. However, our study does not encompass pipeling (in contrast to the previous section, which was fully general). The study of pipeling in time-based LTTA requires further investigation.

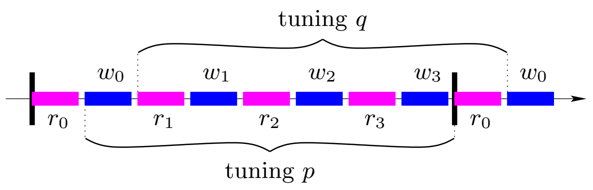

Figure 13: Showing the schedule $r_0 w_0 r_1 w_1 \ldots r_L w_L$ for a reaction, when $L = 3$. The long horizontal directed arrow figures the growth of time. The thick vertical bars indicate barriers separating successive reactions. Compare with Figure 10.

Relaxing Assumption 4 requires retuning parameters p and q of the protocol of section 5, using the notion of *level* introduced in (4). Let L be the maximal level in the considered synchronous application. Then the synchronous semantics is preserved if every reaction follows the schedule $r_0 w_0 r_1 w_1 \ldots r_L w_L$, where r_ℓ and w_ℓ denote the readings and writings by nodes of level ℓ, respectively, see Figure 13. This figure also illustrates the principle for tuning the parameters p and q. Parameter p must ensure that, for a node of level ℓ, reading will keep frozen while $w_\ell, r_{\ell+1}, w_{\ell+1}, \ldots, r_L, w_L, r'_0, w'_0, \ldots, w'_{\ell-1}$ is being performed, where "prime" refers to the next reaction. It is indeed enough to enforce the following properties:

PROPERTY 3 (READS). *The kth firing of transition r_i for a node of level ℓ occurs only after all places $j \in \{1 \ldots n\}$ as in Figure 9 have been written $k-1$ times and places j of level $< \ell$ have all been written k times.*

PROPERTY 4 (WRITES). *The kth firing of one of the (conflicting) transitions $w_i^1 \ldots w_i^q$ for a node of level ℓ occurs only after all transitions $r_j, j \in \{1 \ldots n\}$ have all been fired k times for any node j of level $\leq \ell$.*

Properties 3 and 4 are again proved by induction over $k > 0$. Assume they are true up to $k-1$ and suppose the first $(k-1)$st writing by some node of level ℓ occurs at real-time t. Then the last $(k-1)$st writing by some node occurs at latest at time

$$\min\left(t + qT_{\max}\,,\, t + (L+1)(\tau_{\max} + T_{\max})\right) \qquad (24)$$

Then, the earliest kth reading by a node of level ℓ cannot occur before $t + pT_{\min}$. Hence the following condition

$$p \ \geq \ \frac{(L+1)(\tau_{\max} + T_{\max})}{T_{\min}} \qquad (25)$$

which expands as $pT_{\min} \geq (L+1)(\tau_{\max} + T_{\max})$, ensures that property 3 keeps valid at the kth round. On the other hand, the latest kth reading cannot occur later than

$$\begin{array}{ll} & t + \min\left(qT_{\max}\,,\, \tau_{\max} + T_{\max}\right) + pT_{\max} \\ \leq & t + \tau_{\max} + (p+1)T_{\max} \end{array} \qquad (26)$$

Finally the earliest kth writing by a node of level ℓ cannot occur earlier than $qT_{\min}$ after the earliest kth reading. Hence it cannot occur before $t+pT_{\min}+qT_{\min}$. Hence condition (20), which expands as $qT_{\min} \geq \tau_{\max} + p(T_{\max} - T_{\min})$, ensures that property 4 keeps valid at the kth round. $\diamond$

THEOREM 6. *Conditions (25) and (20) ensure* $\widehat{\mathcal{L}}_{\mathcal{M}} = \mathcal{L}_{\widehat{\mathcal{N}}}$, *which expresses the preservation of synchronous semantics.*

Finally, Theorem 4 regarding performance still holds, but with the values for $p_\star$ and $q_\star$ being given by Theorem 6.

8. 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., [16], TTA cannot be used as the single architectural paradigm in a complex, multi-layered, embedded system.

LTTA was proposed as a softening of TTA for the very same layers. In fact, the objective of LTTA is to offer an abstraction that emulates TTA. This paper has unified the work done on LTTA [21, 8], by proposing a single framework where the two existing variants of LTTA can be cast. This study reveals that the back-pressure based LTTA is more flexible but less robust against failures than time-based LTTA. It makes therefore sense to use different versions of LTTA for different parts of the system. We thus have proposed 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.

9. REFERENCES

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

[2] 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.

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

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

[5] 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.

[6] L. P. Carloni, "The role of back-pressure in implementing latency-insensitive systems," *Electr. Notes Theor. Comput. Sci.*, vol. 146, no. 2, pp. 61–80, 2006.

[7] 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.

[8] P. Caspi and A. Benveniste, "Time-robust discrete control over networked loosely time-triggered architectures," in *IEEE Control and Decision Conference*, December 2008.

[9] J. Cortadella and M. Kishinevsky, "Synchronous elastic circuits with early evaluation and token counterflow," in *DAC*, 2007, pp. 416–419.

[10] 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.

[11] R. Cuninghame-Green, *Minimax Algebras*, ser. Lecture notes in economics and mathematical systems. Springer Verlag, 1979, vol. 166.

[12] M. Gondran and M. Minoux, *Graphs, Dioids and Semirings*, ser. Interface series. Springer Verlag, 2008, vol. 41.

[13] 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.

[14] 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.

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

[16] ——, "The complexity challenge in embedded system design," in *ISORC*, 2008, pp. 3–12.

[17] C. Kossentini and P. Caspi, "Approximation, sampling and voting in hybrid computing systems." in *HSCC*, 2006, pp. 363–376.

[18] 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.

[19] J. Romberg and A. Bauer, "Loose synchronization of event-triggered networks for distribution of synchronous programs." in *EMSOFT*, 2004, pp. 193–202.

[20] P. Traverse, I. Lacaze, and J. Souyris, "Airbus fly-by-wire: A total approach to dependability," in *IFIP World Congress, Toulouse.* IFIP, 2004.

[21] 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.

APPENDIX

A. FROM SYNCHRONOUS MODELS TO KAHN PROCESS NETWORKS

A synchronous model consists of a set of communicating Mealy machines (possibly infinite-state). Structurally, this can be represented as a directed graph the nodes of which correspond to machines and the edges correspond to communication link between two machines: if there is a link $M \to M'$ then an output of M is an input to M'. Since the composition of Mealy machines may not always be well-defined, because of dependency cycles, we assume that every loop in the graph is "broken" by a *unit delay* element (UD). A UD has an initial value which is also the value of its output at the initial instant. At every subsequent instant, the value of its output is equal to the value of its input in the previous instant.

We assume that the model has no *self-loops*, that is, there is no link $M_i \to M_i$ (not even a unit-delay link). This is not a restrictive assumption: such links would need to be unit-delay links (by our initial assumption), therefore, they essentially correspond to part of the internal state of M_i. This internal state does not have to be exposed at the synchronous model level (it can be "hidden" inside M_i).

We define a partial order $\prec$ on the set of machines in a synchronous model, as follows. Given two machines M_i and M_j, if there is a link without unit delay from M_i to M_j then $M_i \prec M_j$. We then complete $\prec$ with its reflexive and transitive closure. From the assumption that every loop in the synchronous model is broken by a unit delay, $\prec$ is indeed a partial order. M_i is a *minimal element* with respect to $\prec$ if there is no $M_j \neq M_i$ such that $M_j \prec M_i$.

For simplicity, we will assume that all streams in the model take values in the same domain of values V. Under this assumption we give two semantics to this model and then we show that they are equivalent.

Strictly synchronous semantics

For the first semantics, we regard each machine M_i as a *transition function*

$$F_i \ : \ V^{n_i} \times S_i \to V^{m_i} \times S_i, \tag{27}$$

where n_i and m_i are the numbers of inputs and outputs of M_i, respectively, and S_i is the set of states of this machine. Thus, the machine maps the tuple (input, state) to the tuple (output, next state). S_i may be infinite. S_i may also be a singleton, modeling a "memoryless" or "combinational" machine. The unit-delay element is just a particular case of the above general form (27), namely:

$$s'_{\mathrm{UD}} = x \, , \, y = s_{\mathrm{UD}}$$

where x, y, and s_{UD} are the input, output, and internal state of the delay. An initial value $s_{0,i}$ for local state s_i is provided. Note that the output y depends only on the current state s_{UD}, not on the current input x.

Performing a nonterminating while loop of steps of the form (27), fed by the successive values of the inputs, yields the following model for machine M_i:

$$M_i \ : \ (V^{n_i})^{\mathbb{N}} \times S_i \to (V^{m_i})^{\mathbb{N}} \tag{28}$$

Transition functions $(F_i)_{i \in I}$ compose by matching outputs of some transition functions to inputs of others according to the network specified by the synchronous model of machines. States remain local. Note that, since self-loops were forbidden, the directed graph showing the output-to-input connections is broken by UD delay operators. Hence, the so obtained composition $\|_{i \in I} F_i$ of the transition functions $F_i, i \in I$ yields another transition function F of the form (27), which in turn induces a machine

$$M = \|_{i \in I} M_i \tag{29}$$

in the same way stream equation M_i derives from transition function F_i via (28). Initial states of M are the tuples $(s_{0,i})_{i \in I}$ and inputs of M are those inputs of the components M_i that are not connected to an output of another machine.

Assuming the functions M_i of each machine are computable, this solution can be computed up to any k, by repeatedly "firing" the machines in a statically specified order. This order is any total order that respects the partial order $\prec$.

Kahn Process Network semantics

For the second semantics, we regard directly each machine as a function mapping tuples of input streams and initial state, to tuples of output streams:

$$M_i^K \ : \ \left(V^{\mathbb{N}}\right)^{n_i} \times S_i \to \left(V^{\mathbb{N}}\right)^{m_i} \tag{30}$$

Thus our synchronous models are just systems of stream equations of the form (30) that we interpret according to the Kahn Process Network semantics [14]. The latter yields a parallel composition that we denote by:

$$M^K = \|_{i \in I}^K M_i^K \tag{31}$$

The following result proves equivalence of the two semantics:

THEOREM 7. *The two semantics are equivalent in that M defined in (29) identifies with M^K defined in (31). Furthermore, synchronous models possess a Kahn Process Network semantics that can be implemented with FIFOs of size ≤ 2.*

PROOF. By construction, $M_i = M_i^K$ holds for each i. Since $F = \|_{i \in I} F_i$ is the parallel composition of the transition functions, then M associated with F via (28) is a solution of the system of stream equations defined by the $M_i = M_i^K$, i.e., a Kahn Process Network semantics for the family $(M_i^K)_{i \in I}$ of processes. Since the Kahn Process Network semantics is unique, the first statement of the theorem follows.

For the second statement, observe that 1/ the semantics of M is obtained by executing $F = \|_{i \in I} F_i$ repeadtedly in loop, and 2/ executing F requires at most two buffers for storing previous and current state. $\diamond$

COROLLARY 1. *Every synchronous model can be implemented in a semantics-preserving way on a Kahn Process Network equipped with queues of size ≤ 2.*